Network Working Group
Internet-Draft
Expires: April 25, 2004

P. Saint-Andre
J. Miller
Jabber Software Foundation

October 26, 2003

XMPP Core
draft-ietf-xmpp-core-19

Status of this Memo

Copyright Notice

Abstract

This memo defines the core features of the Extensible Messaging and Presence Protocol (XMPP), a protocol for streaming XML [1] elements in order to exchange messages and presence information in close to real time. While XMPP provides a generalized, extensible framework for transporting structured information, it is used mainly for the purpose of building instant messaging and presence applications that meet the requirements of RFC 2779.

Table of Contents

1. Introduction

1.1 Overview

The Extensible Messaging and Presence Protocol (XMPP) is an open XML [1] protocol for near-real-time messaging, presence, and request-response services. The basic syntax and semantics were developed originally within the Jabber open-source community, mainly in 1999. In 2002, the XMPP WG was chartered with developing an adaptation of the Jabber protocol that would be suitable as an IETF instant messaging (IM) and presence technology. As a result of work by the XMPP WG, the current memo defines the core features of XMPP; XMPP IM [21] defines the extensions required to provide the instant messaging and presence functionality defined in RFC 2779 [2].

1.2 Terminology

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 RFC 2119 [3].

1.3 Discussion Venue

The authors welcome discussion and comments related to the topics presented in this document. The preferred forum is the <xmppwg@jabber.org> mailing list, for which archives and subscription information are available at <http://www.jabber.org/cgi-bin/mailman/listinfo/xmppwg/>.

1.4 Intellectual Property Notice

This document is in full compliance with all provisions of Section 10 of RFC 2026. Parts of this specification use the term "jabber" for identifying namespaces and other protocol syntax. Jabber[tm] is a registered trademark of Jabber, Inc. Jabber, Inc. grants permission to the IETF for use of the Jabber trademark in association with this specification and its successors, if any.

2. Generalized Architecture

2.1 Overview

Although XMPP is not wedded to any specific network architecture, to date it usually has been implemented via a typical client-server architecture, wherein a client utilizing XMPP accesses a server over a TCP [4] socket.

The following diagram provides a high-level overview of this architecture (where "-" represents communications that use XMPP and "=" represents communications that use any other protocol).

```
C1 -  S1 - S2 - C3
     /  \
C2 -     G1 = FN1 = FC1
```

The symbols are as follows:

- C1, C2, C3 -- XMPP clients
- S1, S2 -- XMPP servers
- G1 -- A gateway that translates between XMPP and the protocol(s) used on a foreign (non-XMPP) messaging network
- FN1 -- A foreign messaging network
- FC1 -- A client on a foreign messaging network

2.2 Server

A server acts as an intelligent abstraction layer for XMPP communications. Its primary responsibilities are to manage connections from or sessions for other entities (in the form of XML streams (Section 4) to and from authorized clients, servers, and other entities) and to route appropriately-addressed XML stanzas (Section 9) among such entities over XML streams. Most XMPP-compliant servers also assume responsibility for the storage of data that is used by clients (e.g., contact lists for users of XMPP-based instant messaging and presence applications); in this case, the XML data is processed directly by the server itself on behalf of the client and is not routed to another entity. Compliant server implementations MUST ensure in-order processing of XML stanzas between any two entities.

2.3 Client

Most clients connect directly to a server over a TCP socket and use XMPP to take full advantage of the functionality provided by a server and any associated services. Although there is no necessary coupling of an XML stream to a TCP socket (e.g., a client COULD connect via HTTP [22] polling or some other mechanism), this specification defines a binding of XMPP to TCP only. Multiple resources (e.g., devices or locations) MAY connect simultaneously to a server on behalf of each authorized client, with each resource differentiated

by the resource identifier of a JID (e.g., <node@domain/home> vs. <node@domain/work>) as defined under Addressing Scheme (Section 3). The RECOMMENDED port for connections between a client and a server is 5222, as registered with the Internet Assigned Numbers Authority (IANA) [5] (see Port Numbers (Section 11.9)).

2.4 Gateway

A gateway is a special-purpose server-side service whose primary function is to translate XMPP into the protocol used by a foreign (non-XMPP) messaging system, as well as to translate the return data back into XMPP. Examples are gateways to Internet Relay Chat (IRC), Short Message Service (SMS), SIMPLE, SMTP, and legacy instant messaging networks such as AIM, ICQ, MSN Messenger, and Yahoo! Instant Messenger. Communications between gateways and servers, and between gateways and the foreign messaging system, are not defined in this document.

2.5 Network

Because each server is identified by a network address and because server-to-server communications are a straightforward extension of the client-to-server protocol, in practice the system consists of a network of servers that inter-communicate. Thus user-a@domain1 is able to exchange messages, presence, and other information with user-b@domain2. This pattern is familiar from messaging protocols (such as SMTP) that make use of network addressing standards. Communications between any two servers are OPTIONAL. If enabled, such communications SHOULD occur over XML streams that are bound to TCP sockets. The RECOMMENDED port for connections between servers is 5222, as registered with the Internet Assigned Numbers Authority (IANA) [5] (see Port Numbers (Section 11.9)).

3. Addressing Scheme

3.1 Overview

An entity is anything that can be considered a network endpoint (i.e., an ID on the network) and that can communicate using XMPP. All such entities are uniquely addressable in a form that is consistent with RFC 2396 [23]. For historical reasons, the address of such an entity is called a Jabber Identifier or JID. A valid JID contains a set of ordered elements formed of a domain identifier, node identifier, and resource identifier in the following format: [node@]domain[/resource]. Each allowable portion of a JID (node identifier, domain identifier, and resource identifier) may be up to 1023 bytes in length, resulting in a maximum total size (including the '@' and '/' separators) of 3071 bytes.

All JIDs are based on the foregoing structure. The most common use of this structure is to identify an instant messaging user, the server to which the user connects, and the user's active session or connection (e.g., a specific client) in the form of <user@host/resource>. However, node types other than clients are possible; for example, a specific chat room offered by a multi-user chat service could be addressed as <room@service> (where "room" is the name of the chat room and "service" is the hostname of the multi-user chat service) and a specific occupant of such a room could be addressed as <room@service/nick> (where "nick" is the occupant's room nickname). Many other JID types are possible (e.g., <domain/resource> could be a server-side script or service).

3.2 Domain Identifier

The domain identifier is the primary identifier and is the only REQUIRED element of a JID (a mere domain identifier is a valid JID). It usually represents the network gateway or "primary" server to which other entities connect for XML routing and data management capabilities. However, the entity referenced by a domain identifier is not always a server, and may be a service that is addressed as a subdomain of a server and that provides functionality above and beyond the capabilities of a server (e.g., a multi-user chat service, a user directory, or a gateway to a foreign messaging system).

The domain identifier for every server or service that will communicate over a network SHOULD resolve to a Fully Qualified Domain Name. A domain identifier MUST be not more than 1023 bytes in length and MUST conform to the Nameprep [6] profile of stringprep [7].

3.3 Node Identifier

The node identifier is an optional secondary identifier placed before the domain identifier and separated from the latter by the '@' character. It usually represents the entity requesting and using network access provided by the server or gateway (i.e., a client), although it can also represent other kinds of entities (e.g., a chat room associated with a multi-user chat service). The entity represented by a node identifier is addressed within the context of a specific domain; within instant messaging and presence applications of XMPP this address is called a "bare JID" and is of the form <node@domain>.

A node identifier MUST be no more than 1023 bytes in length and MUST conform to the Nodeprep (Appendix A) profile of stringprep [7].

3.4 Resource Identifier

The resource identifier is an optional tertiary identifier placed after the domain identifier and separated from the latter by the '/' character. A resource identifier may modify either a <node@domain> or mere <domain> address. It usually represents a specific session, connection (e.g., a device or location), or object (e.g., a participant in a multi-user chat room) belonging to the entity associated with a node identifier. A resource identifier is opaque to both servers and other clients, and is typically defined by a client implementation when it provides the information necessary to complete Resource Binding (Section 7) (although it may be generated by a server on behalf of a client). An entity may maintain multiple resources simultaneously.

A resource identifier MUST be no more than 1023 bytes in length and MUST conform to the Resourceprep (Appendix B) profile of stringprep [7].

3.5 Formal Syntax

The syntax for a JID is defined below using Augmented Backus-Naur Form as defined in RFC 2234 [8]. The IPv4address and IPv6address rules are defined in Appendix B of RFC 2373 [9]; the hostname rule is defined in Section 3.2.2 of RFC 2396 [23]; the allowable character sequences that conform to the node rule are defined by the Nodeprep (Appendix A) profile of stringprep [7] as documented in this memo; the allowable character sequences that conform to the resource rule are defined by the Resourceprep (Appendix B) profile of stringprep [7] as documented in this memo.

```
jid           = [ node "@" ] domain [ "/" resource ]
domain        = hostname / IPv4address / IPv6address
```

3.6 Determination of Addresses

After Stream Authentication (Section 6) and, if appropriate, Resource Binding (Section 7), the receiving entity for a stream MUST determine the initiating entity's JID.

For server-to-server communications, the initiating entity's JID SHOULD be the authorization identity, derived from the authentication identity as defined in RFC 2222 [13] if no authorization identity was specified during stream authentication.

For client-to-server communications, the "bare JID" (<node@domain>) SHOULD be the authorization identity, derived from the authentication identity as defined in RFC 2222 [13] if no authorization identity was specified during stream authentication; the resource identifier

portion of the "full JID" (<node@domain/resource>) SHOULD be the resource identifier negotiated by the client and server during Resource Binding (Section 7).

The receiving entity MUST ensure that the resulting JID (including node identifier, domain identifier, resource identifier, and separator characters) conforms to the rules and formats defined earlier in this section.

4. XML Streams

4.1 Overview

Two fundamental concepts make possible the rapid, asynchronous exchange of relatively small payloads of structured information between presence-aware entities: XML streams and XML stanzas. These terms are defined as follows:

Definition of XML Stream: An XML stream is a container for the exchange of XML elements between any two entities over a network. An XML stream is negotiated from an initiating entity (usually a client or server) to a receiving entity (usually a server), normally over a TCP socket, and corresponds to the initiating entity's "session" with the receiving entity. The start of the XML stream is denoted unambiguously by an opening XML <stream> tag (with appropriate attributes and namespace declarations), while the end of the XML stream is denoted unambiguously by a closing XML </stream> tag. An XML stream is unidirectional; in order to enable bidirectional information exchange, the initiating entity and receiving entity MUST negotiate one stream in each direction (the "initial stream" and the "response stream"), normally over the same TCP connection.

Definition of XML Stanza: An XML stanza is a discrete semantic unit of structured information that is sent from one entity to another over an XML stream. An XML stanza exists at the direct child level of the root <stream/> element and is said to be well-balanced if it matches production [43] content of the XML specification [1]). The start of any XML stanza is denoted unambiguously by the element start tag at depth=1 of the XML stream (e.g., <presence>), and the end of any XML stanza is denoted unambiguously by the corresponding close tag at depth=1 (e.g., </presence>). An XML stanza MAY contain child elements (with accompanying attributes, elements, and CDATA) as necessary in order to convey the desired information. The only defined XML stanzas are <message/>, <presence/>, and <iq/> as defined under XML Stanzas (Section 9); an XML element sent for the purpose of stream encryption (Section 5), stream authentication (Section 6), or server dialback (Section

8) is not considered to be an XML stanza.

Consider the example of a client's session with a server. In order to connect to a server, a client MUST initiate an XML stream by sending an opening <stream> tag to the server, optionally preceded by a text declaration specifying the XML version and the character encoding supported (see Inclusion of Text Declaration (Section 10.4); see also Character Encoding (Section 10.5)). Subject to local policies and service provisioning, the server SHOULD then reply with a second XML stream back to the client, again optionally preceded by a text declaration. Once the client has completed Stream Authentication (Section 6), the client MAY send an unbounded number of XML stanzas over the stream to any recipient on the network. When the client desires to close the stream, it simply sends a closing </stream> tag to the server (alternatively, the stream may be closed by the server), after which both the client and server SHOULD close the underlying TCP connection as well.

Those who are accustomed to thinking of XML in a document-centric manner may wish to view a client's session with a server as consisting of two open-ended XML documents: one from the client to the server and one from the server to the client. From this perspective, the root <stream/> element can be considered the document entity for each "document", and the two "documents" are built up through the accumulation of XML stanzas sent over the two XML streams. However, this perspective is a convenience only, and XMPP does not deal in documents but in XML streams and XML stanzas.

In essence, then, an XML stream acts as an envelope for all the XML stanzas sent during a session. We can represent this in a simplistic fashion as follows:

```
|--------------------|
| <stream>           |
|--------------------|
| <presence>         |
|   <show/>          |
| </presence>        |
|--------------------|
| <message to='foo'> |
|   <body/>          |
| </message>         |
|--------------------|
| <iq to='bar'>      |
|   <query/>         |
| </iq>              |
|--------------------|
| ...                |
```

```
|--------------------|
| </stream>          |
|--------------------|
```

4.2 Stream Attributes

The attributes of the stream element are as follows:

- to -- The 'to' attribute SHOULD be used only in the XML stream header from the initiating entity to the receiving entity, and MUST be set to the JID of the receiving entity. There SHOULD be no 'to' attribute set in the XML stream header by which the receiving entity replies to the initiating entity; however, if a 'to' attribute is included, it SHOULD be silently ignored by the initiating entity.

- from -- The 'from' attribute SHOULD be used only in the XML stream header from the receiving entity to the initiating entity, and MUST be set to the JID of the receiving entity granting access to the initiating entity. There SHOULD be no 'from' attribute on the XML stream header sent from the initiating entity to the receiving entity; however, if a 'from' attribute is included, it SHOULD be silently ignored by the receiving entity.

- id -- The 'id' attribute SHOULD be used only in the XML stream header from the receiving entity to the initiating entity. This attribute is a unique identifier created by the receiving entity to function as a session key for the initiating entity's streams with the receiving entity, and MUST be unique within the receiving application (normally a server). There SHOULD be no 'id' attribute on the XML stream header sent from the initiating entity to the receiving entity; however, if an 'id' attribute is included, it SHOULD be silently ignored by the receiving entity.

- xml:lang -- An 'xml:lang' attribute (as defined in Section 2.12 of the XML specification [1]) SHOULD be included by the initiating entity on the header for the initial stream to specify the default language of any human-readable XML character data it sends over that stream. If included, the receiving entity SHOULD remember that value as the default for both the initial stream and the response stream; if not included, the receiving entity SHOULD use a configurable default value for both streams, which it MUST communicate in the header for the response stream. For all stanzas sent over the initial stream, if the initiating entity does not include an 'xml:lang' attribute, the receiving entity SHOULD apply the default value; if the initiating entity does include an 'xml:lang' attribute, the receiving entity MUST NOT modify or

delete it (see also xml:lang (Section 9.1.5)). The value of the 'xml:lang' attribute MUST be an NMTOKEN and MUST conform to the format defined in RFC 3066 [16].

o version -- The presence of the version attribute set to a value of "1.0" signals support for the stream-related protocols (including stream features) defined in this specification. Detailed rules regarding generation and handling of this attribute are defined below.

We can summarize as follows:

	initiating to receiving	receiving to initiating
to	hostname of receiver	silently ignored
from	silently ignored	hostname of receiver
id	silently ignored	session key
xml:lang	default language	default language
version	signals XMPP 1.0 support	signals XMPP 1.0 support

4.2.1 Version Support

The following rules apply to the generation and handling of the 'version' attribute:

1. If the initiating entity complies with the XML streams protocol defined herein (including Stream Encryption (Section 5), Stream Authentication (Section 6), and Stream Errors (Section 4.6)), it MUST include the 'version' attribute in the XML stream header it sends to the receiving entity, and it MUST set the value of the 'version' attribute to "1.0".

2. If the initiating entity includes the 'version' attribute set to a value of "1.0" in its stream header and the receiving entity supports XMPP 1.0, the receiving entity MUST reciprocate by including the 'version' attribute set to a value of "1.0" in its stream header response.

3. If the initiating entity does not include the 'version' attribute in its stream header, the receiving entity still SHOULD include the 'version' attribute set to a value of "1.0" in its stream header response.

4. If the initiating entity includes the 'version' attribute set to a value other than "1.0", the receiving entity SHOULD include the 'version' attribute set to a value of "1.0" in its stream header response, but MAY at its discretion generate an

<unsupported-version/> stream error and terminate the XML stream and underlying TCP connection.

5. If the receiving entity includes the 'version' attribute set to a value other than "1.0" in its stream header response, the initiating entity SHOULD generate an <unsupported-version/> stream error and terminate the XML stream and underlying TCP connection.

4.3 Namespace Declarations

The stream element MUST possess both a streams namespace declaration and a default namespace declaration (as "namespace declaration" is defined in the XML namespaces specification [10]). For detailed information regarding the streams namespace and default namespace, see Namespace Names and Prefixes (Section 10.2).

4.4 Stream Features

If the initiating entity includes the 'version' attribute set to a value of "1.0" in the initial stream header, the receiving entity MUST send a <features/> child element (prefixed by the streams namespace prefix) to the initiating entity in order to announce any stream-level features that can be negotiated (or capabilities that otherwise need to be advertised). Currently this is used only for Stream Encryption (Section 5), Stream Authentication (Section 6), and Resource Binding (Section 7) as defined herein, and for Session Establishment as defined in XMPP IM [21]; however, the stream features functionality could be used to advertise other negotiable features in the future. If an entity does not understand or support some features, it SHOULD silently ignore them.

4.5 Stream Encryption and Authentication

XML streams in XMPP 1.0 SHOULD be encrypted as defined under Stream Encryption (Section 5) and MUST be authenticated as defined under Stream Authentication (Section 6). If the initiating entity attempts to send an XML Stanza (Section 9) before the stream has been authenticated, the receiving entity SHOULD return a <not-authorized/> stream error to the initiating entity and then terminate both the XML stream and the underlying TCP connection.

4.6 Stream Errors

The root stream element MAY contain an <error/> child element that is prefixed by the streams namespace prefix. The error child MUST be sent by a compliant entity (usually a server rather than a client) if

it perceives that a stream-level error has occurred.

4.6.1 Rules

The following rules apply to stream-level errors:

- It is assumed that all stream-level errors are unrecoverable; therefore, if an error occurs at the level of the stream, the entity that detects the error MUST send a stream error to the other entity, send a closing </stream> tag, and terminate the underlying TCP connection.

- If the error occurs while the stream is being set up, the receiving entity MUST still send the opening <stream> tag, include the <error/> element as a child of the stream element, send the closing </stream> tag, and terminate the underlying TCP connection. In this case, if the initiating entity provides an unknown host in the 'to' attribute (or provides no 'to' attribute at all), the server SHOULD provide the server's authoritative hostname in the 'from' attribute of the stream header sent before termination.

4.6.2 Syntax

The syntax for stream errors is as follows:

```
<stream:error>
  <defined-condition xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
  <text xmlns='urn:ietf:params:xml:ns:xmpp-streams'>
    OPTIONAL descriptive text
  </text>
  [OPTIONAL application-specific condition element]
</stream:error>
```

The <error/> element:

- MUST contain a child element corresponding to one of the defined stanza error conditions defined below; this element MUST be qualified by the 'urn:ietf:params:xml:ns:xmpp-streamstreams' namespace (this namespace name adheres to the format defined in The IETF XML Registry [24])

- MAY contain a <text/> child containing CDATA that describes the error in more detail; this element MUST be qualified by the 'urn:ietf:params:xml:ns:xmpp-streams' namespace and SHOULD possess an 'xml:lang' attribute

- MAY contain a child element for an application-specific error condition; this element MUST be qualified by an application-defined namespace, and its structure is defined by that namespace

The <text/> element is OPTIONAL. If included, it SHOULD be used only to provide descriptive or diagnostic information that supplements the meaning of a defined condition or application-specific condition. It SHOULD NOT be interpreted programmatically by an application. It SHOULD NOT be used as the error message presented to a user, but MAY be shown in addition to the error message associated with the included condition element (or elements).

4.6.3 Defined Conditions

The following stream-level error conditions are defined:

- <bad-format/> -- the entity has sent XML that cannot be processed; this error MAY be used rather than more specific XML-related errors such as <bad-namespace-prefix/>, <invalid-xml/>, <restricted-xml/>, <unsupported-encoding/>, and <xml-not-well-formed/>, although the more specific errors are preferred.

- <bad-namespace-prefix/> -- the entity has sent a namespace prefix that is unsupported, or has sent no namespace prefix on an element that requires such a prefix (see XML Namespace Names and Prefixes (Section 10.2)).

- <conflict/> -- the server is closing the active stream for this entity because a new stream has been initiated that conflicts with the existing stream.

- <connection-timeout/> -- the entity has not generated any traffic over the stream for some period of time (configurable according to a local service policy).

- <host-gone/> -- the value of the 'to' attribute provided by the initiating entity in the stream header corresponds to a hostname that is no longer hosted by the server.

- <host-unknown/> -- the value of the 'to' attribute provided by the initiating entity in the stream header does not correspond to a hostname that is hosted by the server.

- <improper-addressing/> -- a stanza sent between two servers lacks a 'to' or 'from' attribute (or the attribute has no value).

- <internal-server-error/> -- the server has experienced a misconfiguration or an otherwise-undefined internal error that prevents it from servicing the stream.

- <invalid-from/> -- the JID or hostname provided in a 'from' address does not match an authorized JID or validated domain negotiated between servers via SASL or dialback, or between a client and a server via authentication and resource authorization.

- <invalid-id/> -- the stream ID or dialback ID is invalid or does not match an ID previously provided.

- <invalid-namespace/> -- the streams namespace name is something other than "http://etherx.jabber.org/streams" or the dialback namespace name is something other than "jabber:server:dialback" (see XML Namespace Names and Prefixes (Section 10.2)).

- <invalid-xml/> -- the entity has sent invalid XML over the stream to a server that performs validation (see Validation (Section 10.3)).

- <not-authorized/> -- the entity has attempted to send data before the stream has been authenticated, or otherwise is not authorized to perform an action related to stream negotiation; the receiving entity MUST NOT process the offending stanza before sending the stream error.

- <policy-violation/> -- the entity has violated some local service policy; the server MAY choose to specify the policy in the <text/> element.

- <remote-connection-failed/> -- the server is unable to properly connect to a remote resource that is required for authentication or authorization.

- <resource-constraint/> -- the server lacks the system resources necessary to service the stream.

- <restricted-xml/> -- the entity has attempted to send restricted XML features such as a comment, processing instruction, DTD, entity reference, or unescaped character (see Restrictions (Section 10.1)).

- <see-other-host/> -- the server will not provide service to the initiating entity but is redirecting traffic to another host; the server SHOULD specify the alternate hostname or IP address in the CDATA of the <see-other-host/> element.

- <system-shutdown/> -- the server is being shut down and all active streams are being closed.

- <undefined-condition/> -- the error condition is not one of those defined by the other conditions in this list; this error condition SHOULD be used only in conjunction with an application-specific condition.

- <unsupported-encoding/> -- the initiating entity has encoded the stream in an encoding that is not supported by the server (see Character Encoding (Section 10.5)).

- <unsupported-stanza-type/> -- the initiating entity has sent a first-level child of the stream that is not supported by the server.

- <unsupported-version/> -- the value of the 'version' attribute provided by the initiating entity in the stream header specifies a version of XMPP that is not supported by the server; the server MAY specify the version(s) it supports in the <text/> element.

- <xml-not-well-formed/> -- the initiating entity has sent XML that is not well-formed as defined by the XML specification [1].

4.6.4 Application-Specific Conditions

As noted, an application MAY provide application-specific stream error information by including a properly-namespaced child in the error element. The application-specific element SHOULD supplement or further qualify a defined element. Thus the <error/> element will contain two or three child elements:

```
<stream:error>
  <xml-not-well-formed
      xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
    <text xml:lang='en' xmlns='urn:ietf:params:xml:ns:xmpp-streams'>
      Some special application diagnostic information!
    </text>
  <escape-your-data xmlns='application-ns'/>
</stream:error>
</stream:stream>
```

4.7 Simplified Stream Examples

This section contains two simplified examples of a stream-based "session" of a client on a server (where the "C" lines are sent from

the client to the server, and the "S" lines are sent from the server to the client); these examples are included for the purpose of illustrating the concepts introduced thus far.

A basic "session":

```
C: <?xml version='1.0'?>
   <stream:stream
       to='example.com'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       version='1.0'>
S: <?xml version='1.0'?>
   <stream:stream
       from='example.com'
       id='someid'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       version='1.0'>
... encryption, authentication, and resource binding ...
C:   <message from='juliet@example.com'
              to='romeo@example.net'
              xml:lang='en'>
C:     <body>Art thou not Romeo, and a Montague?</body>
C:   </message>
S:   <message from='romeo@example.net'
              to='juliet@example.com'
              xml:lang='en'>
S:     <body>Neither, fair saint, if either thee dislike.</body>
S:   </message>
C: </stream:stream>
S: </stream:stream>
```

A "session" gone bad:

```
C: <?xml version='1.0'?>
   <stream:stream
       to='example.com'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       version='1.0'>
S: <?xml version='1.0'?>
   <stream:stream
       from='example.com'
       id='someid'
       xmlns='jabber:client'
       xmlns:stream='http://etherx.jabber.org/streams'
       version='1.0'>
```

```
... encryption, authentication, and resource binding ...
C: <message xml:lang='en'>
     <body>Bad XML, no closing body tag!
   </message>
S: <stream:error>
    <xml-not-well-formed
        xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
   </stream:error>
S: </stream:stream>
```

5. Stream Encryption

5.1 Overview

XMPP includes a method for securing the stream from tampering and eavesdropping. This channel encryption method makes use of the Transport Layer Security (TLS) [11] protocol, along with a "STARTTLS" extension that is modelled after similar extensions for the IMAP [25], POP3 [26], and ACAP [27] protocols as described in RFC 2595 [28]. The namespace name for the STARTTLS extension is 'urn:ietf:params:xml:ns:xmpp-tls', which adheres to the format defined in The IETF XML Registry [24].

An administrator of a given domain MAY require the use of TLS for client-to-server communications, server-to-server communications, or both. Clients SHOULD use TLS to secure the streams prior to attempting to complete Stream Authentication (Section 6), and servers SHOULD use TLS between two domains for the purpose of securing server-to-server communications.

The following rules apply:

1. An initiating entity that complies with this specification MUST include the 'version' attribute set to a value of "1.0" in the initial stream header.

2. If the TLS negotiation occurs between two servers, communications MUST NOT proceed until the Domain Name System (DNS) hostnames asserted by the servers have been resolved (see Server-to-Server Communications (Section 13.3)).

3. When a receiving entity that complies with this specification receives an initial stream header that includes the 'version' attribute set to a value of "1.0", after sending a stream header in reply (including the version flag) it MUST include a <starttls/> element (qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace) along with the list

of other stream features it supports.

4. If the initiating entity chooses to use TLS for stream encryption, TLS negotiation MUST be completed before proceeding to SASL negotiation; this order of negotiation is required in order to help safeguard authentication information sent during SASL negotiation, as well as to make it possible to base the use of the SASL EXTERNAL mechanism on a certificate provided during prior TLS negotiation.

5. During TLS negotiation, an entity MUST NOT send any white space characters (matching production [3] content of the XML specification [1]) within the root stream element as separators between elements (any white space characters shown in the TLS examples below are included for the sake of readability only); this prohibition helps to ensure proper security layer byte precision.

6. The receiving entity MUST consider the TLS negotiation to have begun immediately after sending the closing ">" character of the <proceed/> element. The initiating entity MUST consider the TLS negotiation to have begun immediately after receiving the closing ">" character of the <proceed/> element from the receiving entity.

7. The initiating entity MUST validate the certificate presented by the receiving entity; there are two cases:

   Case 1 -- The initiating entity has been configured with a set of trusted root certificates: Normal certificate validation processing is appropriate, and SHOULD follow the rules defined for HTTP over TLS [12]. The trusted roots may be either a well-known public set or a manually configured Root CA (e.g., an organization's own Certificate Authority or a self-signed Root CA for the service as defined under High Security (Section 13.1)). This case is RECOMMENDED.

   Case 2 -- The initiating entity has been configured with the receiving entity's self-signed service certificate: Simple comparison of public keys is appropriate. This case is NOT RECOMMENDED (see High Security (Section 13.1) for details).

   If the above methods fail, the certificate SHOULD be presented to a human (e.g., an end user or server administrator) for approval; if presented, the receiver MUST deliver the entire certificate chain to the human, who SHOULD be given the option to store the Root CA certificate (not the service or End Entity certificate) and to not be queried again regarding acceptance of

the certificate for some reasonable period of time.

8. If the TLS negotiation is successful, the receiving entity MUST discard any knowledge obtained from the initiating entity before TLS takes effect.

9. If the TLS negotiation is successful, the initiating entity MUST discard any knowledge obtained from the receiving entity before TLS takes effect.

10. If the TLS negotiation is successful, the receiving entity MUST NOT offer the STARTTLS extension to the initiating entity along with the other stream features that are offered when the stream is restarted.

11. If the TLS negotiation is successful, the initiating entity MUST continue with SASL negotiation.

12. If the TLS negotiation results in failure, the receiving entity MUST terminate both the XML stream and the underlying TCP connection.

13. See Mandatory-to-Implement Technologies (Section 13.5) regarding mechanisms that MUST be supported.

5.2 Narrative

When an initiating entity secures a stream with a receiving entity, the steps involved are as follows:

1. The initiating entity opens a TCP connection and initiates the stream by sending the opening XML stream header to the receiving entity, including the 'version' attribute set to a value of "1.0".

2. The receiving entity responds by opening a TCP connection and sending an XML stream header to the initiating entity, including the 'version' attribute set to a value of "1.0".

3. The receiving entity offers the STARTTLS extension to the initiating entity by including it with the list of other supported stream features (if TLS is required for interaction with the receiving entity, it SHOULD signal that fact by including a <required/> element as a child of the <starttls/> element).

4. The initiating entity issues the STARTTLS command (i.e., a

<starttls/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace) to instruct the receiving entity that it wishes to begin a TLS negotiation to secure the stream.

5. The receiving entity MUST reply with either a <proceed/> element or a <failure/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-tls' namespace. If the failure case occurs, the receiving entity MUST terminate both the XML stream and the underlying TCP connection. If the proceed case occurs, the entities MUST attempt to complete the TLS negotiation over the TCP connection and MUST NOT send any further XML data until the TLS negotiation is complete.

6. The initiating entity and receiving entity attempt to complete a TLS negotiation in accordance with RFC 2246 [11].

7. If the TLS negotiation is unsuccessful, the receiving entity MUST terminate the TCP connection (it is not necessary to send a closing </stream> tag first, since the receiving entity and initiating entity MUST consider the original stream to be closed upon sending or receiving the <success/> element). If the TLS negotiation is successful, the initiating entity MUST initiate a new stream by sending an opening XML stream header to the receiving entity.

8. Upon receiving the new stream header from the initiating entity, the receiving entity MUST respond by sending a new XML stream header to the initiating entity along with the available features (but NOT including the STARTTLS feature).

5.3 Client-to-Server Example

The following example shows the data flow for a client securing a stream using STARTTLS (note: the alternate steps shown below are provided to illustrate the protocol for failure cases; they are not exhaustive and would not necessarily be triggered by the data sent in the example).

Step 1: Client initiates stream to server:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Step 2: Server responds by sending a stream tag to client:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_123'
    from='example.com'
    version='1.0'>
```

Step 3: Server sends the STARTTLS extension to client along with authentication mechanisms and any other stream features:

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'>
    <required/>
  </starttls>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
</stream:features>
```

Step 4: Client sends the STARTTLS command to server:

```
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Step 5: Server informs client to proceed:

```
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Step 5 (alt): Server informs client that TLS negotiation has failed and closes both stream and TCP connection:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
</stream:stream>
```

Step 6: Client and server attempt to complete TLS negotiation over the existing TCP connection.

Step 7: If TLS negotiation is successful, client initiates a new stream to server:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Step 7 (alt): If TLS negotiation is unsuccessful, Server2 closes TCP connection.

Step 8: Server responds by sending a stream header to client along with any available stream features:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='example.com'
    id='c2s_234'
    version='1.0'>
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
    <mechanism>EXTERNAL</mechanism>
  </mechanisms>
</stream:features>
```

Step 9: Client continues with Stream Authentication (Section 6).

5.4 Server-to-Server Example

The following example shows the data flow for two servers securing a stream using STARTTLS (note: the alternate steps shown below are provided to illustrate the protocol for failure cases; they are not exhaustive and would not necessarily be triggered by the data sent in the example).

Step 1: Server1 initiates stream to Server2:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Step 2: Server2 responds by sending a stream tag to Server1:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='example.com'
    id='s2s_123'
    version='1.0'>
```

Step 3: Server2 sends the STARTTLS extension to Server1 along with authentication mechanisms and any other stream features:

```
<stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
    <required/>
  </starttls>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>KERBEROS_V4</mechanism>
  </mechanisms>
</stream:features>
```

Step 4: Server1 sends the STARTTLS command to Server2:

```
<starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Step 5: Server2 informs Server1 to proceed:

```
<proceed xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
```

Step 5 (alt): Server2 informs Server1 that TLS negotiation has failed and closes stream:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
</stream:stream>
```

Step 6: Server1 and Server2 attempt to complete TLS negotiation via TCP.

Step 7: If TLS negotiation is successful, Server1 initiates a new stream to Server2:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Step 7 (alt): If TLS negotiation is unsuccessful, server closes TCP connection.

Step 8: Server2 responds by sending a stream header to Server1 along with any available stream features:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
```

```
    from='example.com'
    id='s2s_234'
    version='1.0'>
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>KERBEROS_V4</mechanism>
    <mechanism>EXTERNAL</mechanism>
  </mechanisms>
</stream:features>
```

Step 9: Server1 continues with Stream Authentication (Section 6).

6. Stream Authentication

6.1 Overview

XMPP includes a method for authenticating a stream by means of an XMPP-specific profile of the Simple Authentication and Security Layer (SASL) [13]. SASL provides a generalized method for adding authentication support to connection-based protocols, and XMPP uses a generic XML namespace profile for SASL that conforms to Section 4 ("Profiling Requirements") of RFC 2222 [13] (the namespace name that qualifies XML elements used in stream authentication is 'urn:ietf:params:xml:ns:xmpp-sasl', which adheres to the format defined in The IETF XML Registry [24]).

The following rules apply:

1. If the SASL negotiation occurs between two servers, communications MUST NOT proceed until the Domain Name System (DNS) hostnames asserted by the servers have been resolved (see Server-to-Server Communications (Section 13.3)).

2. If the initiating entity is capable of stream authentication via SASL, it MUST include the 'version' attribute set to a value of "1.0" in the initial stream header.

3. If the receiving entity is capable of stream authentication via SASL, it MUST send one or more authentication mechanisms within a <mechanisms/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace in reply to the opening stream tag received from the initiating entity (if the opening stream tag included the 'version' attribute set to a value of "1.0").

4. During SASL negotiation, an entity MUST NOT send any white space characters (matching production [3] content of the XML

specification [1]) within the root stream element as separators between elements (any white space characters shown in the SASL examples below are included for the sake of readability only); this prohibition helps to ensure proper security layer byte precision.

5. Any character data contained within the XML elements used during SASL negotiation MUST be encoded using base64 [14].

6. If supported by the selected SASL mechanism, the initiating entity SHOULD provide a username during SASL negotiation. The username-value SHOULD be the initiating entity's sending domain in the case of server-to-server communications, and SHOULD be the initiating entity's registered username in the case of client-to-server communications.

7. If supported by the selected SASL mechanism, the initiating entity MAY provide an authorization identity during SASL negotiation, which SHOULD be a non-default identity for which the entity is seeking authorization to impersonate (i.e., not the default authorization identity, which is derived from the authentication identity as described in RFC 2222 [13]). If provided, the authzid-value MUST be of the form <domain> (i.e., a domain identifier only) for servers and of the form <node@domain> (i.e., node identifier and domain identifier) for clients.

8. Upon successful SASL negotiation that involves negotiation of a security layer, the receiving entity MUST discard any knowledge obtained from the initiating entity which was not obtained from the SASL negotiation itself.

9. Upon successful SASL negotiation that involves negotiation of a security layer, the initiating entity MUST discard any knowledge obtained from the receiving entity which was not obtained from the SASL negotiation itself.

10. See Mandatory-to-Implement Technologies (Section 13.5) regarding mechanisms that MUST be supported.

6.2 Narrative

When an initiating entity authenticates with a receiving entity, the steps involved are as follows:

1. The initiating entity requests SASL authentication by including the 'version' attribute in the opening XML stream header sent to

the receiving entity, with the value set to "1.0".

2. After sending an XML stream header in reply, the receiving entity sends a list of available SASL authentication mechanisms; each of these is a <mechanism/> element included as a child within a <mechanisms/> container element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace, which in turn is a child of a <features/> element in the streams namespace. If Stream Encryption (Section 5) needs to be established before a particular authentication mechanism may be used, the receiving entity MUST NOT provide that mechanism in the list of available SASL authentication mechanisms prior to stream encryption. If the initiating entity presents a valid certificate during prior TLS negotiation, the receiving entity SHOULD offer the SASL EXTERNAL mechanism to the initiating entity during stream authentication (refer to RFC 2222 [13]), although the EXTERNAL mechanism MAY be offered under other circumstances as well.

3. The initiating entity selects a mechanism by sending an <auth/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace to the receiving entity and including an appropriate value for the 'mechanism' attribute; this element MAY optionally contain character data (in SASL terminology, the "initial response") if the mechanism supports or requires it. If the initiating entity selects the EXTERNAL mechanism for authentication and presented a certificate during prior TLS negotiation, the authentication credentials SHOULD be taken from that certificate.

4. If necessary, the receiving entity challenges the initiating entity by sending a <challenge/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace to the initiating entity; this element MAY optionally contain character data (which MUST be computed in accordance with the definition of the SASL mechanism chosen by the initiating entity).

5. The initiating entity responds to the challenge by sending a <response/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace to the receiving entity; this element MAY optionally contain character data (which MUST be computed in accordance with the definition of the SASL mechanism chosen by the initiating entity).

6. If necessary, the receiving entity sends more challenges and the initiating entity sends more responses.

This series of challenge/response pairs continues until one of three things happens:

1. The initiating entity aborts the handshake by sending an <abort/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace to the receiving entity. Upon receiving an <abort/> element, the receiving entity SHOULD allow a configurable but reasonable number of retries (at least 2), after which it MUST terminate the TCP connection; this allows the initiating entity (e.g., an end-user client) to tolerate incorrectly-provided credentials (e.g., a mistyped password) without being forced to reconnect.

2. The receiving entity reports failure of the handshake by sending a <failure/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace to the initiating entity (the particular cause of failure SHOULD be communicated in an appropriate child element of the <failure/> element as defined under SASL Errors (Section 6.4)). If the failure case occurs, the receiving entity SHOULD allow a configurable but reasonable number of retries (at least 2), after which it MUST terminate the TCP connection; this allows the initiating entity (e.g., an end-user client) to tolerate incorrectly-provided credentials (e.g., a mistyped password) without being forced to reconnect.

3. The receiving entity reports success of the handshake by sending a <success/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-sasl' namespace to the initiating entity; this element MAY optionally contain character data (in SASL terminology, "additional data with success") if required by the chosen SASL mechanism. Upon receiving the <success/> element, the initiating entity MUST initiate a new stream by sending an opening XML stream header to the receiving entity (it is not necessary to send a closing </stream> tag first, since the receiving entity and initiating entity MUST consider the original stream to be closed upon sending or receiving the <success/> element). Upon receiving the new stream header from the initiating entity, the receiving entity MUST respond by sending a new XML stream header to the initiating entity, along with any available features (but NOT including the STARTTLS feature) or an empty <features/> element (to signify that no additional features are available); any such additional features not defined herein MUST be defined by the relevant extension to XMPP.

6.3 SASL Definition

Section 4 of the SASL specification [13] requires that the following information be supplied by a protocol definition:

service name: "xmpp"

initiation sequence: After the initiating entity provides an opening XML stream header and the receiving entity replies in kind, the receiving entity provides a list of acceptable authentication methods. The initiating entity chooses one method from the list and sends it to the receiving entity as the value of the 'mechanism' attribute possessed by an <auth/> element, optionally including an initial response to avoid a round trip.

exchange sequence: Challenges and responses are carried through the exchange of <challenge/> elements from receiving entity to initiating entity and <response/> elements from initiating entity to receiving entity. The receiving entity reports failure by sending a <failure/> element and success by sending a <success/> element; the initiating entity aborts the exchange by sending an <abort/> element. Upon successful negotiation, both sides consider the original XML stream to be closed and new stream headers are sent by both entities.

security layer negotiation: The security layer takes effect immediately after sending the closing ">" character of the <success/> element for the receiving entity, and immediately after receiving the closing ">" character of the <success/> element for the initiating entity. The order of layers is first TCP, then TLS, then SASL, then XMPP.

use of the authorization identity: The authorization identity may be used by xmpp to denote the <node@domain> of a client or the sending <domain> of a server.

6.4 SASL Errors

The following SASL-related error conditions are defined:

- <aborted/> -- The receiving entity acknowledges an <abort/> element sent by the initiating entity; sent in reply to the <abort/> element.

- <incorrect-encoding/> -- The data provided by the initiating entity could not be processed because the base64 [14] encoding is incorrect; sent in reply to a <response/> element or an <auth/> element with initial challenge data.

- <invalid-authzid/> -- The authzid provided by the initiating entity is invalid, either because it is incorrectly formatted or because the initiating entity does not have permissions to

authorize that ID; sent in reply to a <response/> element or an <auth/> element with initial challenge data.

- <invalid-mechanism/> -- The initiating entity did not provide a mechanism or requested a mechanism that is not supported by the receiving entity; sent in reply to an <auth/> element.

- <mechanism-too-weak/> -- The mechanism requested by the initiating entity is weaker than server policy permits for that initiating entity; sent in reply to a <response/> element or an <auth/> element with initial challenge data.

- <not-authorized/> -- The authentication failed because the initiating entity did not provide valid credentials (this includes but is not limited to the case of an unknown username); sent in reply to a <response/> element or an <auth/> element with initial challenge data.

- <temporary-auth-failure/> -- The authentication failed because of a temporary error condition within the receiving entity; sent in reply to an <auth/> element or <response/> element.

6.5 Client-to-Server Example

The following example shows the data flow for a client authenticating with a server using SASL, normally after successful TLS negotiation (note: the alternate steps shown below are provided to illustrate the protocol for failure cases; they are not exhaustive and would not necessarily be triggered by the data sent in the example).

Step 1: Client initiates stream to server:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Step 2: Server responds with a stream tag sent to client:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_234'
    from='example.com'
    version='1.0'>
```

Step 3: Server informs client of available authentication mechanisms:

```
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>PLAIN</mechanism>
  </mechanisms>
</stream:features>
```

Step 4: Client selects an authentication mechanism:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='DIGEST-MD5'/>
```

Step 5: Server sends a base64 [14]-encoded challenge to client:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cmVhbG09InNvbWVyZWFsbSIsbm9uY2U9Ik9BNk1HOXRFUUdtMmhoIixxb3A9ImF1dGgi
LGNoYXJzZXQ9dXRmLTgsYWxnb3JpdGhtPW1kNS1zZXNzCg==
</challenge>
```

The decoded challenge is:

```
realm="somerealm",nonce="OA6MG9tEQGm2hh",\
qop="auth",charset=utf-8,algorithm=md5-sess
```

Step 5 (alt): Server returns error to client:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <incorrect-encoding/>
</failure>
</stream:stream>
```

Step 6: Client sends a base64 [14]-encoded response to the challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
dXNlcm5hbWU9InNvbWVub2RlIixyZWFsbT0ic29tZXJlYWxtIixub25jZT0i
T0E2TUc5dEVRR20yaGgiLGNub25jZT0iT0E2TUhYaDZWcVRyUmsiLG5jPTAw
MDAwMDAxLHFvcD1hdXRoLGRpZ2VzdC11cmk9InhtcHAvZXhhbXBsZS5jb20i
LHJlc3BvbnNlPWQzODhkYWQ5MGQ0YmJkNzYwYTE1MjMyMWYyMTQzYWY3LGNo
YXJzZXQ9dXRmLTgK
</response>
```

The decoded response is:

```
username="somenode",realm="somerealm",\
nonce="OA6MG9tEQGm2hh",cnonce="OA6MHXh6VqTrRk",\
nc=00000001,qop=auth,digest-uri="xmpp/example.com",\
```

response=d388dad90d4bbd760a152321f2143af7,charset=utf-8

Step 7: Server sends another base64 [14]-encoded challenge to client:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cnNwYXV0aD1lYTQwZjYwMzM1YzQyN2I1NTI3Yjg0ZGJhYmNkZmZmZAo=
</challenge>
```

The decoded challenge is:

rspauth=ea40f60335c427b5527b84dbabcdfffd

Step 7 (alt): Server returns error to client:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <mechanism-too-weak/>
</failure>
</stream:stream>
```

Step 8: Client responds to the challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Step 9: Server informs client of successful authentication:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Step 9 (alt): Server informs client of failed authentication:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <temporary-auth-failure/>
</failure>
</stream:stream>
```

Step 10: Client initiates a new stream to server:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Step 11: Server responds by sending a stream header to client along with any additional features (or an empty features element):

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
```

```
    id='c2s_345'
    from='example.com'
    version='1.0'>
<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
  <session xmlns='urn:ietf:params:xml:ns:xmpp-session'>
</stream:features>
```

6.6 Server-to-Server Example

The following example shows the data flow for a server authenticating with another server using SASL, normally after successful TLS negotiation (note: the alternate steps shown below are provided to illustrate the protocol for failure cases; they are not exhaustive and would not necessarily be triggered by the data sent in the example).

Step 1: Server1 initiates stream to Server2:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Step 2: Server2 responds with a stream tag sent to Server1:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='example.com'
    id='s2s_234'
    version='1.0'>
```

Step 3: Server2 informs Server1 of available authentication mechanisms:

```
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
    <mechanism>DIGEST-MD5</mechanism>
    <mechanism>KERBEROS_V4</mechanism>
  </mechanisms>
</stream:features>
```

Step 4: Server1 selects an authentication mechanism:

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
```

```
      mechanism='DIGEST-MD5'/>
```

Step 5: Server2 sends a base64 [14]-encoded challenge to Server1:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
dXNlcm5hbWU9InNvbWVkb21haW4iLHJlYWxtPSJzb21lcmVhbG0iLG5vbmNl
PSJPQTZNRzl0RVFHbTJoaCIscW9wPSJhdXRoIixjaGFyc2V0PXV0Zi04LGFs
Z29yaXRobT1tZDUtc2Vzcwo=
</challenge>
```

The decoded challenge is:

```
username="somedomain",realm="somerealm",\
nonce="OA6MG9tEQGm2hh",qop="auth",\
charset=utf-8,algorithm=md5-sess
```

Step 5 (alt): Server2 returns error to Server1:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <incorrect-encoding/>
</failure>
</stream:stream>
```

Step 6: Server1 sends a base64 [14]-encoded response to the challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
dXNlcm5hbWU9InNvbWVkb21haW4iLHJlYWxtPSJzb21lcmVhbG0iLG5vbmNl
PSJPQTZNRzl0RVFHbTJoaCIsY25vbmNlPSJPQTZNSFhoNlZxVHJSayIsbmM9
MDAwMDAwMDEscW9wPWF1dGgsZGlnZXN0LXVyaT0ieG1wcC9leGFtcGxlLmNv
bSIscmVzcG9uc2U9ZDM4OGRhZDkwZDRiYmQ3NjBhMTUyMzIxZjIxNDNhZjcs
Y2hhcnNldD11dGYtOAo=
</response>
```

The decoded response is:

```
username="somedomain",realm="somerealm",\
nonce="OA6MG9tEQGm2hh",cnonce="OA6MHXh6VqTrRk",\
nc=00000001,qop=auth,digest-uri="xmpp/example.com",\
response=d388dad90d4bbd760a152321f2143af7,charset=utf-8
```

Step 7: Server2 sends another base64 [14]-encoded challenge to Server1:

```
<challenge xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
cnNwYXV0aD1lYTQwZjYwMzM1YzQyN2I1NTI3Yjg0ZGJhYmNkZmZmZAo=
</challenge>
```

The decoded challenge is:

rspauth=ea40f60335c427b5527b84dbabcdfffd

Step 7 (alt): Server2 returns error to Server1:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <invalid-authzid/>
</failure>
</stream:stream>
```

Step 8: Server1 responds to the challenge:

```
<response xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Step 8 (alt): Server1 aborts negotiation:

```
<abort xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Step 9: Server2 informs Server1 of successful authentication:

```
<success xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
```

Step 9 (alt): Server2 informs Server1 of failed authentication:

```
<failure xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
  <aborted/>
</failure>
</stream:stream>
```

Step 10: Server1 initiates a new stream to Server2:

```
<stream:stream
    xmlns='jabber:server'
    xmlns:stream='http://etherx.jabber.org/streams'
    to='example.com'
    version='1.0'>
```

Step 11: Server2 responds by sending a stream header to Server1 along with any additional features (or an empty features element):

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    from='example.com'
    id='s2s_345'
    version='1.0'>
<stream:features/>
```

7. Resource Binding

After Stream Authentication (Section 6) with the receiving entity, the initiating entity MAY want or need to bind a specific resource to that stream. In general this applies only to clients: in order to conform to the addressing format (Section 3) and stanza delivery rules (Section 14) specified herein, there MUST be a resource identifier associated with the <node@domain> of the client (which is either generated by the server or provided by the client application); this ensures that the address for use over that stream is a "full JID" of the form <node@domain/resource>.

Upon receiving a success indication within the SASL negotiation, the client MUST send a new stream header to the server, to which the server MUST respond with a stream header as well as a list of available stream features. Specifically, if the server requires the client to bind a resource to the stream after successful stream authentication, it MUST include an empty <bind/> element qualified by the 'urn:ietf:params:xml:ns:xmpp-bind' namespace in the stream features list it presents to the client upon sending the header for the response stream sent after successful stream authentication (but not before):

Server advertises resource binding feature to client:

```
<stream:stream
    xmlns='jabber:client'
    xmlns:stream='http://etherx.jabber.org/streams'
    id='c2s_345'
    from='example.com'
    version='1.0'>
<stream:features>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
</stream:features>
```

Upon being so informed that resource binding is required, the client MUST bind a resource to the stream by sending to the server an IQ stanza of type "set" (see IQ Semantics (Section 9.2.3)) containing data qualified by the 'urn:ietf:params:xml:ns:xmpp-bind' namespace.

If the client wishes to allow the server to generate the resource identifier on its behalf, it sends an IQ stanza of type "set" that contains an empty <bind/> element:

Client asks server to bind a resource:

```
<iq type='set' id='bind_1'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'/>
```

```
</iq>
```

A server that supports resource binding MUST be able to generate a resource identifier on behalf of a client. A resource identifier generated by the server MUST be unique for that <node@domain>.

If the client wishes to specify the resource identifier, it sends an IQ stanza of type "set" that contains the desired resource identifier as the CDATA of a <resource/> element that is a child of the <bind/> element:

Client binds a resource:

```
<iq type='set' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
</iq>
```

Once the server has generated a resource identifier for the client or accepted the resource identifier provided by the client, it MUST return an IQ stanza of type "result" to the client, which MUST include a <jid/> child element that specifies the full JID for the client as determined by the server:

Server informs client of successful resource binding:

```
<iq type='result' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <jid>somenode@somedomain/someresource</jid>
  </bind>
</iq>
```

A server is NOT REQUIRED to accept the resource identifier provided by the client, and MAY override it with a resource identifier that the server generates; in this case, the server SHOULD NOT return a stanza error (e.g., <forbidden/>) to the client but instead SHOULD communicate the generated resource identifier to the client in the IQ result as shown above.

When a client supplies a resource identifier, the following stanza error conditions may occur (see Stanza Errors (Section 9.3)):

- The provided resource identifier cannot be processed by the server in accordance with Resourceprep (Appendix B).

- The client is not allowed to bind a resource to the stream (e.g., because the client has reached a limit on the number of bound

resources allowed).

o The provided resource identifier is already in use.

The protocol for these error conditions is shown below.

Resource identifier cannot be processed:

```
<iq type='error' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Client is not allowed to bind a resource:

```
<iq type='error' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
  <error type='cancel'>
    <not-allowed xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

Resource identifier is in use:

```
<iq type='error' id='bind_2'>
  <bind xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
    <resource>someresource</resource>
  </bind>
  <error type='cancel'>
    <conflict xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8. Server Dialback

8.1 Overview

The Jabber protocol from which XMPP was adapted includes a "server dialback" method for protecting against domain spoofing, thus making it more difficult to spoof XML stanzas (see Server-to-Server Communications (Section 13.3) regarding this method's security

characteristics). Server dialback also makes it easier to deploy systems in which outbound messages and inbound messages are handled by different machines for the same domain. The server dialback method is made possible by the existence of the Domain Name System (DNS), since one server can (normally) discover the authoritative server for a given domain.

Because dialback depends on DNS, inter-domain communications MUST NOT proceed until the DNS hostnames asserted by the servers have been resolved (see Server-to-Server Communications (Section 13.3)).

The method for generating and verifying the keys used in server dialback MUST take into account the hostnames being used, the random ID generated for the stream, and a secret known by the authoritative server's network.

Any error that occurs during dialback negotiation MUST be considered a stream error, resulting in termination of the stream and of the underlying TCP connection. The possible error conditions are specified in the protocol description below.

The following terminology applies:

- Originating Server -- the server that is attempting to establish a connection between two domains.
- Receiving Server -- the server that is trying to authenticate that Originating Server represents the domain which it claims to be.
- Authoritative Server -- the server that answers to the DNS hostname asserted by Originating Server; for basic environments this will be Originating Server, but it could be a separate machine in Originating Server's network.

8.2 Order of Events

The following is a brief summary of the order of events in dialback:

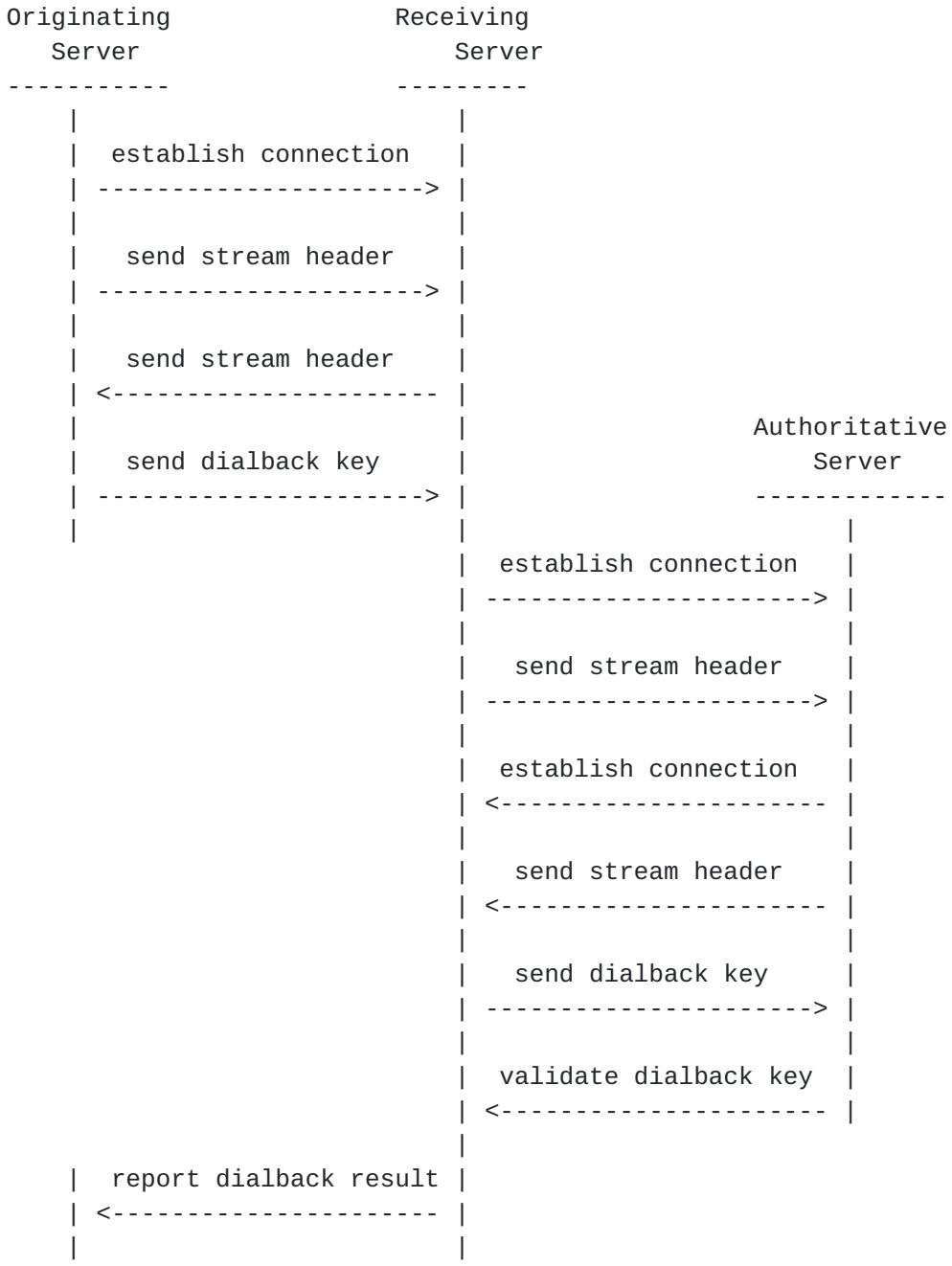
1. Originating Server establishes a connection to Receiving Server.
2. Originating Server sends a 'key' value over the connection to Receiving Server.
3. Receiving Server establishes a connection to Authoritative Server.
4. Receiving Server sends the same 'key' value to Authoritative

Server.

5. Authoritative Server replies that key is valid or invalid.

6. Receiving Server informs Originating Server whether it is authenticated or not.

We can represent this flow of events graphically as follows:



8.3 Protocol

The detailed protocol interaction between the servers is as follows:

1. Originating Server establishes TCP connection to Receiving Server.

2. Originating Server sends a stream header to Receiving Server:

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:server'
    xmlns:db='jabber:server:dialback'>
```

Note: the 'to' and 'from' attributes are NOT REQUIRED on the root stream element. The inclusion of the xmlns:db namespace declaration with the name shown indicates to Receiving Server that Originating Server supports dialback. If the namespace name is incorrect, then Receiving Server MUST generate an <invalid-namespace/> stream error condition and terminate both the XML stream and the underlying TCP connection.

3. Receiving Server SHOULD send a stream header back to Originating Server, including a unique ID for this interaction:

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:server'
    xmlns:db='jabber:server:dialback'
    id='457F9224A0...'>
```

Note: The 'to' and 'from' attributes are NOT REQUIRED on the root stream element. If the namespace name is incorrect, then Originating Server MUST generate an <invalid-namespace/> stream error condition and terminate both the XML stream and the underlying TCP connection. Note well that Receiving Server is NOT REQUIRED to reply and MAY silently terminate the XML stream and underlying TCP connection depending on security policies in place.

4. Originating Server sends a dialback key to Receiving Server:

```
<db:result
    to='Receiving Server'
    from='Originating Server'>
  98AF014EDC0...
</db:result>
```

Note: this key is not examined by Receiving Server, since Receiving Server does not keep information about Originating Server between sessions. The key generated by Originating Server MUST be based in part on the value of the ID provided by Receiving Server in the previous step, and in part on a secret shared by Originating Server and Authoritative Server. If the value of the 'to' address does not match a hostname recognized by Receiving Server, then Receiving Server MUST generate a <host-unknown/> stream error condition and terminate both the XML stream and the underlying TCP connection. If the value of the 'from' address matches a domain with which Receiving Server already has an established connection, then Receiving Server MUST maintain the existing connection until it validates whether the new connection is legitimate; additionally, Receiving Server MAY choose to generate a <not-authorized/> stream error condition for the new connection and then terminate both the XML stream and the underlying TCP connection related to the new request.

5. Receiving Server establishes a TCP connection back to the domain name asserted by Originating Server, as a result of which it connects to Authoritative Server. (Note: as an optimization, an implementation MAY reuse an existing trusted connection here rather than opening a new TCP connection.)

6. Receiving Server sends Authoritative Server a stream header:

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:server'
    xmlns:db='jabber:server:dialback'>
```

Note: the 'to' and 'from' attributes are NOT REQUIRED on the root stream element. If the namespace name is incorrect, then Authoritative Server MUST generate an <invalid-namespace/> stream error condition and terminate both the XML stream and the underlying TCP connection.

7. Authoritative Server sends Receiving Server a stream header:

```
<stream:stream
    xmlns:stream='http://etherx.jabber.org/streams'
    xmlns='jabber:server'
    xmlns:db='jabber:server:dialback'
    id='1251A342B...'>
```

Note: if the namespace name is incorrect, then Receiving Server MUST generate an <invalid-namespace/> stream error condition and

terminate both the XML stream and the underlying TCP connection between it and Authoritative Server. If a stream error occurs between Receiving Server and Authoritative Server, then Receiving Server MUST generate a <remote-connection-failed/> stream error condition and terminate both the XML stream and the underlying TCP connection between it and Originating Server.

8. Receiving Server sends Authoritative Server a stanza requesting that Authoritative Server verify a key:

```
<db:verify
    from='Receiving Server'
    to='Originating Server'
    id='457F9224A0...'>
  98AF014EDC0...
</db:verify>
```

Note: passed here are the hostnames, the original identifier from Receiving Server's stream header to Originating Server in Step 3, and the key that Originating Server sent to Receiving Server in Step 4. Based on this information as well as shared secret information within the Authoritative Server's network, the key is verified. Any verifiable method MAY be used to generate the key. If the value of the 'to' address does not match a hostname recognized by Authoritative Server, then Authoritative Server MUST generate a <host-unknown/> stream error condition and terminate both the XML stream and the underlying TCP connection. If the value of the 'from' address does not match the hostname represented by Receiving Server when opening the TCP connection (or any validated domain), then Authoritative Server MUST generate an <invalid-from/> stream error condition and terminate both the XML stream and the underlying TCP connection.

9. Authoritative Server sends a stanza back to Receiving Server verifying whether the key was valid or invalid:

```
<db:verify
    from='Originating Server'
    to='Receiving Server'
    type='valid'
    id='457F9224A0...'/>
```

or

```
<db:verify
    from='Originating Server'
    to='Receiving Server'
```

```
    type='invalid'
    id='457F9224A0...'/>
```

Note: if the ID does not match that provided by Receiving Server in Step 3, then Receiving Server MUST generate an <invalid-id/> stream error condition and terminate both the XML stream and the underlying TCP connection. If the value of the 'to' address does not match a hostname recognized by Receiving Server, then Receiving Server MUST generate a <host-unknown/> stream error condition and terminate both the XML stream and the underlying TCP connection. If the value of the 'from' address does not match the hostname represented by Originating Server when opening the TCP connection (or any validated domain), then Receiving Server MUST generate an <invalid-from/> stream error condition and terminate both the XML stream and the underlying TCP connection.

10. Receiving Server informs Originating Server of the result:

```
<db:result
    from='Receiving Server'
    to='Originating Server'
    type='valid'/>
```

Note: At this point the connection has either been validated via a type='valid', or reported as invalid. If the connection is invalid, then Receiving Server MUST terminate both the XML stream and the underlying TCP connection. If the connection is validated, data can be sent by Originating Server and read by Receiving Server; before that, all data stanzas sent to Receiving Server SHOULD be silently dropped.

Even if dialback negotiation is successful, a server MUST verify that all XML stanzas received from the other server include a 'from' attribute and a 'to' attribute; if a stanza does not meet this restriction, the server that receives the stanza MUST generate an <improper-addressing/> stream error condition and terminate both the XML stream and the underlying TCP connection. Furthermore, a server MUST verify that the 'from' attribute of stanzas received from the other server includes a validated domain for the stream; if a stanza does not meet this restriction, the server that receives the stanza MUST generate an <invalid-from/> stream error condition and terminate both the XML stream and the underlying TCP connection. Both of these checks help to prevent spoofing related to particular stanzas.

9. XML Stanzas

After Stream Encryption (Section 5) if desired, Stream Authentication

(Section 6), and Resource Binding (Section 7) if necessary, XML stanzas can be sent over the streams. Three kinds of XML stanza are defined for the 'jabber:client' and 'jabber:server' namespaces: <message/>, <presence/>, and <iq/>. In addition, there are five common attributes for these kinds of stanza. These common attributes, as well as the basic semantics of the three stanza kinds, are defined herein; more detailed information regarding the syntax of XML stanzas in relation to instant messaging and presence applications is provided in XMPP IM [21].

9.1 Common Attributes

The following five attributes are common to message, presence, and IQ stanzas:

9.1.1 to

The 'to' attribute specifies the JID of the intended recipient for the stanza.

In the 'jabber:client' namespace, a stanza SHOULD possess a 'to' attribute, although a stanza sent from a client to a server for handling by that server (e.g., presence sent to the server for broadcasting to other entities) SHOULD NOT possess a 'to' attribute.

In the 'jabber:server' namespace, a stanza MUST possess a 'to' attribute; if a server receives a stanza that does not meet this restriction, it MUST generate an <improper-addressing/> stream error condition and terminate both the XML stream and the underlying TCP connection with the offending server.

If the value of the 'to' attribute is invalid or cannot be contacted, the entity discovering that fact (usually the sender's or recipient's server) MUST return an appropriate error to the sender, setting the 'from' attribute of the error stanza to the value provided in the 'to' attribute of the offending stanza.

9.1.2 from

The 'from' attribute specifies the JID of the sender.

When a server receives an XML stanza within the context of an authenticated stream qualified by the 'jabber:client' namespace, it MUST do one of the following:

1. validate that the value of the 'from' attribute provided by the client is that of an authorized resource for the associated entity

2. add a 'from' address to the stanza whose value is the full JID (<node@domain/resource>) determined by the server for the connected resource that generated the stanza (see Determination of Addresses (Section 3.6))

If a client attempts to send an XML stanza for which the value of the 'from' attribute does not match one of the connected resources for that entity, the server SHOULD return an <invalid-from/> stream error to the client. If a client attempts to send an XML stanza over a stream that is not yet authenticated, the server SHOULD return a <not-authorized/> stream error to the client. If generated, both of these conditions MUST result in closing of the stream and termination of the underlying TCP connection; this helps to prevent a denial of service attack launched from a rogue client.

In the 'jabber:server' namespace, a stanza MUST possess a 'from' attribute; if a server receives a stanza that does not meet this restriction, it MUST generate an <improper-addressing/> stream error condition. Furthermore, the domain identifier portion of the JID contained in the 'from' attribute MUST match the hostname (or any validated domain) of the sending server as communicated in the SASL negotiation or dialback negotiation; if a server receives a stanza that does not meet this restriction, it MUST generate an <invalid-from/> stream error condition. Both of these conditions MUST result in closing of the stream and termination of the underlying TCP connection; this helps to prevent a denial of service attack launched from a rogue server.

9.1.3 id

The optional 'id' attribute MAY be used by a sending entity for internal tracking of stanzas that it sends and receives (especially for tracking the request-response interaction inherent in the semantics of IQ stanzas). The value of the 'id' attribute is NOT REQUIRED to be unique either globally, within a domain, or within a stream. The semantics of IQ stanzas impose additional restrictions; see IQ Semantics (Section 9.2.3).

9.1.4 type

The 'type' attribute specifies detailed information about the purpose or context of the message, presence, or IQ stanza. The particular allowable values for the 'type' attribute vary depending on whether the stanza is a message, presence, or IQ; the values for message and presence stanzas are specific to instant messaging and presence applications and therefore are defined in XMPP IM [21], whereas the values for IQ stanzas specify the role of an IQ stanza in a structured request-response "conversation" and thus are defined under

IQ Semantics (Section 9.2.3) below. The only 'type' value common to all three stanzas is "error", for which see Stanza Errors (Section 9.3).

9.1.5 xml:lang

A stanza SHOULD possess an 'xml:lang' attribute (as defined in Section 2.12 of the XML specification [1]) if the stanza contains XML character data that is intended to be presented to a human user (as explained in RFC 2277 [15], "internationalization is for humans"). The value of the 'xml:lang' attribute specifies the default language of any such human-readable XML character data, which MAY be overridden by the 'xml:lang' attribute of a specific child element. If a stanza does not possess an 'xml:lang' attribute, an implementation MUST assume that the default language is that specified for the stream as defined under Stream Attributes (Section 4.2) above. The value of the 'xml:lang' attribute MUST be an NMTOKEN and MUST conform to the format defined in RFC 3066 [16].

9.2 Basic Semantics

9.2.1 Message Semantics

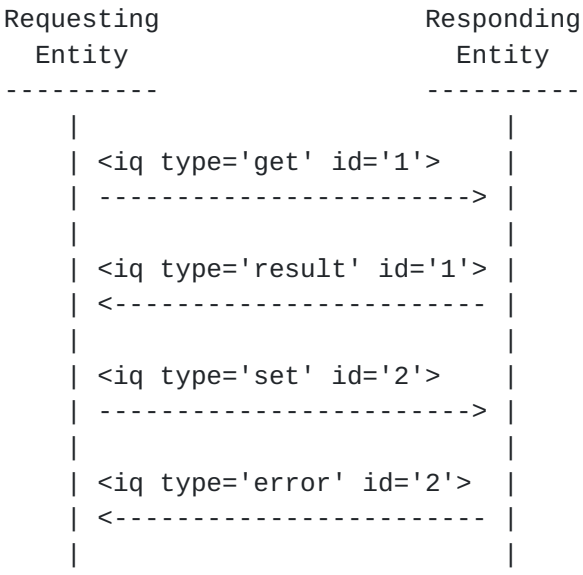
The <message/> stanza kind can be seen as a "push" mechanism whereby one entity pushes information to another entity, similar to the communications that occur in a system such as email. All message stanzas SHOULD possess a 'to' attribute that specifies the intended recipient of the message; upon receiving such a stanza, a server SHOULD route or deliver it to the intended recipient (see Server Rules for Handling XML Stanzas (Section 14) for general routing and delivery rules related to XML stanzas).

9.2.2 Presence Semantics

The <presence/> element can be seen as a basic broadcast or "publish-subscribe" mechanism, whereby multiple entities receive information (in this case, presence information) about an entity to which they have subscribed. In general, a publishing entity SHOULD send a presence stanza with no 'to' attribute, in which case the server to which the entity is connected SHOULD broadcast or multiplex that stanza to all subscribing entities. However, a publishing entity MAY also send a presence stanza with a 'to' attribute, in which case the server SHOULD route or deliver that stanza to the intended recipient. See Server Rules for Handling XML Stanzas (Section 14) for general routing and delivery rules related to XML stanzas, and XMPP IM [21] for presence-specific rules in the context of an instant messaging and presence application.

9.2.3 IQ Semantics

Info/Query, or IQ, is a request-response mechanism, similar in some ways to HTTP [22]. The semantics of IQ enable an entity to make a request of, and receive a response from, another entity. The data content of the request and response is defined by the namespace declaration of a direct child element of the IQ element, and the interaction is tracked by the requesting entity through use of the 'id' attribute. Thus IQ interactions follow a common pattern of structured data exchange such as get/result or set/result (although an error may be returned in reply to a request if appropriate):



In order to enforce these semantics, the following rules apply:

1. The 'id' attribute is REQUIRED for IQ stanzas.

2. The 'type' attribute is REQUIRED for IQ stanzas. The value SHOULD be one of the following (all other values SHOULD be ignored):

   * get -- The stanza is a request for information or requirements.

   * set -- The stanza provides required data, sets new values, or replaces existing values.

   * result -- The stanza is a response to a successful get or set request.

   * error -- An error has occurred regarding processing or delivery of a previously-sent get or set (see Stanza Errors (Section 9.3)).

3. An entity that receives an IQ request of type "get" or "set" MUST reply with an IQ response of type "result" or "error" (which response MUST preserve the 'id' attribute of the request).

4. An entity that receives a stanza of type "result" or "error" MUST NOT respond to the stanza by sending a further IQ response of type "result" or "error"; however, as shown above, the requesting entity MAY send another request (e.g., an IQ of type "set" in order to provide required information discovered through a get/result pair).

5. An IQ stanza of type "get" or "set" MUST contain one and only one child element (properly-namespaced as defined in XMPP IM [21]) that specifies the semantics of the particular request or response.

6. An IQ stanza of type "result" MUST include zero or one child elements.

7. An IQ stanza of type "error" SHOULD include the child element contained in the associated "get" or "set" and MUST include an <error/> child; for details, see Stanza Errors (Section 9.3).

9.3 Stanza Errors

Stanza-related errors are handled in a manner similar to stream errors (Section 4.6). However, stanza errors are not unrecoverable, as stream errors are; therefore error stanzas include hints regarding actions that the original sender can take in order to remedy the error.

9.3.1 Rules

The following rules apply to stanza-related errors:

- The receiving or processing entity that detects an error condition in relation to a stanza MUST return to the sending entity a stanza of the same kind (message, presence, or IQ) whose 'type' attribute is set to a value of "error" (such a stanza is called an "error stanza" herein).

- The entity that generates an error stanza SHOULD (but is NOT REQUIRED to) include the original XML sent so that the sender can inspect and if necessary correct the XML before attempting to resend.

- An error stanza MUST contain an <error/> child element.

- An <error/> child MUST NOT be included if the 'type' attribute has a value other than "error" (or if there is no 'type' attribute).

- An entity that receives an error stanza MUST NOT respond to the stanza with a further error stanza; this helps to prevent looping.

9.3.2 Syntax

The syntax for stanza-related errors is as follows:

```
<stanza-name to='sender' type='error'>
  [RECOMMENDED to include sender XML here]
  <error type='error-type'>
    <defined-condition xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <text xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      OPTIONAL descriptive text
    </text>
    [OPTIONAL application-specific condition element]
  </error>
</stanza-name>
```

The stanza-name is one of message, presence, or iq.

The value of the <error/> element's 'type' attribute MUST be one of the following:

- cancel -- do not retry (the error is unrecoverable)

- continue -- proceed (the condition was only a warning)

- modify -- retry after changing the data sent

- auth -- retry after providing credentials

- wait -- retry after waiting (the error is temporary)

The <error/> element:

- MUST contain a child element corresponding to one of the defined stanza error conditions specified below; this element MUST be qualified by the 'urn:ietf:params:xml:ns:xmpp-stanzas' namespace.

- MAY contain a <text/> child containing CDATA that describes the error in more detail; this element MUST be qualified by the 'urn:ietf:params:xml:ns:xmpp-stanzas' namespace and SHOULD possess an 'xml:lang' attribute.

- MAY contain a child element for an application-specific error condition; this element MUST be qualified by an application-defined namespace, and its structure is defined by that namespace.

The <text/> element is OPTIONAL. If included, it SHOULD be used only to provide descriptive or diagnostic information that supplements the meaning of a defined condition or application-specific condition. It SHOULD NOT be interpreted programmatically by an application. It SHOULD NOT be used as the error message presented to a user, but MAY be shown in addition to the error message associated with the included condition element (or elements).

Note: the XML namespace name 'urn:ietf:params:xml:ns:xmpp-stanzas' that qualifies the descriptive element adheres to the format defined in The IETF XML Registry [24].

9.3.3 Defined Conditions

The following stanza-related error conditions are defined for use in stanza errors.

- <bad-request/> -- the sender has sent XML that is malformed or that cannot be processed (e.g., an IQ stanza that includes an unrecognized value of the 'type' attribute); the associated error type SHOULD be "modify".

- <conflict/> -- access cannot be granted because an existing resource or session exists with the same name or address; the associated error type SHOULD be "cancel".

- <feature-not-implemented/> -- the feature requested is not implemented by the recipient or server and therefore cannot be processed; the associated error type SHOULD be "cancel".

- <forbidden/> -- the requesting entity does not possess the required permissions to perform the action; the associated error type SHOULD be "auth".

- <gone/> -- the recipient or server can no longer be contacted at this address (the error stanza MAY contain a new address in the CDATA of the <gongone/> element); the associated error type SHOULD be "modify".

- <internal-server-error/> -- the server could not process the stanza because of a misconfiguration or an otherwise-undefined internal server error; the associated error type SHOULD be "wait".

- <item-not-found/> -- the addressed JID or item requested cannot be found; the associated error type SHOULD be "cancel".

- <jid-malformed/> -- the value of the 'to' attribute in the sender's stanza does not adhere to the syntax defined in Addressing Scheme (Section 3); the associated error type SHOULD be "modify".

- <not-acceptable/> -- the recipient or server understands the request but is refusing to process it because it does not meet criteria defined by the recipient or server (e.g., a local policy regarding acceptable words in messages); the associated error type SHOULD be "cancel".

- <not-allowed/> -- the recipient or server does not allow any entity to perform the action; the associated error type SHOULD be "cancel".

- <payment-required/> -- the requesting entity is not authorized to access the requested service because payment is required; the associated error type SHOULD be "auth".

- <recipient-unavailable/> -- the intended recipient is temporarily unavailable; the associated error type SHOULD be "wait" (note: an application MUST NOT return this error if doing so would provide information about the intended recipient's network availability to an entity that is not authorized to know such information).

- <redirect/> -- the recipient or server is redirecting requests for this information to another entity, usually temporarily (the error stanza MAY contain a new address in the CDATA of the <redirect/> element); the associated error type SHOULD be "modify".

- <registration-required/> -- the requesting entity is not authorized to access the requested service because registration is required; the associated error type SHOULD be "auth".

- <remote-server-not-found/> -- a remote server or service specified as part or all of the JID of the intended recipient does not exist; the associated error type SHOULD be "cancel".

- <remote-server-timeout/> -- a remote server or service specified as part or all of the JID of the intended recipient could not be contacted within a reasonable amount of time; the associated error type SHOULD be "wait".

- <resource-constraint/> -- the server or recipient lacks the system resources necessary to service the request; the associated error

type SHOULD be "wait".

- <service-unavailable/> -- the server or recipient does not currently provide the requested service; the associated error type SHOULD be "cancel".

- <subscription-required/> -- the requesting entity is not authorized to access the requested service because a subscription is required; the associated error type SHOULD be "auth".

- <undefined-condition/> -- the error condition is not one of those defined by the other conditions in this list; any error type may be associated with this condition, and it SHOULD be used only in conjunction with an application-specific condition.

- <unexpected-request/> -- the recipient or server understood the request but was not expecting it at this time (e.g., the request was out of order); the associated error type SHOULD be "wait".

9.3.4 Application-Specific Conditions

As noted, an application MAY provide application-specific stanza error information by including a properly-namespaced child in the error element. The application-specific element SHOULD supplement or further qualify a defined element. Thus the <error/> element will contain two or three child elements:

```
<iq type='error' id='some-id'>
  <error type='modify'>
    <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <too-many-parameters xmlns='application-ns'/>
  </error>
</iq>


<message type='error' id='another-id'>
  <error type='modify'>
    <undefined-condition
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
    <text xml:lang='en'
          xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'>
      Some special application diagnostic information...
    </text>
    <special-application-condition xmlns='application-ns'/>
  </error>
</message>
```

10. XML Usage within XMPP

10.1 Restrictions

XMPP is a simplified and specialized protocol for streaming XML elements in order to exchange structured information in close to real time. Because XMPP does not require the parsing of arbitrary and complete XML documents, there is no requirement that XMPP needs to support the full XML specification [1]. In particular, the following restrictions apply.

With regard to XML generation, an XMPP implementation MUST NOT inject into an XML stream any of the following:

- comments (as defined in Section 2.5 of the XML specification [1])
- processing instructions (Section 2.6 therein)
- internal or external DTD subsets (Section 2.8 therein)
- internal or external entity references (Section 4.2 therein) with the exception of predefined entities (Section 4.6 therein)
- character data or attribute values containing unescaped characters that map to the predefined entities (Section 4.6 therein); such characters MUST be escaped

With regard to XML processing, if an XMPP implementation receives such restricted XML data, it MUST ignore the data.

10.2 XML Namespace Names and Prefixes

XML Namespaces [10] are used within all XMPP-compliant XML to create strict boundaries of data ownership. The basic function of namespaces is to separate different vocabularies of XML elements that are structurally mixed together. Ensuring that XMPP-compliant XML is namespace-aware enables any allowable XML to be structurally mixed with any data element within XMPP. Rules for XML namespace names and prefixes are defined in the following subsections.

10.2.1 Streams Namespace

A streams namespace declaration is REQUIRED in all XML stream headers. The name of the streams namespace MUST be 'http://etherx.jabber.org/streams'. The element names of the <stream/> element and its <features/> and <error/> children MUST be qualified by the streams namespace prefix in all instances. An implementation SHOULD generate only the 'stream:' prefix for these elements, and for

historical reasons MAY accept only the 'stream:' prefix.

10.2.2 Default Namespace

A default namespace declaration is REQUIRED and is used in all XML streams in order to define the allowable first-level children of the root stream element. This namespace declaration MUST be the same for the initial stream and the response stream so that both streams are qualified consistently. The default namespace declaration applies to the stream and all stanzas sent within a stream (unless explicitly qualified by another namespace, or by the prefix of the streams namespace or the dialback namespace).

A server implementation MUST support the following two default namespaces (for historical reasons, some implementations MAY support only these two default namespaces):

- jabber:client -- this default namespace is declared when the stream is used for communications between a client and a server

- jabber:server -- this default namespace is declared when the stream is used for communications between two servers

A client implementation MUST support the 'jabber:client' default namespace, and for historical reasons MAY support only that default namespace.

An implementation MUST NOT generate namespace prefixes for elements in the default namespace if the default namespace is 'jabber:client' or 'jabber:server'. An implementation SHOULD NOT generate namespace prefixes for elements qualified by content (as opposed to stream) namespaces other than 'jabber:client' and 'jabber:server'.

Note: the 'jabber:client' and 'jabber:server' namespaces are nearly identical but are used in different contexts (client-to-server communications for 'jabber:client' and server-to-server communications for 'jabber:server'). The only difference between the two is that the 'to' and 'from' attributes are OPTIONAL on stanzas sent within 'jabber:client', whereas they are REQUIRED on stanzas sent within 'jabber:server'. If a compliant implementation accepts a stream that is qualified by the 'jabber:client' or 'jabber:server' namespace, it MUST support the common attributes (Section 9.1) and basic semantics (Section 9.2) of all three core stanza kinds (message, presence, and IQ).

10.2.3 Dialback Namespace

A dialback namespace declaration is REQUIRED for all elements used in

server dialback (Section 8). The name of the dialback namespace MUST be 'jabber:server:dialback'. All elements qualified by this namespace MUST be prefixed. An implementation SHOULD generate only the 'db:' prefix for such elements and MAY accept only the 'db:' prefix.

10.3 Validation

Except as noted with regard to 'to' and 'from' addresses for stanzas within the 'jabber:server' namespace, a server is not responsible for validating the XML elements forwarded to a client or another server; an implementation MAY choose to provide only validated data elements but is NOT REQUIRED to do so (although an implementation MUST NOT accept XML that is not well-formed). Clients SHOULD NOT rely on the ability to send data which does not conform to the schemas, and SHOULD ignore any non-conformant elements or attributes on the incoming XML stream. Validation of XML streams and stanzas is NOT REQUIRED or recommended, and schemas are included herein for descriptive purposes only.

10.4 Inclusion of Text Declaration

Implementations SHOULD send a text declaration before sending a stream header. Applications MUST follow the rules in the XML specification [1] regarding the circumstances under which a text declaration is included.

10.5 Character Encoding

Implementations MUST support the UTF-8 (RFC 2279 [17]) transformation of Universal Character Set (ISO/IEC 10646-1 [18]) characters, as required by RFC 2277 [15]. Implementations MUST NOT attempt to use any other encoding.

11. IANA Considerations

11.1 XML Namespace Name for TLS Data

A URN sub-namespace for TLS-related data in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows.

URI: urn:ietf:params:xml:ns:xmpp-tls

Specification: [RFCXXXX]

Description: This is the XML namespace name for TLS-related data in the Extensible Messaging and Presence Protocol (XMPP) as defined by [RFCXXXX].

Registrant Contact: IETF, XMPP Working Group, <xmppwg@jabber.org>

11.2 XML Namespace Name for SASL Data

A URN sub-namespace for SASL-related data in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows.

URI: urn:ietf:params:xml:ns:xmpp-sasl

Specification: [RFCXXXX]

Description: This is the XML namespace name for SASL-related data in the Extensible Messaging and Presence Protocol (XMPP) as defined by [RFCXXXX].

Registrant Contact: IETF, XMPP Working Group, <xmppwg@jabber.org>

11.3 XML Namespace Name for Stream Errors

A URN sub-namespace for stream-related error data in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows.

URI: urn:ietf:params:xml:ns:xmpp-streams

Specification: [RFCXXXX]

Description: This is the XML namespace name for stream-related error data in the Extensible Messaging and Presence Protocol (XMPP) as defined by [RFCXXXX].

Registrant Contact: IETF, XMPP Working Group, <xmppwg@jabber.org>

11.4 XML Namespace Name for Resource Binding

A URN sub-namespace for resource binding in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows.

URI: urn:ietf:params:xml:ns:xmpp-bind

Specification: [RFCXXXX]

Description: This is the XML namespace name for resource binding in the Extensible Messaging and Presence Protocol (XMPP) as defined by [RFCXXXX].

Registrant Contact: IETF, XMPP Working Group, <xmppwg@jabber.org>

11.5 XML Namespace Name for Stanza Errors

A URN sub-namespace for stanza-related error data in the Extensible Messaging and Presence Protocol (XMPP) is defined as follows.

URI: urn:ietf:params:xml:ns:xmpp-stanzas

Specification: [RFCXXXX]

Description: This is the XML namespace name for stanza-related error data in the Extensible Messaging and Presence Protocol (XMPP) as defined by [RFCXXXX].

Registrant Contact: IETF, XMPP Working Group, <xmppwg@jabber.org>

11.6 Nodeprep Profile of Stringprep

The Nodeprep profile of stringprep is defined under Nodeprep (Appendix A). The IANA registers Nodeprep in the stringprep profile registry.

Name of this profile:

Nodeprep

RFC in which the profile is defined:

[RFCXXXX]

Indicator whether or not this is the newest version of the profile:

This is the first version of Nodeprep

11.7 Resourceprep Profile of Stringprep

The Resourceprep profile of stringprep is defined under Resourceprep (Appendix B). The IANA registers Resourceprep in the stringprep profile registry.

Name of this profile:

Resourceprep

RFC in which the profile is defined:

[RFCXXXX]

Indicator whether or not this is the newest version of the profile:

This is the first version of Resourceprep

11.8 GSSAPI Service Name

The IANA registers "xmpp" as a GSSAPI [19] service name, as defined under SASL Definition (Section 6.3).

11.9 Port Numbers

The IANA currently registers "jabber-client" and "jabber-server" as keywords for TCP ports 5222 and 5269 respectively. The IANA shall change these registrations to "xmpp-client" and "xmpp-server" respectively.

These ports SHOULD be used for client-to-server and server-to-server communications respectively, but their use is NOT REQUIRED.

12. Internationalization Considerations

XML streams MUST be encoded in UTF-8 as specified under Character Encoding (Section 10.5). As specified under Stream Attributes (Section 4.2), an XML stream SHOULD include an 'xml:lang' attribute that is treated as the default language for any XML character data sent over the stream that is intended to be presented to a human user. As specified under xml:lang (Section 9.1.5), an XML stanza SHOULD include an 'xml:lang' attribute if the stanza contains XML character data that is intended to be presented to a human user. A server SHOULD apply the default 'xml:lang' attribute to stanzas it routes or delivers on behalf of connected entities, and MUST NOT modify or delete 'xml:lang' attributes from stanzas it receives from other entities.

13. Security Considerations

13.1 High Security

For the purposes of XMPP communications (client-to-server and server-to-server), the term "high security" refers to the use of security technologies that provide both mutual authentication and

integrity-checking; in particular, when using certificate-based authentication to provide high security, a chain-of-trust SHOULD be established out-of-band, although a shared certificate authority signing certificates could allow a previously unknown certificate to establish trust in-band.

Standalone, self-signed service certificates SHOULD NOT be used; rather, an entity that wishes to generate a self-signed service certificate SHOULD first generate a self-signed Root CA certificate and then generate a signed service certificate. Entities that communicate with the service SHOULD be configured with the Root CA certificate rather than the service certificate; this avoids problems associated with simple comparison of service certificates. If a self-signed service certificate is used, an entity SHOULD NOT trust it if it is changed to another self-signed certificate or a certificate signed by an unrecognized authority.

Implementations MUST support high security. Service provisioning SHOULD use high security, subject to local security policies.

13.2 Client-to-Server Communications

A compliant implementation MUST support both TLS and SASL for connections to a server.

The TLS protocol for encrypting XML streams (defined under Stream Encryption (Section 5)) provides a reliable mechanism for helping to ensure the confidentiality and data integrity of data exchanged between two entities.

The SASL protocol for authenticating XML streams (defined under Stream Authentication (Section 6)) provides a reliable mechanism for validating that a client connecting to a server is who it claims to be.

Client-to-server communications MUST NOT proceed until the DNS hostname asserted by the server has been resolved. Such resolutions SHOULD first attempt to resolve the hostname using an SRV [20] Service of "xmpp-client" and Proto of "tcp", resulting in resource records such as "_xmpp-client._tcp.example.com." (the use of the string "xmpp-client" for the service identifier is consistent with the IANA registration). If the SRV lookup fails, the fallback is a normal IPv4/IPv6 address record resolution to determine the IP address, using the "xmpp-client" port of 5222 assigned by the Internet Assigned Numbers Authority [5].

The IP address and method of access of clients MUST NOT be made available by a server, nor are any connections other than the

original server connection required. This helps to protect the client's server from direct attack or identification by third parties.

13.3 Server-to-Server Communications

A compliant implementation MUST support both TLS and SASL for inter-domain communications. For historical reasons, a compliant implementation SHOULD also support Server Dialback (Section 8).

Because service provisioning is a matter of policy, it is OPTIONAL for any given domain to communicate with other domains, and server-to-server communications MAY be disabled by the administrator of any given deployment. If a particular domain enables inter-domain communications, it SHOULD enable high security.

Administrators may want to require use of SASL for server-to-server communications in order to ensure both authentication and confidentiality (e.g., on an organization's private network). Compliant implementations SHOULD support SASL for this purpose.

Inter-domain connections MUST NOT proceed until the DNS hostnames asserted by the servers have been resolved. Such resolutions MUST first attempt to resolve the hostname using an SRV [20] Service of "xmpp-server" and Proto of "tcp", resulting in resource records such as "_xmpp-server._tcp.example.com." (the use of the string "xmpp-server" for the service identifier is consistent with the IANA registration; note well that the "xmpp-server" service identifier supersedes the earlier use of a "jabber" service identifier, since the earlier usage did not conform to RFC 2782 [20]; implementations desiring to be backwards compatible should continue to look for or answer to the "jabber" service identifier as well). If the SRV lookup fails, the fallback is a normal IPv4/IPv6 address record resolution to determine the IP address, using the "xmpp-server" port of 5269 assigned by the Internet Assigned Numbers Authority [5].

Server dialback helps protect against domain spoofing, thus making it more difficult to spoof XML stanzas. It is not a mechanism for authenticating, securing, or encrypting streams between servers as is done via SASL and TLS. Furthermore, it is susceptible to DNS poisoning attacks unless DNSSec [29] is used, and even if the DNS information is accurate, dialback cannot protect from attacks where the attacker is capable of hijacking the IP address of the remote domain. Domains requiring robust security SHOULD use TLS and SASL. If SASL is used for server-to-server authentication, dialback SHOULD NOT be used since it is unnecessary.

13.4 Order of Layers

The order of layers in which protocols MUST be stacked is as follows:

1. TCP
2. TLS
3. SASL
4. XMPP

The rationale for this order is that TCP is the base connection layer used by all of the protocols stacked on top of TCP, TLS is often provided at the operating system layer, SASL is often provided at the application layer, and XMPP is the application itself.

13.5 Mandatory-to-Implement Technologies

At a minimum, all implementations MUST support the following mechanisms:

for authentication: the SASL DIGEST-MD5 mechanism

for confidentiality: TLS (using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher)

for both: TLS plus SASL EXTERNAL(using the TLS_RSA_WITH_3DES_EDE_CBC_SHA cipher supporting client-side certificates)

13.6 Firewalls

Communications using XMPP normally occur over TCP sockets on port 5222 (client-to-server) or port 5269 (server-to-server), as registered with the IANA [5] (see IANA Considerations (Section 11)). Use of these well-known ports allows administrators to easily enable or disable XMPP activity through existing and commonly-deployed firewalls.

13.7 Use of base64 in SASL

Both the client and the server SHOULD verify any base64 [14] data received during SASL negotiation. An implementation MUST reject (not ignore) any characters that are not explicitly allowed by the base64 alphabet; this helps to guard against creation of a covert channel that could be used to "leak" information. An implementation MUST NOT

break on invalid input and MUST reject any sequence of base64 characters containing the pad ('=') character if that character is included as something other than the last character of the data (e.g. "=AAA" or "BBBB=CCC"); this helps to guard against buffer overflow attacks and other attacks on the implementation. Base encoding visually hides otherwise easily recognized information, such as passwords, but does not provide any computational confidentiality. Base 64 encoding MUST follow the definition in Section 3 of RFC 3548 [14].

13.8 Stringprep Profiles

XMPP makes use of the Nameprep [6] profile of stringprep [7] for processing of domain identifiers; for security considerations related to Nameprep, refer to the appropriate section of RFC 3491.

In addition, XMPP defines two profiles of stringprep [7]: Nodeprep (Appendix A) for node identifiers and Resourceprep (Appendix B) for resource identifiers.

The Unicode and ISO/IEC 10646 repertoires have many characters that look similar. In many cases, users of security protocols might do visual matching, such as when comparing the names of trusted third parties. Because it is impossible to map similar-looking characters without a great deal of context such as knowing the fonts used, stringprep does nothing to map similar-looking characters together nor to prohibit some characters because they look like others.

A node identifier can be employed as one part of an entity's address in XMPP. One common usage is as the username of an instant messaging user; another is as the name of a multi-user chat room; and many other kinds of entities could use node identifiers as part of their addresses. The security of such services could be compromised based on different interpretations of the internationalized node identifier; for example, a user entering a single internationalized node identifier could access another user's account information, or a user could gain access to an otherwise restricted chat room or service.

A resource identifier can be employed as one part of an entity's address in XMPP. One common usage is as the name for an instant messaging user's active session; another is as the nickname of a user in a multi-user chat room; and many other kinds of entities could use resource identifiers as part of their addresses. The security of such services could be compromised based on different interpretations of the internationalized resource identifier; for example, a user could attempt to initiate multiple sessions with the same name, or a user could send a message to someone other than the intended recipient in

a multi-user chat room.

14. Server Rules for Handling XML Stanzas

Each server implementation will contain its own "delivery tree" for handling stanzas it receives. Such a tree determines whether a stanza needs to be routed to another domain, processed internally, or delivered to a resource associated with a connected node. The following rules apply:

14.1 No 'to' Address

If the stanza possesses no 'to' attribute, the server SHOULD process it on behalf of the entity that sent it. Because all stanzas received from other servers MUST possess a 'to' attribute, this rule applies only to stanzas received from a registered entity (such as a client) that is connected to the server. If the server receives a presence stanza with no 'to' attribute, the server SHOULD broadcast it to the entities that are subscribed to the sending entity's presence, if applicable (the semantics of presence broadcast for instant messaging and presence applications are defined in XMPP IM [21]). If the server receives an IQ stanza of type "get" or "set" with no 'to' attribute and it understands the namespace that qualifies the content of the stanza, it MUST either process the stanza on behalf of sending entity (where the meaning of "process" is determined by the semantics of the qualifying namespace) or return an error to the sending entity.

14.2 Foreign Domain

If the hostname of the domain identifier portion of the JID contained in the 'to' attribute does not match one of the configured hostnames of the server itself or a subdomain thereof, the server SHOULD route the stanza to the foreign domain (subject to local service provisioning and security policies regarding inter-domain communication). There are two possible cases:

A server-to-server stream already exists between the two domains: The sender's server routes the stanza to the authoritative server for the foreign domain over the existing stream

There exists no server-to-server stream between the two domains: The sender's server (1) resolves the hostname of the foreign domain (as defined under Server-to-Server Communications (Section 13.3)), (2) negotiates a server-to-server stream between the two domains (as defined under Stream Encryption (Section 5) and Stream Authentication (Section 6)), and (3) routes the stanza to the authoritative server for the foreign domain over the newly-established stream

If routing to the recipient's server is unsuccessful, the sender's server MUST return an error to the sender; if the recipient's server can be contacted but delivery by the recipient's server to the recipient is unsuccessful, the recipient's server MUST return an error to the sender by way of the sender's server.

14.3 Subdomain

If the hostname of the domain identifier portion of the JID contained in the 'to' attribute matches a subdomain of one of the configured hostnames of the server itself, the server MUST either process the stanza itself or route the stanza to a specialized service that is responsible for that subdomain (if the subdomain is configured), or return an error to the sender (if the subdomain is not configured).

14.4 Mere Domain or Specific Resource

If the hostname of the domain identifier portion of the JID contained in the 'to' attribute matches a configured hostname of the server itself and the JID contained in the 'to' attribute is of the form <domain> or <domain/resource>, the server (or a defined resource thereof) MUST either process the stanza as appropriate for the stanza kind or return an error stanza to the sender.

14.5 Node in Same Domain

If the hostname of the domain identifier portion of the JID contained in the 'to' attribute matches a configured hostname of the server itself and the JID contained in the 'to' attribute is of the form <node@domain> or <node@domain/resource>, the server SHOULD deliver the stanza to the intended recipient of the stanza as represented by the JID contained in the 'to' attribute. The following rules apply:

1. If the JID contains a resource identifier (i.e., is of the form <node@domain/resource>) and there is an available resource that matches the full JID, the recipient's server SHOULD deliver the stanza to the stream or session that exactly matches the resource identifier.

2. If the JID contains a resource identifier and there is no available resource that matches the full JID, the recipient's server SHOULD return to the sender a <service-unavailable/> stanza error.

3. If the JID is of the form <node@domain> and there is at least one available resource available for the node, the recipient's server MUST deliver the stanza to at least one of the available resources, according to application-specific rules (a set of

delivery rules for instant messaging and presence applications is defined in XMPP IM [21]).

15. Compliance Requirements

This section summarizes the specific aspects of the Extensible Messaging and Presence Protocol that MUST be supported by servers and clients in order to be considered compliant implementations, as well as additional protocol aspects that SHOULD be supported. For compliance purposes, we draw a distinction between core protocols (which MUST be supported by any server or client, regardless of the specific application) and instant messaging protocols (which MUST be supported only by instant messaging and presence applications built on top of the core protocols). Compliance requirements that apply to all servers and clients are specified in this section; compliance requirements for instant messaging servers and clients are specified in the corresponding section of XMPP IM [21].

15.1 Servers

In addition to all defined requirements with regard to security, XML usage, and internationalization, a server MUST support the following core protocols in order to be considered compliant:

- Enforcement of the Nameprep [6], Nodeprep (Appendix A), and Resourceprep (Appendix B) profiles of stringprep
- XML streams (Section 4), including stream encryption (Section 5) using TLS, stream authentication (Section 6) using SASL, and resource binding (Section 7)
- The basic semantics of the three defined stanza kinds (i.e., <message/>, <presence/>, and <iq/>) as specified in stanza semantics (Section 9.2)
- Generation (and, where appropriate, handling) of error syntax and semantics related to streams, TLS, SASL, and XML stanzas

In addition, a server SHOULD support the following core protocol:

- Server dialback (Section 8)

15.2 Clients

A client MUST support the following core protocols in order to be considered compliant:

- XML streams (Section 4), including stream encryption (Section 5) using TLS, stream authentication (Section 6) using SASL, and resource binding (Section 7)

- The basic semantics of the three defined stanza kinds (i.e., <message/>, <presence/>, and <iq/>) as specified in stanza semantics (Section 9.2)

- Handling (and, where appropriate, generation) of error syntax and semantics related to streams, TLS, SASL, and XML stanzas

In addition, a client SHOULD support the following core protocols:

- Generation of addresses in accordance with the Nameprep [6], Nodeprep (Appendix A), and Resourceprep (Appendix B) profiles of stringprep

Normative References

[1] Bray, T., Paoli, J., Sperberg-McQueen, C. and E. Maler, "Extensible Markup Language (XML) 1.0 (2nd ed)", W3C REC-xml, October 2000, <http://www.w3.org/TR/REC-xml>.

[2] Day, M., Aggarwal, S. and J. Vincent, "Instant Messaging / Presence Protocol Requirements", RFC 2779, February 2000.

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

[4] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[5] Internet Assigned Numbers Authority, "Internet Assigned Numbers Authority", January 1998, <http://www.iana.org/>.

[6] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

[7] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[8] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", RFC 2234, November 1997.

[9] Hinden, R. and S. Deering, "IP Version 6 Addressing Architecture", RFC 2373, July 1998.

[10] Bray, T., Hollander, D. and A. Layman, "Namespaces in XML", W3C REC-xml-names, January 1999, <http://www.w3.org/TR/REC-xml-names>.

[11] Dierks, T., Allen, C., Treese, W., Karlton, P., Freier, A. and P. Kocher, "The TLS Protocol Version 1.0", RFC 2246, January 1999.

[12] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

[13] Myers, J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997.

[14] Josefsson, S., "The Base16, Base32, and Base64 Data Encodings", RFC 3548, July 2003.

[15] Alvestrand, H., "IETF Policy on Character Sets and Languages", BCP 18, RFC 2277, January 1998.

[16] Alvestrand, H., "Tags for the Identification of Languages", BCP 47, RFC 3066, January 2001.

[17] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[18] International Organization for Standardization, "Information Technology - Universal Multiple-octet coded Character Set (UCS) - Amendment 2: UCS Transformation Format 8 (UTF-8)", ISO Standard 10646-1 Addendum 2, October 1996.

[19] Linn, J., "Generic Security Service Application Program Interface, Version 2", RFC 2078, January 1997.

[20] Gulbrandsen, A., Vixie, P. and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

Informative References

[21] Saint-Andre, P. and J. Miller, "XMPP Instant Messaging", draft-ietf-xmpp-im-18 (work in progress), October 2003.

[22] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P. and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[23] Berners-Lee, T., Fielding, R. and L. Masinter, "Uniform Resource Identifiers (URI): Generic Syntax", RFC 2396, August

1998.

[24] Mealling, M., "The IETF XML Registry", draft-mealling-iana-xmlns-registry-05 (work in progress), June 2003.

[25] Crispin, M., "Internet Message Access Protocol - Version 4rev1", RFC 2060, December 1996.

[26] Myers, J. and M. Rose, "Post Office Protocol - Version 3", STD 53, RFC 1939, May 1996.

[27] Newman, C. and J. Myers, "ACAP -- Application Configuration Access Protocol", RFC 2244, November 1997.

[28] Newman, C., "Using TLS with IMAP, POP3 and ACAP", RFC 2595, June 1999.

[29] Eastlake, D., "Domain Name System Security Extensions", RFC 2535, March 1999.

[30] Jabber Software Foundation, "Jabber Software Foundation", <http://www.jabber.org/>.

Authors' Addresses

Peter Saint-Andre
Jabber Software Foundation

EMail: stpeter@jabber.org

Jeremie Miller
Jabber Software Foundation

EMail: jeremie@jabber.org

Appendix A. Nodeprep

A.1 Introduction

This appendix defines the "Nodeprep" profile of stringprep (RFC 3454 [7]). As such, it specifies processing rules that will enable users to enter internationalized node identifiers in the Extensible Messaging and Presence Protocol (XMPP) and have the highest chance of getting the content of the strings correct. (An XMPP node identifier is the optional portion of an XMPP address that precedes a domain

identifier and the '@' separator; it is often but not exclusively associated with an instant messaging username.) These processing rules are intended only for XMPP node identifiers and are not intended for arbitrary text or any other aspect of an XMPP address.

This profile defines the following, as required by RFC 3454 [7]:

- The intended applicability of the profile: internationalized node identifiers within XMPP
- The character repertoire that is the input and output to stringprep: Unicode 3.2, specified in Section 2 of this Appendix
- The mappings used: specified in Section 3
- The Unicode normalization used: specified in Section 4
- The characters that are prohibited as output: specified in Section 5
- Bidirectional character handling: specified in Section 6

A.2 Character Repertoire

This profile uses Unicode 3.2 with the list of unassigned code points being Table A.1, both defined in Appendix A of RFC 3454 [7].

A.3 Mapping

This profile specifies mapping using the following tables from RFC 3454 [7]:

Table B.1

Table B.2

A.4 Normalization

This profile specifies using Unicode normalization form KC, as described in RFC 3454 [7].

A.5 Prohibited Output

This profile specifies prohibiting use of the following tables from RFC 3454 [7].

Table C.1.1

Table C.1.2

Table C.2.1

Table C.2.2

Table C.3

Table C.4

Table C.5

Table C.6

Table C.7

Table C.8

Table C.9

In addition, the following Unicode characters are also prohibited:

#x22 (")

#x26 (&)

#x27 (')

#x2F (/)

#x3A (:)

#x3C (<)

#x3E (>)

#x40 (@)

A.6 Bidirectional Characters

This profile specifies checking bidirectional strings as described in Section 6 of RFC 3454 [7].

Appendix B. Resourceprep

B.1 Introduction

This appendix defines the "Resourceprep" profile of stringprep (RFC 3454 [7]). As such, it specifies processing rules that will enable users to enter internationalized resource identifiers in the Extensible Messaging and Presence Protocol (XMPP) and have the highest chance of getting the content of the strings correct. (An XMPP resource identifier is the optional portion of an XMPP address that follows a domain identifier and the '/' separator; it is often but not exclusively associated with an instant messaging session name.) These processing rules are intended only for XMPP resource identifiers and are not intended for arbitrary text or any other aspect of an XMPP address.

This profile defines the following, as required by RFC 3454 [7]:

- The intended applicability of the profile: internationalized resource identifiers within XMPP
- The character repertoire that is the input and output to stringprep: Unicode 3.2, specified in Section 2 of this Appendix
- The mappings used: specified in Section 3
- The Unicode normalization used: specified in Section 4
- The characters that are prohibited as output: specified in Section 5
- Bidirectional character handling: specified in Section 6

B.2 Character Repertoire

This profile uses Unicode 3.2 with the list of unassigned code points being Table A.1, both defined in Appendix A of RFC 3454 [7].

B.3 Mapping

This profile specifies mapping using the following tables from RFC 3454 [7]:

Table B.1

B.4 Normalization

This profile specifies using Unicode normalization form KC, as

described in RFC 3454 [7].

B.5 Prohibited Output

This profile specifies prohibiting use of the following tables from RFC 3454 [7].

Table C.1.2

Table C.2.1

Table C.2.2

Table C.3

Table C.4

Table C.5

Table C.6

Table C.7

Table C.8

Table C.9

B.6 Bidirectional Characters

This profile specifies checking bidirectional strings as described in Section 6 of RFC 3454 [7].

Appendix C. XML Schemas

The following XML schemas are descriptive, not normative. For schemas defining the 'jabber:client' and 'jabber:server' namespaces, refer to XMPP IM [21].

C.1 Streams namespace

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    targetNamespace='http://etherx.jabber.org/streams'
    xmlns='http://etherx.jabber.org/streams'
    elementFormDefault='unqualified'>
```

```
  <xs:import namespace='http://www.w3.org/XML/1998/namespace'
             schemaLocation='http://www.w3.org/2001/xml.xsd'/>

  <xs:element name='stream'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='features' minOccurs='0' maxOccurs='1'/>
        <xs:choice minOccurs='0' maxOccurs='1'>
          <xs:any namespace='jabber:client'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
          <xs:any namespace='jabber:server'
                  minOccurs='0'
                  maxOccurs='unbounded'/>
        </xs:choice>
        <xs:element ref='error' minOccurs='0' maxOccurs='1'/>
      </xs:sequence>
      <xs:attribute name='to' type='xs:string' use='optional'/>
      <xs:attribute name='from' type='xs:string' use='optional'/>
      <xs:attribute name='id' type='xs:NMTOKEN' use='optional'/>
      <xs:attribute ref='xml:lang' use='optional'/>
      <xs:attribute name='version' type='xs:decimal' use='optional'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='features'>
    <xs:complexType>
      <xs:sequence>
        <xs:any
            namespace='##other'
            minOccurs='0'
            maxOccurs='unbounded'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='error'>
    <xs:complexType>
      <xs:sequence>
        <xs:any namespace='urn:ietf:params:xml:ns:xmpp-streams'
                maxOccurs='2'/>
        <xs:any
            namespace='##other'
            minOccurs='0'
            maxOccurs='1'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

```
   </xs:schema>
```

C.2 Stream error namespace

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

   <xs:schema
       xmlns:xs='http://www.w3.org/2001/XMLSchema'
       xmlns:xml='http://www.w3.org/XML/1998/namespace'
       targetNamespace='urn:ietf:params:xml:ns:xmpp-streams'
       xmlns='urn:ietf:params:xml:ns:xmpp-streams'
       elementFormDefault='qualified'>

     <xs:import namespace='http://www.w3.org/XML/1998/namespace'
                schemaLocation='http://www.w3.org/2001/xml.xsd'/>

     <xs:element name='bad-format' type='empty'/>
     <xs:element name='bad-namespace-prefix' type='empty'/>
     <xs:element name='conflict' type='empty'/>
     <xs:element name='connection-timeout' type='empty'/>
     <xs:element name='host-gone' type='empty'/>
     <xs:element name='host-unknown' type='empty'/>
     <xs:element name='improper-addressing' type='empty'/>
     <xs:element name='internal-server-error' type='empty'/>
     <xs:element name='invalid-from' type='empty'/>
     <xs:element name='invalid-id' type='empty'/>
     <xs:element name='invalid-namespace' type='empty'/>
     <xs:element name='invalid-xml' type='empty'/>
     <xs:element name='not-authorized' type='empty'/>
     <xs:element name='policy-violation' type='empty'/>
     <xs:element name='remote-connection-failed' type='empty'/>
     <xs:element name='resource-constraint' type='empty'/>
     <xs:element name='restricted-xml' type='empty'/>
     <xs:element name='see-other-host' type='xs:string'/>
     <xs:element name='system-shutdown' type='empty'/>
     <xs:element name='undefined-condition' type='empty'/>
     <xs:element name='unsupported-encoding' type='empty'/>
     <xs:element name='unsupported-stanza-type' type='empty'/>
     <xs:element name='unsupported-version' type='empty'/>
     <xs:element name='xml-not-well-formed' type='empty'/>

     <xs:element name='text' type='xs:string'>
       <xs:complexType>
         <xs:attribute ref='xml:lang' use='optional'/>
       </xs:complexType>
     </xs:element>
```

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

</xs:schema>
```

C.3 TLS namespace

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

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

  <xs:element name='starttls'>
    <xs:complexType>
      <xs:sequence>
        <xs:element
            ref='required'
            minOccurs='0'
            maxOccurs='1'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:element name='required' type='empty'/>
  <xs:element name='proceed' type='empty'/>
  <xs:element name='failure' type='empty'/>

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

</xs:schema>
```

C.4 SASL namespace

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

<xs:schema
```

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

    <xs:element name='mechanisms'>
      <xs:complexType>
        <xs:sequence>
          <xs:element ref='mechanism' maxOccurs='unbounded'/>
        </xs:sequence>
      </xs:complexType>
    </xs:element>

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

    <xs:element name='auth'>
      <xs:complexType>
        <xs:attribute name='mechanism'
                      type='xs:NMTOKEN'
                      use='optional'/>
      </xs:complexType>
    </xs:element>

    <xs:element name='challenge' type='xs:NMTOKEN'/>
    <xs:element name='response' type='xs:NMTOKEN'/>
    <xs:element name='abort' type='empty'/>
    <xs:element name='success' type='empty'/>

    <xs:element name='failure'>
      <xs:complexType>
        <xs:choice maxOccurs='1'>
          <xs:element ref='aborted'/>
          <xs:element ref='incorrect-encoding'/>
          <xs:element ref='invalid-authzid'/>
          <xs:element ref='invalid-mechanism'/>
          <xs:element ref='mechanism-too-weak'/>
          <xs:element ref='not-authorized'/>
          <xs:element ref='temporary-auth-failure'/>
        </xs:choice>
      </xs:complexType>
    </xs:element>

    <xs:element name='aborted' type='empty'/>
    <xs:element name='incorrect-encoding' type='empty'/>
    <xs:element name='invalid-authzid' type='empty'/>
    <xs:element name='invalid-mechanism' type='empty'/>
    <xs:element name='mechanism-too-weak' type='empty'/>
    <xs:element name='not-authorized' type='empty'/>
```

```
    <xs:element name='temporary-auth-failure' type='empty'/>

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

  </xs:schema>
```

C.5 Resource binding namespace

```
  <?xml version='1.0' encoding='UTF-8'?>
  <xs:schema
      xmlns:xs='http://www.w3.org/2001/XMLSchema'
      targetNamespace='urn:ietf:params:xml:ns:xmpp-bind'
      xmlns='urn:ietf:params:xml:ns:xmpp-bind'
      elementFormDefault='qualified'>

    <xs:element name='bind'>
      <xs:complexType>
        <xs:choice minOccurs='0' maxOccurs='1'>
          <xs:element ref='resource'/>
          <xs:element ref='jid'/>
        </xs:choice>
      </xs:complexType>
    </xs:element>

    <xs:element name='resource' type='xs:string'/>
    <xs:element name='jid' type='xs:string'/>

  </xs:schema>
```

C.6 Dialback namespace

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

  <xs:schema
      xmlns:xs='http://www.w3.org/2001/XMLSchema'
      targetNamespace='jabber:server:dialback'
      xmlns='jabber:server:dialback'
      elementFormDefault='qualified'>

    <xs:element name='result'>
      <xs:complexType>
        <xs:simpleContent>
```

```
          <xs:extension base='xs:NMTOKEN'>
            <xs:attribute name='from' type='xs:string' use='required'/>
            <xs:attribute name='to' type='xs:string' use='required'/>
            <xs:attribute name='type' use='optional'>
              <xs:simpleType>
                <xs:restriction base='xs:NCName'>
                  <xs:enumeration value='invalid'/>
                  <xs:enumeration value='valid'/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>

    <xs:element name='verify'>
      <xs:complexType>
        <xs:simpleContent>
          <xs:extension base='xs:NMTOKEN'>
            <xs:attribute name='from' type='xs:string' use='required'/>
            <xs:attribute name='to' type='xs:string' use='required'/>
            <xs:attribute name='id' type='xs:NMTOKEN' use='required'/>
            <xs:attribute name='type' use='optional'>
              <xs:simpleType>
                <xs:restriction base='xs:NCName'>
                  <xs:enumeration value='invalid'/>
                  <xs:enumeration value='valid'/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
          </xs:extension>
        </xs:simpleContent>
      </xs:complexType>
    </xs:element>

  </xs:schema>
```

C.7 Stanza error namespace

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

  <xs:schema
      xmlns:xs='http://www.w3.org/2001/XMLSchema'
      xmlns:xml='http://www.w3.org/XML/1998/namespace'
      targetNamespace='urn:ietf:params:xml:ns:xmpp-stanzas'
      xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'
```

```
      elementFormDefault='qualified'>

    <xs:import namespace='http://www.w3.org/XML/1998/namespace'
               schemaLocation='http://www.w3.org/2001/xml.xsd'/>

    <xs:element name='bad-request' type='empty'/>
    <xs:element name='conflict' type='empty'/>
    <xs:element name='feature-not-implemented' type='empty'/>
    <xs:element name='forbidden' type='empty'/>
    <xs:element name='gone' type='xs:string'/>
    <xs:element name='internal-server-error' type='empty'/>
    <xs:element name='item-not-found' type='empty'/>
    <xs:element name='jid-malformed' type='empty'/>
    <xs:element name='not-acceptable' type='empty'/>
    <xs:element name='not-allowed' type='empty'/>
    <xs:element name='payment-required' type='empty'/>
    <xs:element name='recipient-unavailable' type='empty'/>
    <xs:element name='redirect' type='xs:string'/>
    <xs:element name='registration-required' type='empty'/>
    <xs:element name='remote-server-not-found' type='empty'/>
    <xs:element name='remote-server-timeout' type='empty'/>
    <xs:element name='resource-constraint' type='empty'/>
    <xs:element name='service-unavailable' type='empty'/>
    <xs:element name='subscription-required' type='empty'/>
    <xs:element name='undefined-condition' type='empty'/>
    <xs:element name='unexpected-request' type='empty'/>

    <xs:element name='text' type='xs:string'>
      <xs:complexType>
        <xs:attribute ref='xml:lang' use='optional'/>
      </xs:complexType>
    </xs:element>

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

  </xs:schema>
```

Appendix D. Differences Between Core Jabber Protocol and XMPP

This section is non-normative.

XMPP has been adapted from the protocols originally developed in the Jabber open-source community, which can be thought of as "XMPP 0.9".

Because there exists a large installed base of Jabber implementations and deployments, it may be helpful to specify the key differences between Jabber and XMPP in order to expedite and encourage upgrades of those implementations and deployments to XMPP. This section summarizes the core differences, while the corresponding section of XMPP IM [21] summarizes the differences that relate specifically to instant messaging and presence applications.

D.1 Channel Encryption

It is common practice in the Jabber community to use SSL for channel encryption on ports other than 5222 and 5269 (the convention is to use ports 5223 and 5270). XMPP uses TLS over the IANA-registered ports for channel encryption, as defined under Stream Encryption (Section 5) herein.

D.2 Authentication

The client-server authentication protocol developed in the Jabber community uses a basic IQ interaction qualified by the 'jabber:iq:auth' namespace (documentation of this protocol is contained in "JEP-0078: Non-SASL Authentication", published by the Jabber Software Foundation [30]). XMPP uses SASL for authentication, as defined under Stream Authentication (Section 6) herein.

The Jabber community does not currently possess an authentication protocol for server-to-server communications, only the Server Dialback (Section 8) protocol to prevent server spoofing. XMPP augments Server Dialback with a true server-to-server authentication protocol, as defined under Stream Authentication (Section 6) herein.

D.3 Resource Binding

Resource binding in the Jabber community is handled via the 'jabber:iq:auth' namespace that is also used for client authentication with a server. XMPP defines a dedicated namespace for resource binding as well as the ability for a server to generate a resource identifier on behalf of a client, as defined under Resource Binding (Section 7).

D.4 JID Processing

JID processing was somewhat loosely defined by the Jabber community (documentation of forbidden characters and case handling is contained in "JEP-0029: Definition of Jabber Identifiers", published by the Jabber Software Foundation [30]). XMPP specifies the use of Nameprep [6] for domain identifiers and supplements Nameprep with two additional stringprep [7] profiles for JID processing: Nodeprep

(Appendix A) for node identifiers and Resourceprep (Appendix B) for resource identifiers .

D.5 Error Handling

Stream-related errors are handled in the Jabber community via simple CDATA text in a <stream:error/> element. In XMPP, stream-related errors are handled via an extensible mechanism defined under Stream Errors (Section 4.6) herein.

Stanza-related errors are handled in the Jabber community via HTTP-style error codes. In XMPP, stanza-related errors are handled via an extensible mechanism defined under Stanza Errors (Section 9.3) herein. (Documentation of a mapping between Jabber and XMPP error handling mechanisms is contained in "JEP-0086: Legacy Errors", published by the Jabber Software Foundation [30].)

D.6 Internationalization

Although use of UTF-8 has always been standard practice within the Jabber community, the community did not define mechanisms for specifying the language of human-readable text provided in CDATA sections. XMPP specifies the use of the 'xml:lang' attribute in such contexts, as defined under Stream Attributes (Section 4.2) and xml:lang (Section 9.1.5) herein.

D.7 Stream Version Attribute

The Jabber community does not include a 'version' attribute in stream headers. XMPP specifies inclusion of that attribute, with a value of '1.0', as a way to signal support for the stream features (authentication, encryption, etc.) defined under Version Support (Section 4.2.1) herein.

Appendix E. Revision History

Note to RFC Editor: please remove this entire appendix, and the corresponding entries in the table of contents, prior to publication.

E.1 Changes from draft-ietf-xmpp-core-18

- Added the 'xml:lang' attribute to the root <stream/> element per previous consensus and list discussion.

- Added the <gone/>, <not-acceptable/>, and <redirect/> stanza errors.

- Changed datatype of <see-other-host/> stream error and of <gone/>

and <redirect/> stanza errors to xs:string so that these elements may contain programmatic information.

- Removed <invalid-realm/> and <bad-protocol/> SASL errors.
- Removed references to RFC 952 and RFC 1123 (domain name format is handled by reference to Nameprep).
- Changed address record resolution text so that it is not specific to IPv4.
- Clarified text in appendices regarding scope of Nodeprep and Resourceprep.
- Removed requirement that receiving entity terminate the TCP connection upon receiving an <abort/> element from or sending a <failure/> element to the initiating entity during SASL negotiation.
- Removed recommendation that TLS and SASL security layer should not both be used simultaneously.
- Added subsection to Security Considerations regarding use of base64 in SASL.
- Specified rules regarding inclusion of username in SASL negotiation.
- Adjusted content related to SASL authorization identities, since the previous text did not track RFC2222bis.
- Added section on resource binding to compensate for changes to SASL authorization identity text.
- Specified ABNF for JIDs.
- Checked all references.
- Completed a thorough proofreading and consistency check of the entire text.

E.2 Changes from draft-ietf-xmpp-core-17

- Specified that UTF-8 is the only allowable encoding.
- Added stream errors for <bad-namespace-prefix/>, <invalid-xml/>, and <restricted-xml/>, as well as a <bad-format/> error for

generic XML error conditions.

- Folded Nodeprep and Resourceprep profiles into this document.
- Moved most delivery handling rules from XMPP IM to XMPP Core.
- Moved detailed stanza syntax descriptions from XMPP Core to XMPP IM.
- Moved stanza schemas from XMPP Core to XMPP IM.

E.3 Changes from draft-ietf-xmpp-core-16

- Added <conflict/> and <unsupported-encoding/> stream errors.
- Changed the datatype for the <see-other-host/> and <unsupported-version/> stream errors from 'xs:string' to 'empty'.
- Further clarified server handling of the basic stanza kinds.
- Further clarified character encoding rules per list discussion.
- Specified meaning of version='1.0' flag in stream headers.
- Added stream closure to SASL failure cases in order to mirror handling of TLS failures.
- Added section on compliance requirements for server and client implementations.
- Added non-normative section on differences between Jabber usage and XMPP specifications.

E.4 Changes from draft-ietf-xmpp-core-15

- Added <connection-timeout/> and <policy-violation/> stream errors.
- Added <aborted/> SASL error and clarified <bad-protocol/> error.
- Made 'id' required for IQ stanzas.

E.5 Changes from draft-ietf-xmpp-core-14

- Added SRV lookup for client-to-server communications.

- Changed server SRV record to conform to RFC 2782; specifically, the service identifier was changed from 'jabber' to 'jabber-server'.

E.6 Changes from draft-ietf-xmpp-core-13

- Clarified stream restart after successful TLS and SASL negotiation.
- Clarified requirement for resolution of DNS hostnames.
- Clarified text regarding namespaces.
- Clarified examples regarding empty <stream:features/> element.
- Added several more SASL error conditions.
- Changed <invalid-xml/> stream error to <improper-addressing/> and added to schema.
- Made small editorial changes and fixed several schema errors.

E.7 Changes from draft-ietf-xmpp-core-12

- Moved server dialback to a separate section; clarified its security characteristics and its role in the protocol.
- Adjusted error handling syntax and semantics per list discussion.
- Further clarified length of node identifiers and total length of JIDs.
- Documented message type='normal'.
- Corrected several small errors in the TLS and SASL sections.
- Corrected several errors in the schemas.

E.8 Changes from draft-ietf-xmpp-core-11

- Corrected several small errors in the TLS and SASL sections.
- Made small editorial changes and fixed several schema errors.

E.9 Changes from draft-ietf-xmpp-core-10

- Adjusted TLS content regarding certificate validation process.
- Specified that stanza error extensions for specific applications are to be properly namespaced children of the relevant descriptive element.
- Clarified rules for inclusion of the 'id' attribute.
- Specified that the 'xml:lang' attribute SHOULD be included (per list discussion).
- Made small editorial changes and fixed several schema errors.

E.10 Changes from draft-ietf-xmpp-core-09

- Fixed several dialback error conditions.
- Cleaned up rules regarding TLS and certificate processing based on off-list feedback.
- Changed <stream-condition/> and <stanza-condition/> elements to <condition/>.
- Added or modified several stream and stanza error conditions.
- Specified only one child allowed for IQ, or two if type="error".
- Fixed several errors in the schemas.

E.11 Changes from draft-ietf-xmpp-core-08

- Incorporated list discussion regarding addressing, SASL, TLS, TCP, dialback, namespaces, extensibility, and the meaning of 'ignore' for routers and recipients.
- Specified dialback error conditions.
- Made small editorial changes to address RFC Editor requirements.

E.12 Changes from draft-ietf-xmpp-core-07

- Made several small editorial changes.

E.13 Changes from draft-ietf-xmpp-core-06

- Added text regarding certificate validation in TLS negotiation per list discussion.
- Clarified nature of XML restrictions per discussion with W3C, and moved XML Restrictions subsection under "XML Usage within XMPP".
- Further clarified that XML streams are unidirectional.
- Changed stream error and stanza error namespace names to conform to the format defined in The IETF XML Registry [24].
- Removed note to RFC Editor regarding provisional namespace names.

E.14 Changes from draft-ietf-xmpp-core-05

- Added <invalid-namespace/> as a stream error condition.
- Adjusted security considerations per discussion at IETF 56 and on list.

E.15 Changes from draft-ietf-xmpp-core-04

- Added server-to-server examples for TLS and SASL.
- Changed error syntax, rules, and examples based on list discussion.
- Added schemas for the TLS, stream error, and stanza error namespaces.
- Added note to RFC Editor regarding provisional namespace names.
- Made numerous small editorial changes and clarified text throughout.

E.16 Changes from draft-ietf-xmpp-core-03

- Clarified rules and procedures for TLS and SASL.
- Amplified stream error code syntax per list discussion.
- Made numerous small editorial changes.

E.17 Changes from draft-ietf-xmpp-core-02

- Added dialback schema.
- Removed all DTDs since schemas provide more complete definitions.
- Added stream error codes.
- Clarified error code "philosophy".

E.18 Changes from draft-ietf-xmpp-core-01

- Updated the addressing restrictions per list discussion and added references to the new Nodeprep and Resourceprep profiles.
- Corrected error in Stream Authentication regarding 'version' attribute.
- Made numerous small editorial changes.

E.19 Changes from draft-ietf-xmpp-core-00

- Added information about TLS from list discussion.
- Clarified meaning of "ignore" based on list discussion.
- Clarified information about Universal Character Set data and character encodings.
- Provided base64-decoded information for examples.
- Fixed several errors in the schemas.
- Made numerous small editorial fixes.

E.20 Changes from draft-miller-xmpp-core-02

- Brought Stream Authentication section into line with discussion on list and at IETF 55 meeting.
- Added information about the optional 'xml:lang' attribute per discussion on list and at IETF 55 meeting.
- Specified that validation is neither required nor recommended, and that the formal definitions (DTDs and schemas) are included for

descriptive purposes only.

- Specified that the response to an IQ stanza of type "get" or "set" must be an IQ stanza of type "result" or "error".

- Specified that compliant server implementations must process stanzas in order.

- Specified that for historical reasons some server implementations may accept 'stream:' as the only valid namespace prefix on the root stream element.

- Clarified the difference between 'jabber:client' and 'jabber:server' namespaces, namely, that 'to' and 'from' attributes are required on all stanzas in the latter but not the former.

- Fixed typo in Step 9 of the dialback protocol (changed db:result to db:verify).

- Removed references to TLS pending list discussion.

- Removed the non-normative appendix on OpenPGP usage pending its inclusion in a separate I-D.

- Simplified the architecture diagram, removed most references to services, and removed references to the 'jabber:component:*' namespaces.

- Noted that XMPP activity respects firewall administration policies.

- Further specified the scope and uniqueness of the 'id' attribute in all stanza kinds and the <thread/> element in message stanzas.

- Nomenclature changes: (1) from "chunks" to "stanzas"; (2) from "host" to "server" and from "node" to "client" (except with regard to definition of the addressing scheme).

Intellectual Property Statement

Full Copyright Statement

Acknowledgment

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