Network Working Group	D. Meyer
Internet-Draft	Universitaet Bremen TZI
Intended status: Standards Track	P. Saint-Andre
Expires: January 1, 2010	Cisco
	June 30, 2009

TOC

XTLS: End-to-End Encryption for the Extensible Messaging and Presence Protocol (XMPP) Using Transport Layer Security (TLS)
draft-meyer-xmpp-e2e-encryption-02

Status of this Memo

Copyright Notice

Abstract

This document specifies "XTLS", a protocol for end-to-end encryption of Extensible Messaging and Presence Protocol (XMPP) traffic. XTLS is an application-level usage of Transport Layer Security (TLS) that is set up using the XMPP Jingle extension for session negotiation and

transported using any streaming transport as the data delivery mechanism. Thus XTLS treats the end-to-end exchange of XML stanzas as a virtual transport and uses TLS to secure that transport, enabling XMPP entities to communicate in a way that is designed to ensure the confidentiality and integrity XML stanzas. The protocol can be used for secure end-to-end messaging as well as other XMPP applications, such as file transfer.

Table of Contents

---

1. Introduction

TOC

End-to-end encryption of traffic sent over the Extensible Messaging and Presence Protocol (XMPP) is a desirable goal. Requirements and a threat analysis for XMPP encryption are provided in [E2E-REQ] (Saint-Andre, P., “Requirements for End-to-End Encryption in the Extensible Messaging and Presence Protocol (XMPP),” June 2009.). This document explores the possibility of using the Transport Layer Security [TLS] (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol Version 1.2,” August 2008.) to meet those requirements.

TLS is the most widely implemented protocol for securing network traffic. In addition to applications in the email infrastructure, the World Wide Web [HTTP-TLS] (Rescorla, E., “HTTP Over TLS,” May 2000.),

and datagram transport for multimedia session negotiation [DTLS] (Rescorla, E. and N. Modadugu, “Datagram Transport Layer Security,” April 2006.), TLS is used in XMPP to secure TCP connections from client to server and from server to server, as specified in [XMPP-CORE] (Saint-Andre, P., “Extensible Messaging and Presence Protocol (XMPP): Core,” June 2009.). Therefore TLS is already familiar to XMPP developers.

This specification, called "XTLS", defines a method whereby any XMPP entity that supports the XMPP Jingle negotiation framework [JINGLE] (Ludwig, S., Beda, J., Saint-Andre, P., McQueen, R., Egan, S., and J. Hildebrand, “Jingle,” June 2009.) can use TLS semantics for end-to-end encryption, whether the application data is sent over a streaming transport (like TCP) or a datagram transport (like UDP). The basic use case is to tunnel XMPP stanzas between two IM users for end-to-end secure chat using end-to-end XML streams. However, XTLS is not limited to encryption of one-to-one text chat, since it can be used between two XMPP clients for encryption of any XMPP payloads, between an XMPP client and a remote XMPP service (i.e., a service with which a client does not have a direct XML stream, such as a [MUC] (Saint-Andre, P., “Multi-User Chat,” July 2008.) chatroom), or between two remote XMPP services. Furthermore, XTLS can be used for encrypted file transfer using [JINGLE-FILE] (Saint-Andre, P., “Jingle File Transfer,” February 2009.), for encrypted voice or video sessions using [JINGLE-RTP] (Ludwig, S., Saint-Andre, P., Egan, S., McQueen, R., and D. Cionoiu, “Jingle RTP Sessions,” June 2009.) and [DTLS-SRTP] (McGrew, D. and E. Rescorla, “Datagram Transport Layer Security (DTLS) Extension to Establish Keys for Secure Real-time Transport Protocol (SRTP),” February 2009.), and other applications.

Note: The following capitalized keywords are to be interpreted as described in [TERMS] (Bradner, S., “Key words for use in RFCs to Indicate Requirement Levels,” March 1997.): "MUST", "SHALL", "REQUIRED"; "MUST NOT", "SHALL NOT"; "SHOULD", "RECOMMENDED"; "SHOULD NOT", "NOT RECOMMENDED"; "MAY", "OPTIONAL".

---

2. Approach

TOC

In broad outline, XTLS takes the following approach to end-to-end encryption of XMPP traffic:

1. We assume that all XMPP entities will have X.509 certificates; realistically these certificates are likely to be self-signed and automatically generated by an XMPP client, however certificates issued by known certification authorities are encouraged to overcome problems with self-signed certificates.

2. We use the XMPP Jingle extensions as the negotiation framework (see [JINGLE] (Ludwig, S., Beda, J., Saint-Andre, P., McQueen, R., Egan, S., and J. Hildebrand, “Jingle,” June 2009.)).

3. We use the concept of Jingle security preconditions to ensure that the negotiated transport will be encrypted before used for sending application data.

4. When an entity wishes to encrypt its communications with a second entity, it sends a Jingle session-initiate request that specifies the desired application type, a possible transport, and a TLS security precondition that includes the sender's X.509 fingerprint and optionally hints about the sender's supported TLS methods.

5. If both parties support XTLS, the first data sent over the negotiated transport is TLS handshake data, not application data. Once the TLS handshake has finished, the parties can then send application data over the now-encrypted transport (called an "XTLS tunnel").

6. The simplest scenario is end-to-end encryption of traditional XMPP text chat using end-to-end XML streams as the application and in-band bytestreams [IBB] (Karneges, J., “In-Band Bytestreams (IBB),” March 2009.) as the transport.

7. If the parties have previously negotiated an XTLS tunnel, during the TLS negotiation each party simply needs to verify that the other party is presenting the same certificate as used in previous sessions.

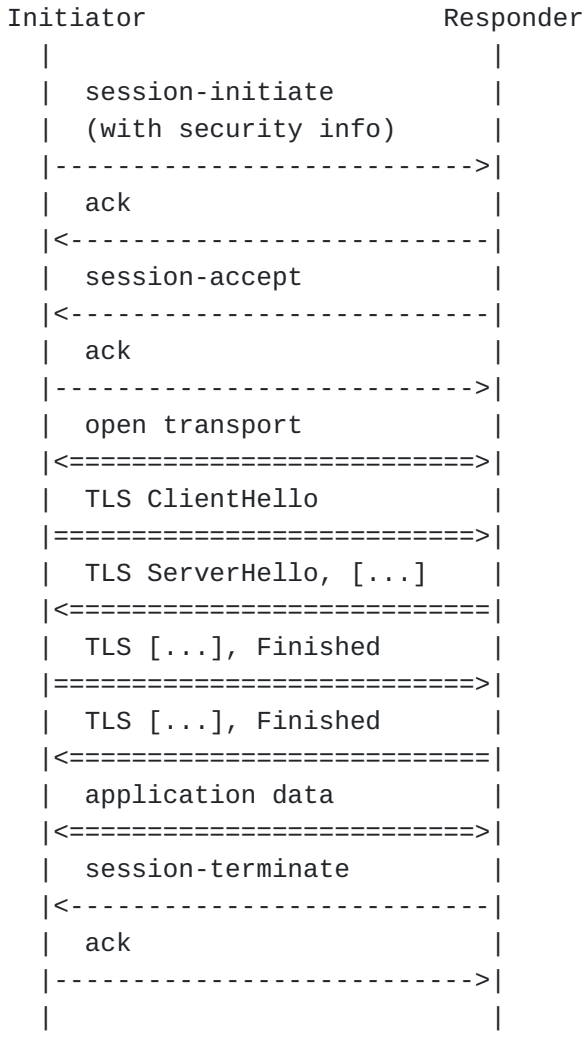
8. If the parties have not previously negotiated an XTLS tunnel, they need to bootstrap trust in their certificates; to do so, it is encouraged to use secure remote passwords rather than leap-of-faith.

We expand on this approach in the following section.
More complex scenarios are theoretically supported (e.g., encrypted file transfer using SOCKS5 bytestreams and encrypted voice chat using DTLS-SRTP) but have not yet been fully defined.
XTLS theoretically can be used to establish a TLS-encrypted streaming transport or a DTLS-encrypted datagram transport, but integration with DTLS [DTLS] (Rescorla, E. and N. Modadugu, “Datagram Transport Layer Security,” April 2006.) has not yet been prototyped so use with streaming transports is the more stable scenario.

TOC

3. XTLS Protocol Flow

The basic flow for an XTLS session is as follows, where traffic represented by single dashes (---) is sent over the XMPP signalling channel and traffic represented by double lines (===) is sent over the negotiated transport.



To simplify the description we assume here that the parties already trust each other's certificates. See discussion under Section 5 (Bootstrapping Trust on First Communication) for information about bootstrapping of certificate trust when the parties first negotiate the use of an XTLS tunnel.

First the initiator sends a Jingle session-initiate request (here the simple case of an end-to-end text chat session using in-band bytestreams [IBB] (Karneges, J., “In-Band Bytestreams (IBB),” March 2009.)). This request includes a <security/> element that contains the fingerprint of the certificate that the initiator will use during the TLS negotiation and a list of TLS methods the initiator supports (here certificate-based authentication [X509] (Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk,

"Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," May 2008.) and TLS with Secure Remote Passwords [TLS-SRP] (Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, "Using the Secure Remote Password (SRP) Protocol for TLS Authentication," November 2007.)). Note that this information is exchanged over the insecure server-based connection. The purpose of the exchange is to gather information about which TLS method should be used in the TLS handshake, e.g. if a client cannot verify the fingerprint of the peer it MAY omit the X.509 method. If both clients can verify the fingerprint of the other, it is likely that X.509 certificate-based authentication will succeed (unless the data is altered); if one client cannot verify the fingerprint the client MAY prompt the user for a password for TLS-SRP based authentication (see Section 5 (Bootstrapping Trust on First Communication) for details).

```
<iq from='romeo@montague.lit/orchard'
    id='xn28s7gk'
    to='juliet@capulet.lit/balcony'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'>
         action='session-initiate'
         initiator='romeo@montague.lit/orchard'
         sid='a73sjjvkla37jfea'>
    <content creator='initiator' name='xmlstream'>
      <description xmlns='urn:xmpp:jingle:apps:xmlstream:0'/>
      <transport xmlns='urn:xmpp:jingle:transports:ibb:0'
                 block-size='4096'
                 sid='ch3d9s71'/>
      <security xmlns='urn:xmpp:jingle:security:xtls:0'>
        <fingerprint algo='sha1'>RomeoX509CertSHA1Hash</fingerprint>
        <method name='x509'/>
        <method name='srp'/>
      </security>
    </content>
  </jingle>
</iq>
```

The responder immediately acknowledges receipt of the session-initiate by sending an IQ stanza of type "result" (not shown here). Depending on the application type, a user agent controlled by a human user might need to wait for the user to affirm a desire to proceed with the session before continuing. When the user agent has received such affirmation (or if the user agent can automatically proceed for any reason, e.g. because no human intervention is expected or because a human user has configured the user agent to automatically accept sessions with a given entity), it returns a Jingle session-accept message. This message will typically contain the offered application type, transport method, and a <security/> element that includes the

fingerprint of the responder's X.509 certificate as well as the responder's supported TLS methods.

```
<iq from='juliet@capulet.com/balcony'
    id='hf64hl'
    to='romeo@montague.net/orchard'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'>
         action='session-accept'
         initiator='romeo@montague.lit/orchard'
         sid='a73sjjvkla37jfea'>
    <content creator='initiator' name='xmlstream'>
      <description xmlns='urn:xmpp:jingle:apps:xmlstream:0'/>
      <transport xmlns='urn:xmpp:jingle:transports:ibb:0'
                 block-size='4096'
                 sid='ch3d9s71'/>
      <security xmlns='urn:xmpp:jingle:security:xtls:0'/>
        <fingerprint algo='sha1'>JulietX509CertSHA1Hash</fingerprint>
        <method name='x509'/>
        <method name='srp'/>
      </security>
    </content>
  </jingle>
</iq>
```

The following rules apply to the responder's handling of the session-initiate message:

1. If the responder does not support XTLS it will silently ignore the <security/> element in the offer and therefore will return a session-accept message without a <security/> element.

2. If the responder supports XTLS it MUST return a session-accept message that contains a <security/> element.

3. If the responder thinks it will be able to verify the initiator's certificate, it MUST include the fingerprint for the responder's certificate in the <security/> element of the session-accept message. This is the "happy path" and will occur when the parties have already verified each other's certificates.

4. If the responder thinks it will not be able to verify the initiator's certificate, it MAY omit the fingerprint for the responder's certificate in the <security/> element of the session-accept message. This indicates that certificate-based authentication is not possible. In this case the responder SHOULD signal that it wishes to use some other authentication

method, such as secure remote passwords (see discussion under Section 5 (Bootstrapping Trust on First Communication)).

5. If the responding client cannot verify the initiator's certificate, it SHOULD ask the responding user if a password was exchanged between the parties that can be used for TLS-SRP. If this is not the case, setting up a mutually-authenticated link will fail and the responder MAY terminate the session. Alternatively it could send its own fingerprint knowing it cannot authenticate the initiator, in which case the responder has to trust that there is no man-in-the-middle (see discussion under Section 5 (Bootstrapping Trust on First Communication)).

When the responder sends the session-accept message, the initiator acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
The following rules apply to the initiator's handling of the session-accept message:

1. If the initiator receives a session-accept without a <security/> element, setting up a secure transport layer has failed. The initiator MAY terminate the session at this point or instead proceed without securing the transport. The client SHOULD ask the initiating user how to processed. This depends on the Jingle application and the initiator's preferences: it makes no sense to use end-to-end XML streams without encryption, but the initiator might continue a file transfer without encryption.

2. If the initiating client cannot verify the responder's certificate it SHOULD ask the initiating user if a password was exchanged between the parties that can be used for TLS-SRP. If this is not the case, setting up a mutually-authenticated link will fail and the responder MAY terminate the session or proceed with leap-of-faith (see discussion under Section 5 (Bootstrapping Trust on First Communication)).

The initiator can now determine if X.509 certificate-based authentication will work or if TLS-SRP will be used. It sends an additional security-info message to the responder to signal its choice. This step is not really necessary because the responder will see the initiator's choice in the first message of the TLS handshake, but it can assist an implementation in setting up its TLS library properly. Because in this section we assume that the parties already have validated each other's certificates, the security method signalled here is "x509".

```
<iq from='romeo@montague.lit/orchard'
    id='hf749j'
    to='juliet@capulet.lit/balcony'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'>
         action='security-info'
         initiator='romeo@montague.lit/orchard'
         sid='a73sjjvkla37jfea'>
    <content creator='initiator' name='xmlstream'>
      <security xmlns='urn:xmpp:jingle:security:xtls:0'>
        <method name='x509'/>
      </security>
    </content>
  </jingle>
</iq>
```

The responder acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
Parallel to the security-info exchange, the clients negotiate a transport for the Jingle session (here the transport is an in-band bytestream as defined in [IBB] (Karneges, J., “In-Band Bytestreams (IBB),” March 2009.), for which the Jingle negotiation process is specified in [XEP-0261] (Saint-Andre, P., “Jingle In-Band Bytestreams Transport,” February 2009.); however other transports could be used, for example SOCKS5 bytestreams as defined in [XEP-0065] (Smith, D., Miller, M., and P. Saint-Andre, “SOCKS5 Bytestreams,” May 2007.) and negotiated for Jingle as specified in [XEP-0260] (Saint-Andre, P. and D. Meyer, “Jingle SOCKS5 Bytestreams Transport Method,” February 2009.)). Because the parties wish to establish end-to-end encryption, they do not send application data over the transport until the transport has been secured. Therefore the first data that they exchange over the transport consists of the standard four-way TLS handshake, encoded in accordance with the negotiated transport method.

> Note: Each transport MUST define a specific time when both clients know that the transport is secured. When XTLS is not used, the Jingle implementation would signal to the using application that the transport is open when the session-accept is sent or received, or when connectivity checks determine media can flow over one of the transport candidates. When XTLS is used, the Jingle implementation starts a TLS handshake on the transport and signals to the using application that the transport is open only after the TLS handshake has finished successfully.

During the TLS handshake, the responder MUST take the role of the TLS server and the initiator MUST take the role of the TLS client. Because the transport is an in-band bytestream, the TLS handshake data is prepared as described in [IBB] (Karneges, J., “In-Band Bytestreams

(IBB)," March 2009.) (i.e., Base64-encoded). First the initiator (acting as the TLS client) constructs a TLS ClientHello, encodes it according to IBB, and sends it to the responder.

```
<iq from='romeo@montague.net/orchard'
    id='vh38s618'
    to='juliet@capulet.com/balcony'
    type='set'>
  <data xmlns='http://jabber.org/protocol/ibb'
        seq='0'
        sid='vj3hs98y'>
    Base64-encoded-TLS-data
  </data>
</iq>
```

The responder (acting as the TLS server) then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
The responder then constructs an appropriate TLS message or messages, such as a ServerHello and a CertificateRequest.

> Note: The responder MUST send a CertificateRequest to the initiator.

```
<iq from='juliet@capulet.com/balcony'
    id='xyw516d0'
    from='romeo@montague.net/orchard'
    type='set'>
  <data xmlns='http://jabber.org/protocol/ibb'
        seq='0'
        sid='vj3hs98y'>
    Base64-encoded-TLS-data
  </data>
</iq>
```

(Because in-band bytestreams are bidirectional and this data is sent from the responder to the initiator, the IBB 'seq' attribute has a value of zero, not 1.)
The initiator then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
After some number of TLS messages, the initiator eventually sends a TLS Finished message to the responder.

```
<iq from='romeo@montague.net/orchard'
    id='s91vd527'
    to='juliet@capulet.com/balcony'
    type='set'>
  <data xmlns='http://jabber.org/protocol/ibb'
        seq='3'
        sid='vj3hs98y'>
    Base64-encoded-TLS-data
  </data>
</iq>
```

The responder then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
The responder then also sends a TLS Finished message.

```
<iq from='juliet@capulet.com/balcony'
    id='z71gs73t'
    from='romeo@montague.net/orchard'
    type='set'>
  <data xmlns='http://jabber.org/protocol/ibb'
        seq='3'
        sid='vj3hs98y'>
    Base64-encoded-TLS-data
  </data>
</iq>
```

The initiator then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
If the TLS negotiation has finished successfully, then the Jingle implementation shall signal to the using application that the transport has been secured and is ready to be used. The parties now have a secure channel for the end-to-end exchange of application data using XMPP as the virtual transport; we call such a channel an XTLS TUNNEL.

---

4. End-to-End Streams over XTLS Protocol Flow

TOC

For end-to-end encryption of XMPP stanzas (<message/>, <presence/>, and <iq/>), the application data is an end-to-end XML stream. After the XTLS tunnel is established, the peers open an XML stream over the tunnel to exchange stanzas. In this example, the tunnel is established using a transport of IBB, but any streaming transport could be used.
First the initiator constructs an initial stream header.

```
<stream:stream
        xmlns='jabber:client'
        xmlns:stream='http://etherx.jabber.org/streams'
        from='romeo@montague.lit/orchard'
        to='juliet@capulet.lit/balcony'
        version='1.0'>
```

Note: In accordance with [XMPP-CORE] (Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core," June 2009.), the initial stream header SHOULD include the 'to' and 'from' attributes, which SHOULD specify the full JIDs of the clients. The initiator SHOULD include the version='1.0' flag as shown in the previous example.
The initiator then transforms the stream header into TLS data, encodes the data into IBB, and sends an IQ-set to the responder.

```
<iq from='romeo@montague.net/orchard'
    id='ur73n153'
    to='juliet@capulet.com/balcony'
    type='set'>
  <data xmlns='http://jabber.org/protocol/ibb'
        seq='4'
        sid='vj3hs98y'>
    Base64-TLS-data-of-the-stream-header
  </data>
</iq>
```

The responder then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
The responder then constructs a response stream header back to the initiator.

```
<stream:stream
        xmlns='jabber:client'
        xmlns:stream='http://etherx.jabber.org/streams'
        from='juliet@capulet.lit/balcony'
        id='hs91gh1836d8s717'
        to='romeo@montague.lit/orchard'
        version='1.0'>
```

The responder then sends the response stream header over the XTLS tunnel.

```
     <iq from='juliet@capulet.com/balcony'
         id='pd61g397'
         to='romeo@montague.net/orchard'
         type='set'>
       <data xmlns='http://jabber.org/protocol/ibb'
             seq='4'
             sid='vj3hs98y'>
         Base64-TLS-data-of-the-responce-stream-header
       </data>
     </iq>
```

The initiator then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
Once the XML stream is established over the XTLS tunnel, either entity then can send XMPP message, presence, and IQ stanzas, with or without 'to' and 'from' addresses.
For example, the initiator could construct an XMPP message.

```
     <message from='romeo@montague.lit/orchard'
              to='juliet@capulet.lit/balcony'>
       <body>
         M&amp;apos;lady, I would be pleased to make your acquaintance.
       </body>
     </message>
```

The initiator then sends the message over the XTLS tunnel.

```
     <iq from='romeo@montague.net/orchard'
         id='iq7dh294'
         to='juliet@capulet.com/balcony'
         type='set'>
       <data xmlns='http://jabber.org/protocol/ibb'
             seq='5'
             sid='vj3hs98y'>
         Base64-TLS-data
       </data>
     </iq>
```

The responder then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
The responder could then construct a reply.

```
     <message from='juliet@capulet.lit/balcony'
              to='romeo@montague.lit/orchard'>
       <body>Art thou not Romeo, and a Montague?</body>
     </message>
```

The responder then sends the reply over the XTLS tunnel.

```
     <iq from='juliet@capulet.com/balcony'
         id='hr91hd63'
         to='romeo@montague.net/orchard'
         type='set'>
       <data xmlns='http://jabber.org/protocol/ibb'
             seq='5'
             sid='vj3hs98y'>
         Base64-TLS-data
       </data>
     </iq>
```

The initiator then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
To close the end-to-end XML stream, either party (here the responder) constructs a closing </stream:stream> element.

```
     </stream:stream>
```

The client sends the closing element to the peer over the XTLS tunnel.

```
     <iq from='juliet@capulet.com/balcony'
         id='kr91n475'
         to='romeo@montague.net/orchard'
         type='set'>
       <data xmlns='http://jabber.org/protocol/ibb'
             seq='6'
             sid='vj3hs98y'>
         Base64-TLS-data
       </data>
     </iq>
```

The peer then acknowledges receipt by sending an IQ stanza of type "result" (not shown here).
However, even after the end-to-end XML stream is terminated, the negotiated Jingle transport (here an in-band bytestream) continues and could be re-used. To completely terminate the Jingle session, the terminating party would then also send a Jingle session-terminate message.

```
     <iq from='juliet@capulet.lit/balcony'
         id='psy617r4'
         to='romeo@montague.lit/orchard'
         type='set'>
       <jingle xmlns='urn:xmpp:jingle:1'
               action='session-terminate'
               initiator='romeo@montague.lit/orchard'
               sid='851ba2'/>
     </iq>
```

The other party then acknowledges the Jingle session-terminate by sending an IQ stanza of type "result" (not shown here).

---

5. Bootstrapping Trust on First Communication

TOC

When two parties first attempt to use XTLS, their certificates might not be accepted (e.g., because they are self-signed or issued by unknown certification authorities). Therefore each party needs to accept the other's certificate for use in future communication sessions. There are several ways to do so:

* Leap of faith. The recipient can hope that there is no man-in-the-middle during the first communication session. If the certificate does not change in future sessions, the recipient at least knows that it is talking with the same entity it talked with during the first session. However, that entity might be a man-in-the-middle rather than the assumed communication partner. Therefore, leap of faith is discouraged.

* Check fingerprints. The parties could validate the certificate fingerprints via some trusted means outside the XMPP band, such as in person, via encrypted email, or over the phone. This is not user-friendly because certificate fingerprints consist of long strings of letters and numbers. As a result, few humans routinely check certificate fingerprints in protocols such as Secure Shell (ssh).

* One-time password. The parties can exchange a user-friendly password known only to themselves and verify it out of band before the TLS handshake finishes. For this purpose, it is REQUIRED for implementations to support at least one TLS cipher that uses Secure Remote Password (SRP) as defined in [TLS-SRP] (Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, “Using the Secure Remote Password (SRP) Protocol for TLS Authentication,” November 2007.).

* Channel binding. It is possible that a future version of this specification will describe how to use an appropriate Simple Authentication and Security Layer (SASL) mechanism, such as [SCRAM] (Menon-Sen, A., Melnikov, A., Newman, C., and N. Williams, “Salted Challenge Response (SCRAM) SASL Mechanism,” May 2009.), to authenticate the XTLS tunnel after the TLS handshake finishes; such a method would use the concept of channel bindings as described in [RFC5056] (Williams, N., “On the Use of Channel Bindings to Secure Channels,” November 2007.).

If the parties use a password or SASL channel binding to bootstrap trust, the process needs to be completed only once. After the clients have authenticated with the shared secret, they can exchange their certificates for future communication.

5.1. Exchanging Certificates

TOC

To retrieve the certificate of the peer for future communications, a client SHOULD request the certificate according to [XEP-0189] (Paterson, I., Saint-Andre, P., and D. Meyer, “Public Key Publishing,” March 2009.) over the secure connection. This works only if XTLS was used to set up an end-to-end secure XML stream; exchanging certificates if XTLS was used for other purposes like file transfer is not possible. A client MUST NOT request the certificate over the insecure stream-based on the connection to the XMPP server.

```
<iq from='romeo@montague.lit/orchard'
    id='hf7634k4'
    to='juliet@capulet.lit/balcony'
    type='get'>
  <pubkeys xmlns='urn:xmpp:pubkey:0'/>
</iq>
```

The peer MUST return its own client certificate. If the user has different clients with different client certificates and one user certificate, the user certificate SHOULD also be returned. The user certificate allows it to verify other client certificates using public key retrieval as described in [XEP-0189] (Paterson, I., Saint-Andre, P., and D. Meyer, “Public Key Publishing,” March 2009.).

```
<iq from='juliet@capulet.com/balcony'
    id='hf7634k4'
    to='romeo@montague.lit/orchard'
    type='result'>
  <pubkeys xmlns='urn:xmpp:pubkey:0'>
    <keyinfo>
      <x509cert>
MIICCTCCAXKgAwIBAgIJALhU0Id6xxwQMA0GCSqGSIb3DQEBBQUAMA4xDDAKBgNV
BAMTA2ZvbzAeFw0wNzEyMjgyMDA1MTRaFw0wODEyMjcyMDA1MTRaMA4xDDAKBgNV
BAMTA2ZvbzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEA0DPcfeJzKWLGE22p
RMINLKr+CxqozF14DqkXkLUwGzTqYRi49yK6aebZ9ssFspTTjqa2uNpw1U32748t
qU6bpACWHbcC+eZ/hm5KymXBhL3Vjfb/dW0xrtxjI9JRFgrgWAyxndlNZUpN2s3D
hKDfVgpPSx/Zp8d/ubbARxqZZZkCAwEAAaNvMG0wHQYDVR0OBBYEFJWwFqmSRGcx
YXmQfdF+XBWkeML4MD4GA1UdIwQ3MDWAFJWwFqmSRGcxYXmQfdF+XBWkeML4oRKk
EDAOMQwwCgYDVQQDEwNmb2+CCQC4VNCHesccEDAMBgNVHRMEBTADAQH/MA0GCSqG
SIb3DQEBBQUAA4GBAIhlUeGZ0d0msNVxYWAXg2lRsJt9INHJQTCJMmoUeTtaRjyp
ffJtuopguNNBDn+MjrEp2/+zLNMahDYLXaTVmBf6zvY0hzB9Ih0kNTh23Fb5j+yK
QChPXQUo0EGCaODWhfhKRNdseUozfNWOz9iTgMGw8eYNLllQRL//iAOfOr/8
      </x509cert>
    </keyinfo>
  </pubkeys>
</iq>
```

5.2. Verification of Non-Human Parties

TOC

If one of the parties is a "bot" (e.g., an automated service or a device such as a set-top box), the password exchange is a bit more complicated. It is similar to Bluetooth peering if the user has access to both clients at the same time. One of the following scenarios might apply:

- The bot can be controlled via a remote control input device. The human user can enter the same password or "PIN" on both the bot and the XMPP client.

- If the bot has no user input but does have a small display, it could display a random password. The human user can then enter the provided password on the XMPP client.

- The bot might not have enough buttons for input and might not have an output screen. In that case the password is fixed. Similar to Bluetooth peering with simple devices such as a headset, the password will be written in the manual or printed on the device. For security reasons the device SHOULD NOT use password-based authentication without any user input. Many

Bluetooth devices have at least one button to set the device into peering mode.

*A bot may be associated with a web service and could display a random password when the user has logged in to the web site using HTTPS. This assumes that an attacker cannot at the same time both control over the web server and perform a man-in-the-middle attack on the XMPP channel. If the web service knows the GPG key of the user it could send an encrypted email.

A user might have different X.509 certificates for each device. [XEP-0189] (Paterson, I., Saint-Andre, P., and D. Meyer, "Public Key Publishing," March 2009.) can be used to manage the user's certificates. A client SHOULD check the peer's PubSub node for certificates. This makes it possible to use the password method only once between two users even if one or both users switch clients. A user can also communicate with a friend's bots: they first open a secure link between two chat clients with a password and exchange the user certificates. After that each device of a user can verify all devices of the other without the need of a password.
The retrieved certificate from the PubSub node might be signed by a certification authority that the client can verify. In that case the client MAY skip the password authentication and rely on the X.509 certificate chain. The client SHOULD ask the user if the certificate is acceptable or if a password exchange is desired.

---

6. Session Termination

TOC

If either client cannot verify the certificate of the peer or receives an invalid message on the TLS layer, it MUST terminate the Jingle session immediately by sending a Jingle session-terminate message that includes a Jingle reason of <security-error/>.

```
<iq from='romeo@montague.lit/orchard'
    id='hz81vf48'
    to='juliet@capulet.lit/balcony'
    type='set'>
  <jingle xmlns='urn:xmpp:jingle:1'
          action='session-terminate'
          initiator='romeo@montague.lit/orchard'
          sid='a73sjjvkla37jfea'>
    <reason><security-error/></reason>
  </jingle>
</iq>
```

The other party then acknowledges the session-terminate by sending an IQ stanza of type "result" (not shown here), and the Jingle session is finished.

---

7. Determining Support

TOC

If an entity wishes to request the use of XTLS, it SHOULD first determine whether the intended responder supports the protocol. This can be done directly via [XEP-0030] (Hildebrand, J., Millard, P., Eatmon, R., and P. Saint-Andre, “Service Discovery,” June 2008.) or indirectly via [XEP-0115] (Hildebrand, J., Saint-Andre, P., Tronçon, R., and J. Konieczny, “Entity Capabilities,” February 2008.).
If an entity supports XTLS, it MUST report that by including a service discovery feature of "urn:xmpp:jingle:security:xtls:1" in response to disco#info requests.

```
<iq from='romeo@montague.lit/orchard'
    id='disco1'
    to='juliet@capulet.lit/chamber'
    type='get'>
  <query xmlns='http://jabber.org/protocol/disco#info'/>
</iq>


<iq from='juliet@capulet.lit/chamber'
    id='disco1'
    to='romeo@montague.lit/orchard'
    type='result'>
  <query xmlns='http://jabber.org/protocol/disco#info'>
    <feature var='urn:xmpp:jingle:security:xtls:1'/>
    <feature var='urn:xmpp:jingle:apps:xmlstream:1'/>
  </query>
</iq>
```

Both service discovery and entity capabilities information could be corrupted or intercepted; for details, see under Section 8.3 (Denial of Service).

---

8. Security Considerations

TOC

This entire document addresses security. Particular security-related issues are discussed in the following sections.

8.1. Mandatory-to-Implement Technologies

TOC

An implementation MUST at a minimum support the "srp" and "x509" methods. A future version of this specification will document mandatory-to-implement TLS ciphers.

8.2. Certificates

TOC

As noted, XTLS can be used between XMPP clients, between an XMPP client and a remote XMPP service (i.e., a service with which a client does not have a direct XML stream), or between remote XMPP services. Therefore, a party to an XTLS bytestream will present either a client certificate or a server certificate as appropriate. Such certificates MUST be generated and validated in accordance with the certificate guidelines guidelines provided in [XMPP-CORE] (Saint-Andre, P., “Extensible Messaging and Presence Protocol (XMPP): Core,” June 2009.).
A future version of this specification might provide additional guidelines regarding certificate validation in the context of client-to-client encryption.

8.3. Denial of Service

TOC

Currently XMPP stanzas such as Jingle negotiation messages and service discovery exchanges are not encrypted or signed. As a result, it is possible for an attacker to intercept these stanzas and modify them, thus convincing one party that the other party does not support XTLS and therefore denying the parties an opportunity to use XTLS.
This is a more general problem with XMPP technologies and needs to be addressed at the core XMPP layer.

---

9. IANA Considerations

TOC

It might be helpful to create a registry of TLS methods that can be used in the context of XTLS (e.g., "openpgp" for use of [RFC5081] (Mavrogiannopoulos, N., “Using OpenPGP Keys for Transport Layer Security (TLS) Authentication,” November 2007.), "srp" for use of [TLS-SRP] (Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, “Using the Secure Remote Password (SRP) Protocol for TLS Authentication,” November 2007.), and "x509" for use of [TLS] (Dierks, T. and E. Rescorla, “The Transport Layer Security (TLS) Protocol

Version 1.2," August 2008.) with certificates). The registry could be maintained by the IANA or by the XMPP Registrar (see [XEP-0053] (Saint-Andre, P., "XMPP Registrar Function," October 2008.)). A future version of this specification will provide more detailed information about the registration requirements.

10. References

TOC

10.1. Normative References

TOC

[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), June 2009 (TXT).
[TERMS]	Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).
[TLS]	Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2," RFC 5246, August 2008 (TXT).
[IBB]	Karneges, J., "In-Band Bytestreams (IBB)," XSF XEP 0047, March 2009.
[JINGLE]	Ludwig, S., Beda, J., Saint-Andre, P., McQueen, R., Egan, S., and J. Hildebrand, "Jingle," XSF XEP 0166, June 2009.
[XMPP-CORE]	Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core," draft-ietf-xmpp-3920bis-00 (work in progress), June 2009 (TXT).

10.2. Informative References

TOC

[DTLS]	Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security," RFC 4347, April 2006 (TXT).
[DTLS-SRTP]	McGrew, D. and E. Rescorla, "Datagram Transport Layer Security (DTLS) Extension to Establish Keys for Secure Real-time Transport Protocol (SRTP)," draft-ietf-avt-dtls-srtp-07 (work in progress), February 2009 (TXT).
[HTTP-TLS]	Rescorla, E., "HTTP Over TLS," RFC 2818, May 2000 (TXT).
[JINGLE-FILE]	Saint-Andre, P., "Jingle File Transfer," XSF XEP 0234, February 2009.
[JINGLE-RTP]	Ludwig, S., Saint-Andre, P., Egan, S., McQueen, R., and D. Cionoiu, "Jingle RTP Sessions," XSF XEP 0167, June 2009.

[MUC]	Saint-Andre, P., "Multi-User Chat," XSF XEP 0045, July 2008.
[RFC5056]	Williams, N., "On the Use of Channel Bindings to Secure Channels," RFC 5056, November 2007 (TXT).
[RFC5081]	Mavrogiannopoulos, N., "Using OpenPGP Keys for Transport Layer Security (TLS) Authentication," RFC 5081, November 2007 (TXT).
[TLS-SRP]	Taylor, D., Wu, T., Mavrogiannopoulos, N., and T. Perrin, "Using the Secure Remote Password (SRP) Protocol for TLS Authentication," RFC 5054, November 2007 (TXT).
[SCRAM]	Menon-Sen, A., Melnikov, A., Newman, C., and N. Williams, "Salted Challenge Response (SCRAM) SASL Mechanism," draft-newman-auth-scram-13 (work in progress), May 2009 (TXT).
[X509]	Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile," RFC 5280, May 2008 (TXT).
[XEP-0030]	Hildebrand, J., Millard, P., Eatmon, R., and P. Saint-Andre, "Service Discovery," XSF XEP 0030, June 2008.
[XEP-0053]	Saint-Andre, P., "XMPP Registrar Function," XSF XEP 0053, October 2008.
[XEP-0065]	Smith, D., Miller, M., and P. Saint-Andre, "SOCKS5 Bytestreams," XSF XEP 0065, May 2007.
[XEP-0115]	Hildebrand, J., Saint-Andre, P., Tronçon, R., and J. Konieczny, "Entity Capabilities," XSF XEP 0115, February 2008.
[XEP-0189]	Paterson, I., Saint-Andre, P., and D. Meyer, "Public Key Publishing," XSF XEP 0189, March 2009.
[XEP-0260]	Saint-Andre, P. and D. Meyer, "Jingle SOCKS5 Bytestreams Transport Method," XSF XEP 0260, February 2009.
[XEP-0261]	Saint-Andre, P., "Jingle In-Band Bytestreams Transport," XSF XEP 0261, February 2009.

Appendix A. XML Schema

TOC

The XML schema will be provided in a later version of this document.

Appendix B. Copying Conditions

TOC

and distribute it in any way that does not diminish the rights of anyone else to use, modify, and distribute it, provided that redistributed derivative works do not contain misleading author or version information. Derivative works need not be licensed under similar terms.

Authors' Addresses

TOC

	Dirk Meyer
	Universitaet Bremen TZI
Email:	dmeyer@tzi.de
	Peter Saint-Andre
	Cisco
Email:	psaintan@cisco.com