SRV record query extension for XMPP
draft-harada-xmpp-srv-record-query-00

Abstract

According to RFC 6120, XMPP clients should use SRV records within the connection establishment process to servers. There are two purposes for using SRV records. The one is to advertise the FQDN of at least one server to clients to establish an initial TCP connection. The other purpose is to advertise at least two or more FQDN with priority and weight information to clients to accomplish load balancing, a redundant server environment and so on. However, most standard resolver libraries of recent client OSs don’t support SRV record resolution. Furthermore, many DNS hosting services also don’t support SRV records. This document proposes a solution that enables a server and client to achieve load balancing and a redundant server environment in case SRV records cannot be used. Moreover, the proposed IQ-result message can include minimum wait time to reconnect, allowing clients to reconnect after the most suitable wait time avoiding congestion.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."
The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html.

This Internet-Draft will expire on July 31, 2012.

Copyright Notice

Copyright (c) 2010 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document.

Table of Contents

1. Background .......................................................... 3
2. Problems .............................................................. 3
   2.1 Client Resolver ................................................... 3
   2.2 DNS Hosting Service ............................................. 3
3. Proposed Solution .................................................... 4
   3.1 Overview of the solution ....................................... 4
   3.2 IQ-get message ................................................... 4
   3.3 IQ-result message ............................................... 4
   3.4 IQ-error message ............................................... 5
4. Reconnection Policy Information ....................................... 5
5. Security Considerations ............................................... 6
6. IANA Considerations ................................................... 6
7. References ............................................................. 6
   7.1 Normative References ............................................. 6
   7.2 Informative References ........................................... 7
1. Background

In section 3 "TCP Binding", RFC 6120 specifies that XMPP clients should use DNS SRV records to connect to servers. There are two purposes for using SRV records within the connection establishment process. They are as follows:

(A.1) To inform initiating entities of at least one server hostname for TCP connection establishment.

(A.2) To inform initiating entities of at least two or more server hostnames with priority and weight information to achieve load-balancing or a redundant server environment.

2. Problems

However, at present, there are many cases of incompletely supported SRV records in both clients and servers.

2.1 Client Resolver

Many of the deployed DNS resolver implementations which are implemented as a standard library on popular client OSs support 'A record' and 'AAAA record' resolution but don’t support 'SRV record' resolution. If a standard library doesn’t support SRV record resolution, client software has to have the original code of resolver functions. However, implementing original resolver functions can result in serious implementation costs.

There are some solutions relating to (A.1), which are A record resolutions or 'hardcode' of the server hostname.

However, there are no solutions relating to (A.2) described in the above section.

2.2 DNS Hosting Service

There are many DNS Hosting Services and many of them don’t support SRV record registration via a web-based user interface. In this case, to accomplish (A.2), internal redundancy technologies like internal clustering technologies will be required. However, establishment of such internal redundancy environments can result in serious operation costs.
3. Proposed Solution

3.1 Overview of the solution

This section describes a new info/query stanza as a solution for the above problem. This is done by sending an <iq/> get over the stream from the client to the server.

3.2 IQ-get message

Example 1. IQ-get message (client - server)

<iq from="alice@wonderland.lit"
     id="ac4cf"
     to="wonderland"
     type="get">
     <query xmlns="jabber:iq:SRVrecord" />
</iq>

The client requests the server for information similar to a SRV record. If the server supports similar SRV record information, it MUST return an IQ-result with that information to the client.

3.3 IQ-result message

Example 2. IQ-result message (server - client)

<iq from="wonderland.lit"
     id="ac4cf"
     to="alice@wonderland.lit"
     type="result">
     <query xmlns="jabber:iq:SRVrecord">
       <item name="sv1.wonderland.lit" priority='10' weight='10'/>
       <item name="sv2.wonderland.lit" priority='10' weight='10'/>
       <item name="sv3.wonderland.lit" priority='20' weight='0'/>
     </query>
</iq>

If the server does not support any SRV record information, the server returns an error message to the client.
3.4 IQ-error message

Example 3. IQ-error message (server – client)

<iq from="wonderland.lit"
  id="ac4cf"
  to="alice@wonderland.lit"
  type="error">
  <body>Sorry! Not support to SRV record query!</body>
  <error code='404' type='cancel'>
    <item-not-found xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>

Other error conditions defined in RFC 6120 could also be returned if appropriate.

4. Reconnection Policy Information

The reconnection algorithm when an XMPP server goes offline unexpectedly is specified as follows in section 3.3 of RFC 6120.

"It can happen that an XMPP server goes offline unexpectedly while servicing TCP connections from connected clients and remote servers. Because the number of such connections can be quite large, the reconnection algorithm employed by entities that seek to reconnect can have a significant impact on software performance and network congestion. If an entity chooses to reconnect, it:

- SHOULD set the number of seconds that expire before reconnecting to an unpredictable number between 0 and 60 (this helps to ensure that not all entities attempt to reconnect at exactly the same number of seconds after being disconnected).

- SHOULD back off increasingly on the time between subsequent reconnection attempts (e.g., in accordance with "truncated binary exponential backoff" as described in [ETHERNET]) if the first reconnection attempt does not succeed."

The purpose of this algorithm is to avoid congestion by a large number of clients reconnect to the XMPP server simultaneously. However, congestion can be avoided without the above algorithms if the extension proposed in this document is used. This can be accomplished by informing a different 'minimum reconnection wait time' to each client. Furthermore, by informing a short 'minimum reconnection wait time' to lively clients and informing a long 'minimum reconnection wait time' to less lively clients, efficiency can be improved. The following example shows a case where the server allows client A to reconnect after 5 seconds and allows client B to reconnect after 30 seconds.
Example 4. IQ-result message (server - client)

<iq from="wonderland.lit"
    id="ac4cf"
    to="alice@wonderland.lit"
    type="result">
    <query xmlns="jabber:iq:SRVrecord">
        <item name="sv1.wonderland.lit" priority='10' weight='10' wait='5'/>
        <item name="sv2.wonderland.lit" priority='10' weight='10' wait='5'/>
        <item name="sv3.wonderland.lit" priority='20' weight='0' wait='5'/>
    </query>
</iq>

Example 5. IQ-result message (server - client)

<iq from="wonderland.lit"
    id="ac4cf"
    to="alice@wonderland.lit"
    type="result">
    <query xmlns="jabber:iq:SRVrecord">
        <item name="sv1.wonderland.lit" priority='10' weight='10' wait='30'/>
        <item name="sv2.wonderland.lit" priority='10' weight='10' wait='30'/>
        <item name="sv3.wonderland.lit" priority='20' weight='0' wait='30'/>
    </query>
</iq>

5. Security Considerations

This document has no requirement for a change to the security models within associated protocols.

6. IANA Considerations

TBD

7. References

7.1. Normative References


[RFC2782] A. Gulbrandsen, P. Vixie, L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", February 2000
7.2. Informative References

Authors' Addresses

Akari Harada
Maebashi Kyoai Gakuen College
1154-4 Koyahara-machi
Maebashi City, Gunma Pref
JAPAN 379-2192
EMail: harada09@c.kyoai.ac.jp
URI: http://www.kyoai.ac.jp/

Nobuo Ogashiwa
Maebashi Kyoai Gakuen College
1154-4 Koyahara-machi
Maebashi City, Gunma Pref
JAPAN 379-2192
EMail: ogashiwa@c.kyoai.ac.jp
URI: http://www.kyoai.ac.jp/

Hirotaka Sato
Keio Univ.
5322 Endo, Kanagawa,
Japan 252-0812
Email: satu@sfc.wide.ad.jp

Yoshihiro Suzuki
D3 Communications
1-18-31 Tamagawa-Gakuen
Machida City, Tokyo
Japan 194-0041
Email: hiro.suzuki@d3communications.jp
Abstract

This document defines the address format for the Extensible Messaging and Presence Protocol (XMPP), including support for code points outside the US-ASCII range. This document obsoletes RFC 6122.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 20, 2012.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
Table of Contents

1. Introduction .................................................. 3
   1.1. Overview .................................................. 3
   1.2. Terminology ................................................ 3
2. Addresses ...................................................... 3
   2.1. Fundamentals .............................................. 3
   2.2. Domainpart ................................................ 5
   2.3. Localpart .................................................. 6
   2.4. Resourcepart .............................................. 7
3. Internationalization Considerations .......................... 8
4. Security Considerations ........................................ 8
   4.1. Reuse of PRECIS ........................................... 8
   4.2. Reuse of Unicode ......................................... 9
   4.3. Address Spoofing ......................................... 9
       4.3.1. Address Forging .................................... 9
       4.3.2. Address Mimicking .................................. 10
5. IANA Considerations ........................................... 11
   5.1. Use of NameClass ......................................... 11
   5.2. Use of FreeClass ......................................... 11
6. Conformance Requirements .................................... 12
7. References .................................................... 13
   7.1. Normative References .................................... 13
   7.2. Informative References .................................. 14
Appendix A. Differences from RFC 6122 .......................... 16
Appendix B. Acknowledgements .................................... 17
Author’s Address .................................................. 17
1. Introduction

1.1. Overview

The Extensible Messaging and Presence Protocol [XMPP] is an application profile of the Extensible Markup Language [XML] for streaming XML data in close to real time between any two or more network-aware entities. The address format for XMPP entities was originally developed in the Jabber open-source community in 1999, first described by [XEP-0029] in 2002, and then defined canonically by [RFC3920] in 2004 and [RFC6122] in 2011.

As specified in RFC 3920 and RFC 6122, the XMPP address format used the "stringprep" technology for preparation of non-ASCII characters [STRINGPREP]. This document defines the XMPP address format in a way that no longer depends on stringprep. Instead, this document depends on the internationalization framework defined by the IETF’s PRECIS Working Group [FRAMEWORK].

This document obsoletes RFC 6122.

1.2. Terminology

Many important terms used in this document are defined in [FRAMEWORK], [I18N-TERMS], [IDNA-DEFS], [UNICODE], and [XMPP].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [KEYWORDS].

2. Addresses

2.1. Fundamentals

An XMPP entity is anything that is network-addressable and that can communicate using XMPP. For historical reasons, the native address of an XMPP entity is called a Jabber Identifier ("JID"). A valid JID is a string of [UNICODE] code points, encoded using [UTF-8], and structured as an ordered sequence of localpart, domainpart, and resourcepart (where the first two parts are demarcated by the '/' character used as a separator, and the last two parts are similarly demarcated by the '/' character).

The syntax for a JID is defined as follows using the Augmented Backus-Naur Form as specified in [ABNF].
jid = [ localpart "@" ] domainpart [ "/" resourcepart ]
localpart = 1*(localpoint)
; a "localpoint" is a UTF-8 encoded Unicode code point that conforms to the localpart subclass of the "NameClass" string class defined in draft-blanchet-precis-framework

domainpart = IP-literal / IPv4address / ifqdn
; the "IPv4address" and "IP-literal" rules are defined in RFC 3986, and the first-match-wins (a.k.a. "greedy") algorithm described in RFC 3986 applies to the matching process
; note well that reuse of the IP-literal rule from RFC 3986 implies that IPv6 addresses are enclosed in square brackets (i.e., beginning with '[' and ending with ']')

ifqdn = 1*(domainpoint)
; a "domainpoint" is a UTF-8 encoded Unicode code point that conforms to the "domain name" string class effectively defined in RFC 5890

resourcepart = 1*(resourcepoint)
; a "resourcepoint" is a UTF-8 encoded Unicode code point that conforms to the localpart subclass of the "FreeClass" string class defined in draft-blanchet-precis-framework

All JIDs are based on the foregoing structure.

Each allowable portion of a JID (localpart, domainpart, and resourcepart) MUST NOT be zero bytes in length and MUST NOT be more than 1023 bytes in length, resulting in a maximum total size (including the '@' and '/' separators) of 3071 bytes.

For the purposes of communication over an XMPP network (e.g., in the 'to' or 'from' address of an XMPP stanza), an entity's address MUST be represented as a JID, not as a Uniform Resource Identifier [URI] or Internationalized Resource Identifier [IRI]. An XMPP URI or IRI [XMPP-URI] is in essence a JID prepended with 'xmpp:'; however, the native addressing format used in XMPP is that of a mere JID without a URI scheme. [XMPP-URI] is provided only for identification and
interaction outside the context of XMPP itself, for example when linking to a JID from a web page. See [XMPP-URI] for information about securely extracting a JID from an XMPP URI or IRI.

Implementation Note: When dividing a JID into its component parts, an implementation needs to match the separator characters '@' and '/' before applying any transformation algorithms, which might decompose certain Unicode code points to the separator characters (e.g., U+FE6B SMALL COMMERCIAL AT might decompose to U+0040 COMMERCIAL AT).

2.2. Domainpart

The domainpart of a JID is that portion after the '@' character (if any) and before the '/' character (if any); it is the primary identifier and is the only REQUIRED element of a JID (a mere domainpart is a valid JID). Typically a domainpart identifies the "home" server to which clients connect for XML routing and data management functionality. However, it is not necessary for an XMPP domainpart to identify an entity that provides core XMPP server functionality (e.g., a domainpart can identify an entity such as a multi-user chat service [XEP-0045], a publish-subscribe service [XEP-0060], or a user directory).

The domainpart for every XMPP service MUST be a fully qualified domain name (FQDN; see [DNS]), IPv4 address, IPv6 address, or unqualified hostname (i.e., a text label that is resolvable on a local network).

Interoperability Note: Domainparts that are IP addresses might not be accepted by other services for the sake of server-to-server communication, and domainparts that are unqualified hostnames cannot be used on public networks because they are resolvable only on a local network.

If the domainpart includes a final character considered to be a label separator (dot) by [DNS], this character MUST be stripped from the domainpart before the JID of which it is a part is used for the purpose of routing an XML stanza, comparing against another JID, or constructing an [XMPP-URI]. In particular, the character MUST be stripped before any other canonicalization steps are taken.

A domainpart MUST NOT be zero bytes in length and MUST NOT be more than 1023 bytes in length. This rule is to be enforced after any mapping or normalization of code points. Naturally, the length limits of [DNS] apply, and nothing in this document is to be interpreted as overriding those more fundamental limits.
In the terms of IDNA2008 [IDNA-DEFS], the domainpart of a JID is a "domain name slot".

A domainpart consisting of a fully qualified domain name MUST be an "internationalized domain name" as defined in [IDNA-DEFS] and MUST consist only of Unicode code points that conform to the rules specified in [IDNA-CODE].

For the purposes of communication over XMPP, the domainpart of a JID MUST be treated as follows, where the operations specified MUST be completed in the order shown:

1. Uppercase and titlecase characters MUST be mapped to their lowercase equivalents.

2. All characters MUST be mapped using Unicode Normalization Form C (NFC). [[OPEN ISSUE: Use NFD instead?]]

3. Each A-label SHOULD be converted to a U-label (however, if it is not converted then the application MUST apply the Punycode algorithm [PUNYCODE] to each A-label and prepend the ACE prefix ("xn--") to the resulting DNS domain name).

With regard to directionality, the "Bidi Rule" provided in [IDNA-BIDI] applies.

2.3. Localpart

The localpart of a JID is an optional identifier placed before the domainpart and separated from the latter by the '@' character. Typically a localpart uniquely identifies the entity requesting and using network access provided by a server (i.e., a local account), although it can also represent other kinds of entities (e.g., a chat room associated with a multi-user chat service [XEP-0045]). The entity represented by an XMPP localpart is addressed within the context of a specific domain (i.e., <localpart@domainpart>).

A localpart MUST NOT be zero bytes in length and MUST NOT be more than 1023 bytes in length. This rule is to be enforced after any mapping or normalization of code points.

A localpart MUST consist only of Unicode code points that conform to the "NameClass" base string class defined in [FRAMEWORK], with the exception of the following characters that are explicitly disallowed in XMPP localparts:
For the purposes of communication over XMPP, the localpart of a JID MUST be treated as follows, where the operations specified MUST be completed in the order shown:

1. Uppercase and titlecase characters MUST be mapped to their lowercase equivalents.

2. All characters MUST be mapped using Unicode Normalization Form C (NFC). [[OPEN ISSUE: Use NFD instead?]]

With regard to directionality, the "Bidi Rule" provided in [IDNA-BIDI] applies.

2.4. Resourcepart

The resourcepart of a JID is an optional identifier placed after the domainpart and separated from the latter by the '/' character. A resourcepart can modify either a <localpart@domainpart> address or a mere <domainpart> address. Typically a resourcepart uniquely identifies a specific connection (e.g., a device or location) or object (e.g., an occupant in a multi-user chat room [XEP-0045]) belonging to the entity associated with an XMPP localpart at a domain (i.e., <localpart@domainpart/resourcepart>).

A resourcepart MUST NOT be zero bytes in length and MUST NOT be more than 1023 bytes in length. This rule is to be enforced after any mapping or normalization of code points.

A resourcepart MUST consist only of Unicode code points that conform to the "FreeClass" base string class defined in [FRAMEWORK].

For the purposes of communication over XMPP, the localpart of a JID MUST be treated as follows, where the operations specified MUST be completed in the order shown:

1. Uppercase and titlecase characters MAY be mapped to their lowercase equivalents.
2. All characters MUST be mapped using Unicode Normalization Form C (NFC). [OPEN ISSUE: Use NFD instead?]

With regard to directionality, the "Bidi Rule" provided in [IDNA-BIDI] applies.

XMPP entities SHOULD consider resourceparts to be opaque strings and SHOULD NOT impute meaning to any given resourcepart. In particular:

- Use of the '/' character as a separator between the domainpart and the resourcepart does not imply that XMPP addresses are hierarchical in the way that, say, HTTP addresses are hierarchical; thus for example an XMPP address of the form
  <localpart@domainpart/foo/bar> does not identify a resource "bar" that exists below a resource "foo" in a hierarchy of resources associated with the entity "localpart@domainpart".

- The '@' character is allowed in the resourcepart and is often used in the "nick" shown in XMPP chatrooms [XEP-0045]. For example, the JID <room@chat.example.com/user@host> describes an entity who is an occupant of the room <room@chat.example.com> with an (asserted) nick of <user@host>. However, chatroom services do not necessarily check such an asserted nick against the occupant’s real JID.

3. Internationalization Considerations

XMPP applications MUST support IDNA2008 for domainparts, the "NameClass" string class from [FRAMEWORK] for localparts (with the exception of certain ASCII characters specified under Section 2.3), and the "FreeClass" string class from [FRAMEWORK] for resourceparts. This enables XMPP addresses to include a wide variety of characters outside the US-ASCII range. Rules for enforcement of the XMPP address format are provided in [XMPP] and specifications for various XMPP extensions.


4. Security Considerations

4.1. Reuse of PRECIS

The security considerations described in [FRAMEWORK] apply to the "NameClass" and "FreeClass" base string classes used in this document.
for XMPP localparts and resourceparts. The security considerations described in [IDNA-DEFS] apply to internationalized domain names, which are used here for XMPP domainparts.

4.2. Reuse of Unicode

The security considerations described in [UTR39] apply to the use of Unicode characters in XMPP addresses.

4.3. Address Spoofing

There are two forms of address spoofing: forging and mimicking.

4.3.1. Address Forging

In the context of XMPP technologies, address forging occurs when an entity is able to generate an XML stanza whose ‘from’ address does not correspond to the account credentials with which the entity authenticated onto the network (or an authorization identity provided during negotiation of SASL authentication [SASL] as described in [XMPP]). For example, address forging occurs if an entity that authenticated as "juliet@im.example.com" is able to send XML stanzas from "nurse@im.example.com" or "romeo@example.net".

Address forging is difficult in XMPP systems, given the requirement for sending servers to stamp ‘from’ addresses and for receiving servers to verify sending domains via server-to-server authentication (see [XMPP]). However, address forging is possible if:

- A poorly implemented server ignores the requirement for stamping the ‘from’ address. This would enable any entity that authenticated with the server to send stanzas from any localpart@domainpart as long as the domainpart matches the sending domain of the server.

- An actively malicious server generates stanzas on behalf of any registered account at the domain or domains hosted at that server.

Therefore, an entity outside the security perimeter of a particular server cannot reliably distinguish between JIDs of the form <localpart@domainpart> at that server and thus can authenticate only the domainpart of such JIDs with any level of assurance. This specification does not define methods for discovering or counteracting the kind of poorly implemented or rogue servers just described. However, the end-to-end authentication or signing of XMPP stanzas could help to mitigate this risk, since it would require the rogue server to generate false credentials for signing or encryption of each stanza, in addition to modifying ‘from’ addresses.
Furthermore, it is possible for an attacker to forge JIDs at other domains by means of a DNS poisoning attack if DNS security extensions [DNSSEC] are not used.

4.3.2. Address Mimicking

Address mimicking occurs when an entity provides legitimate authentication credentials for and sends XML stanzas from an account whose JID appears to a human user to be the same as another JID. Because many characters are visually similar, it is relatively easy to mimic JIDs in XMPP systems. As one simple example, the localpart "ju1iet" (using the Arabic numeral one as the third character) might appear the same as the localpart "juliet" (using lowercase "L" as the third character).

As explained in [IDNA-DEFS], [FRAMEWORK], [UTR36], and [UTR39], there is no straightforward solution to the problem of visually similar characters. Furthermore, IDNA and PRECIS technologies do not attempt to define such a solution. As a result, XMPP domainparts, localparts, and resourceparts could contain such characters, leading to security vulnerabilities such as the following:

- A domainpart is always employed as one part of an entity’s address in XMPP. One common usage is as the address of a server or server-side service, such as a multi-user chat service [XEP-0045]. The security of such services could be compromised based on different interpretations of the internationalized domainpart; for example, a user might authorize a malicious entity at a fake server to view the user’s presence information, or a user could join chatrooms at a fake multi-user chat service.

- A localpart 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 localparts as part of their addresses. The security of such services could be compromised based on different interpretations of the internationalized localpart; for example, a user entering a single internationalized localpart could access another user’s account information, or a user could gain access to a hidden or otherwise restricted chat room or service.

- A resourcepart 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 connected resource; another is as the nickname of a user in a multi-user chat room; and many other kinds of entities could use resourceparts as part of their addresses. The security of such services could be compromised based on different interpretations
of the internationalized resourcepart; for example, two or more confusable resources could be bound at the same time to the same account (resulting in inconsistent authorization decisions in an XMPP application that uses full JIDs), or a user could send a message to someone other than the intended recipient in a multi-user chat room.

XMPP services and clients are strongly encouraged to define and implement consistent policies regarding the registration, storage, and presentation of visually similar characters in XMPP systems. In particular, service providers and software implementers are strongly encouraged to use the policies recommended in [FRAMEWORK].

5. IANA Considerations

5.1. Use of NameClass

The IANA shall add an entry to the PRECIS Usage Registry for reuse of the PRECIS NameClass in XMPP, as follows:

Application Protocol: XMPP.
Base Class: NameClass.
Subclassing: Yes. See Section 2.3 of RFC XXXX.
Directionality: If the string contains at least one right-to-left code point, the entire string is considered to be right-to-left.
Casemapping: Uppercase and titlecase code points are mapped to their lowercase equivalents.
Normalization: NFC.
Specification: RFC XXXX.

5.2. Use of FreeClass

The IANA shall add an entry to the PRECIS Usage Registry for reuse of the PRECIS FreeClass in XMPP, as follows:

Application Protocol: XMPP.
Base Class: FreeClass
Subclassing: No.
Directionality: If the string contains at least one right-to-left code point, the entire string is considered to be right-to-left.
Casemapping: None.
Normalization: NFC.
Specification: RFC XXXX.
6. Conformance Requirements

This section describes a protocol feature set that summarizes the conformance requirements of this specification. This feature set is appropriate for use in software certification, interoperability testing, and implementation reports. For each feature, this section provides the following information:

- A human-readable name
- An informational description
- A reference to the particular section of this document that normatively defines the feature
- Whether the feature applies to the Client role, the Server role, or both (where "N/A" signifies that the feature is not applicable to the specified role)
- Whether the feature MUST or SHOULD be implemented, where the capitalized terms are to be understood as described in [KEYWORDS]

The feature set specified here attempts to adhere to the concepts and formats proposed by Larry Masinter within the IETF’s NEWTRK Working Group in 2005, as captured in [INTEROP]. Although this feature set is more detailed than called for by [REPORTS], it provides a suitable basis for the generation of implementation reports to be submitted in support of advancing this specification from Proposed Standard to Draft Standard in accordance with [PROCESS].

Feature: address-domain-length
Description: Ensure that the domainpart of an XMPP address is at least one byte in length and at most 1023 bytes in length, and conforms to the underlying length limits of the DNS.
Section: Section 2.2
Roles: Both MUST.

Feature: address-domain-prep
Description: Ensure that the domainpart of an XMPP address conforms to IDNA2008, mapped to lowercase and normalized using NFC.
Section: Section 2.2
Roles: Both MUST.

Feature: address-localpart-length
Description: Ensure that the localpart of an XMPP address is at least one byte in length and at most 1023 bytes in length.
Section: Section 2.3
Roles: Both MUST.
Feature: address-localpart-prep
Description: Ensure that the localpart of an XMPP address conforms to the "NameClass" base string class from the PRECIS framework, excluding the eight XMPP prohibited code points (U+0022, U+0026, U+0027, U+002F, U+003A, U+003C, U+003E, and U+0040), with all code points mapped to lowercase and normalized using NFC.
Section: Section 2.3
Roles: Both MUST.

Feature: address-resource-length
Description: Ensure that the resourcepart of an XMPP address is at least one byte in length and at most 1023 bytes in length.
Section: Section 2.4
Roles: Both MUST.

Feature: address-resource-prep
Description: Ensure that the resourcepart of an XMPP address conforms to the "FreeClass" base string class from the PRECIS framework, with all code points normalized using NFC.
Section: Section 2.4
Roles: Both MUST.

7. References

7.1. Normative References


7.2. Informative References


[IDNA2003] Faltstrom, P., Hoffman, P., and A. Costello,
"Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

See Section 1 for an explanation of why the normative reference to an obsoleted specification is needed.

[IDNA-RATIONALE]


Appendix A. Differences from RFC 6122

Based on consensus derived from implementation and deployment experience as well as formal interoperability testing, the following substantive modifications were made from RFC 3920.

- Changed localpart preparation to use PRECIS instead of the Nodeprep profile of Stringprep.
- Changed resourcepart preparation to use PRECIS instead of the Resourceprep profile of Stringprep.
Appendix B. Acknowledgements

Some text in this document was borrowed or adapted from [IDNA-DEFS], [IDNA-PROTO], [IDNA-RATIONALE], and [XEP-0165].

Author’s Address

Peter Saint-Andre
Cisco
1899 Wynkoop Street, Suite 600
Denver, CO  80202
USA

Phone: +1-303-308-3282
Email: psaintan@cisco.com
Domain Name Assertions
draft-ietf-xmpp-dna-01

Abstract

The current authentication process in XMPP requires the XMPP server for a domain to present a certificate that contains that domain's name. This requirement causes several problems in scenarios where XMPP services have been delegated from one domain to another, especially when one domain provides XMPP services for many domains. This document describes an extension to the XMPP authentication process that allows domains to be securely delegated, simplifying authorization in delegation scenarios.

Requirements Language

The 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 [RFC2119].

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 15, 2011.

Copyright Notice

Copyright (c) 2011 IETF Trust and the persons identified as the document authors. All rights reserved.

Barnes & Lindberg       Expires September 15, 2011       [Page 1]
This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 3
2. Terminology .................................................. 3
3. Protocol Overview ............................................. 4
4. Connection Model .............................................. 8
5. Channel Establishment and Authentication .................... 9
6. Authorizing XMPP Stanzas ................................... 13
7. Backward Compatibility ..................................... 15
8. Operational Considerations ................................ 16
9. IANA Considerations .......................................... 16
10. Security Considerations .................................... 16
11. Acknowledgements ........................................... 16
12. Normative References ....................................... 17
Authors’ Addresses ............................................. 17
1. Introduction

When connecting two XMPP services to provide inter-domain communication, it is important for a service to be able to determine the identity of a peer service to prevent traffic spoofing. The Jabber communities first approach to identity verification was the Server Dialback protocol. When the Jabber protocols were formalized by the XMPP working group of the IETF 2002-04, support for strong identity verification using TLS + SASL was added.

Server Dialback [XEP-0220] provides weak identity verification and makes it more difficult to spoof hostnames of servers XMPP network. However, it does not provide authentication between servers and is not a security mechanism. It is susceptible to DNS poisoning attacks (unless DNSSEC is used) and cannot protect against attackers capable of hijacking the IP address of a remote service.

TLS + SASL provides strong identity verification but requires a obtaining a digital certificate by a trusted CA (or the XMPP Intermediate Certification Authority) and using it in the XMPP service, which may be hosted by a 3rd party. This solution does not allow for multiplexing traffic for multiple domain pairs over a connection, possibly requiring a large number of connections between two hosting providers.

Server Dialback can be used with TLS. When STARTTLS negotiation succeeds with a peer service but the peer’s certificate cannot be used to establish the peer’s identity, the remote domain may use on Server Dialback for (weak) identity verification. One use case can be an originating server that wish to use TLS for encryption, but only can present a self signed certificate.

In practice, many XMPP server deployments rely on Server Dialback and either do not support XMPP 1.0 or do not offer negotiation of TLS + SASL.

This goal of this document is to describe secure authentication using a hosting provide TLS certificate from a trusted CA, combined with a dialback mechanism providing secure delegation based on DNS record delgation verified using DNSSEC.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].
We will refer to four different types of domains in this document:

- **Sender domain**: The domain that initially sends out an XMPP message
- **Target domain**: The ultimate destination of an XMPP message
- **Originating domain**: The originating domain of a particular server-to-server connection
- **Receiving domain**: The receiving domain of a particular server-to-server connection

In outsourcing scenarios, the sending and receiving domains are outsourced to the originating and receiving domains, respectively.

### 3. Protocol Overview

Consider a scenario in which the domain `sender.tld` has outsourced XMPP services to `originating.tld`, and `target.tld` has outsourced to `receiving.tld`. The particular hosts providing services are `xmpp1.originating.tld` and `xmpp1.receiving.tld`. Users `romeo@sender.tld` and `juliet@target.tld` maintain client-to-server connections to these servers.

```
romeo@sender.tld -- xmpp1.originating.tld
```

```
xmpp1.receiving.tld -- juliet@target.tld
```

When Romeo wants to send a message to Juliet, Provider A’s server will have to establish a server-to-server connection to Provider B’s server. Since they are both acting on behalf of other domains, however, each side will have to verify that the other is authorized to act in that role.

The first step is to provision records that can be used to verify these delegations. In order for XMPP to work, when the hosting relationships are set up, `sender.tld` and `target.tld` have to provision SRV records pointing to their providers’ servers. To make this delegation secure, they sign these records using DNSSEC [RFC4033]. On the XMPP servers themselves, the originating and receiving domains provision certificates that can be used to authenticate the names `xmpp1.originating.tld` and `xmpp1.receiving.tld`.

When Romeo wants to send a stanza to Juliet, he will first send it to his server, `xmpp1.originating.tld`. Seeing that the ‘to’ domain of the stanza is `target.tld`, the server will retrieve the SRV records for `_xmpp-server._tcp.target.tld`, plus any associated DNSSEC records.
If there are no DNSSEC records, or if the DNSSEC records do not validate, then there is nothing new to do; the server simply connects to the remote domain using normal XMPP procedures. If there is a valid DNSSEC signature on the SRV record, then the server knows that he can allow the remote server to authenticate as either target.tld or xmpp1.receiving.tld.

Once the TLS connection is established, the two sides negotiate a single bidirectional stream to run over it, using their own names:

I: <?xml version='1.0'?>
<stream:stream
  from='xmpp1.originating.tld'
  to='xmpp1.receiving.tld'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:server'
  xmlns:stream='http://etherx.jabber.org/streams'>

R: <?xml version='1.0'?>
<stream:stream
  from='xmpp1.receiving.tld'
  id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
  to='xmpp1.originating.tld'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:server'
  xmlns:stream='http://etherx.jabber.org/streams'>

R: <stream:features>
  <starttls xmlns='urn:ietf:params:xml:ns:xmpp-tls'/>
  <bidi xmlns='urn:ietf:params:xml:ns:xmpp-bidi'/>
</stream:features>

When this stream is created, it can immediately carry stanzas.
directly between the two servers. In order to send messages to and from other domains, the servers have to authenticate and request permission. So to send Romeo’s stanza to Juliet, xmpp1.originating.tld requests permission to send from sender.tld to target.tld.

The originating server uses STARTTLS to set up a TLS connection. In the ClientHello message initiating the connection, the xmpp1.originating.tld includes a Server Name Indication extension set to xmpp1.receiving.tld [RFC4366]. The remote server xmpp1.receiving.tld responds to this request with a certificate for its own name, xmpp1.receiving.tld and requests a client certificate from the originating server. The originating server presents a certificate for its own name, xmpp1.originating.tld.

At this point, the server xmpp1.originating.tld knows that xmpp1.receiving.tld (via the certificate) or target.tld (via DNSSEC). The other server, xmpp1.receiving.tld knows only that the other server represents xmpp1.originating.tld.

Once the two servers have authenticated their own names over TLS, they can request permission to send stanzas:

I: <db:result from='sender.tld' to='target.tld' />

Since xmpp1.receiving.tld doesn’t yet know whether xmpp1.originating.tld is authorized to represent sender.tld, it has to check, using an abbreviated form of dialback. Just as the Provider A server did earlier for target.tld, the Provider B server looks up the SRV records for _xmpp-server._tcp.sender.tld and any associated DNSSEC records. If there are no DNSSEC records or the signature is not valid, then the server rejects the request to send stanzas from that domain. If the record is DNSSEC-signed, then the server checks that the server name in the SRV record is one of the names authenticated for the remote side.

R: <db:result type='invalid' from='sender.tld' to='target.tld' />

On the other hand, if the DNSSEC signature is valid, then the server can accept the request to send stanzas, and the two servers can exchange stanzas for those domains.

R: <db:result type='valid' from='sender.tld' to='target.tld' />

I: <!-- stanza -->

Now that the two servers have established this connection, they can re-used it for other stanzas and other domains. If either server finds another domain that is delegated to the other server, it can
send a `<db:result>` requesting permission to send stanzas for that domain, and the other server will grant or deny permission after checking the delegation.

The following figure summarizes the overall process:

<table>
<thead>
<tr>
<th>Originating Server</th>
<th>DNS Server</th>
<th>Receiving Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup _xmpp-server DNS SRV record for target.tld to find delegation of service to Receiving Server. Verify zone signature</td>
<td>'Receiving Server'</td>
<td></td>
</tr>
</tbody>
</table>

`<stream from='originating.tld' to='receiving.tld'>`<features><starttls></features><proceed/>`<stream from='receiving.tld' to='originating.tld'>`<features><bidi></features>`

`<stream from='originating.tld' to='receiving.tld'>`<features><starttls></features><proceed/>`<stream from='receiving.tld' to='originating.tld'>`<features><bidi></features>`

Barnes & Lindberg Expires September 15, 2011 [Page 7]
4. Connection Model

The core challenge for managing inter-server connections is the multiplexing of stanzas for multiple domains onto a single transport-layer connection. There are two key pieces of state associated with this multiplexing: A list of domain names that have been authenticated for use on a connection, and a table binding pairs of domains that are authorized for a connection.

First table that a server maintains is a connection table. Each entry in this table contains a connection and a set of domain names. The domain names represent the set of names for which the remote server has been authenticated, according to the procedures described in Section 5. This set of domain names constrains the set of domain pairs that can be bound to this channel; the remote server cannot ask to transmit stanzas for an unauthenticated domain name.

<table>
<thead>
<tr>
<th>Connection</th>
<th>Server Domain Names</th>
<th>Delegated Domain Names</th>
</tr>
</thead>
<tbody>
<tr>
<td>XXX</td>
<td>xmpp1.provider.com</td>
<td>capulet.example</td>
</tr>
<tr>
<td>YYY</td>
<td>xmpp2.provider.com</td>
<td>capulet.example</td>
</tr>
<tr>
<td>AAA</td>
<td>paris.example</td>
<td>paris.example</td>
</tr>
</tbody>
</table>

To determine how to handle incoming and outgoing stanzas, each server maintains a channel binding table. Each row in the binding table contains a "local" domain name, a "remote" domain name, and an ordered list of connections. The identifier for a connection is the stream ID for the single XMPP stream that it carries.

<table>
<thead>
<tr>
<th>Local</th>
<th>Remote</th>
<th>Connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>montague.example</td>
<td>capulet.example</td>
<td>XXX, YYY</td>
</tr>
<tr>
<td>laurence.example</td>
<td>capulet.example</td>
<td>AAA</td>
</tr>
<tr>
<td>laurence.example</td>
<td>paris.example</td>
<td>YYY, AAA</td>
</tr>
</tbody>
</table>
The binding table acts as a routing table for outgoing stanzas and a filter for incoming stanzas. When the server wishes to send a stanza, it looks in the binding table for a row that has the 'from' domain as the local domain and the 'to' domain as the remote domain. If there is such a row in the binding table, then the server MUST transmit the stanza on the first connection in the connection list. Thus, in the above example, a stanza from montague.example to capulet.example would be routed on channel XXX.

In the same way, when a server receives a stanza over a connection from a remote server, it looks up the relevant entry in the binding table, this time using the 'to' domain as the local domain and the 'from' domain as the remote domain. If the server finds a binding table entry and the connection over which the stanza arrived is listed in the entry, then it accepts the stanza. Otherwise, it MUST discard the stanza and return a stanza error <invalid-connection/>.

In the above example, a stanza from capulet.example to escalus.example would be accepted on connections AAA and BBB, but no others.

When a connection is opened (and at some points thereafter), entries in the name table are established using the processes in Section 5. Once a connection is open, binding table entries are added or removed using the processes in Section 6. When a connection is closed, both servers MUST delete its entry in the name table and remove it from all entries in the binding table.

5. Channel Establishment and Authentication

When a server wants to send a stanza and doesn’t have an entry in the connection table for the destination domain, it sets one up. The first step is to establish a connection to a server for the destination domain, and validate that the server is authorized to represent the destination domain.

The originating server MUST take the following steps to establish a secure connection to the server for example.com:

1. Retrieve SRV records for XMPP services for example.com [I-D.ietf-xmpp-3920bis].

2. Verify that the SRV records have been signed using DNSSEC [RFC4033]. The originating server may either retrieve DNSSEC records directly or rely on a validating resolver. If the SRV records are not secured with DNSSEC, then the connection fails.
3. If there is already a connection in the connection table that has the target of any SRV record in its "server names" list, then this process terminates and the server attempts to use that connection (See Section 6)

4. If there is no existing connection that matches, establish a TCP connection to any of the servers listed in an SRV record and negotiate an XMPP stream with the following parameters:
   * 'from' domain: The originating server’s name
   * 'to' domain: The receiving server’s name from the SRV record
   * [[ TODO: Add a stream feature to indicate support for this extension ]]

5. Upgrade the connection to TLS using STARTTLS, using a cipher suite that requires the server to present an X.509 certificate.

6. Verify that the certificate is valid and chains to a local trust anchor. If the certificate is invalid, the connection fails.

7. Construct a list of all names that the certificate presents [I-D.saintandre-tls-server-id-check].

8. Verify that the target name in the SRV record is one of the names in the certificate. If the target name is not found in the list of names from the certificate, then the connection fails.

A server receiving such a connection MUST perform the following steps:

1. Accept the TCP connection from the remote side and accept the stream negotiation using server names.

2. In the TLS negotiation, require a client certificate from the remote side.

3. Verify that the remote server name in the stream matches the client certificate [I-D.saintandre-tls-server-id-check]. If the certificate does not match, the TLS negotiation fails, and the server MAY terminate the TCP connection.

If this process establishes a new connection, then the originating server knows that it has established a connection to a server that legitimately represents example.com. It should thus initialize a row in the connection table for this connection:
o Server names: The list of names in the server’s certificate

o Delegated names: example.com

If the process terminated at Step 3, then the server simply updates the connection table entry to add example.com to the list of delegated names. In either case, the row for a connection is removed from the connection table when the connection is closed.

In order for this process to work, the domain owner and the hosting provider need to publish information that other XMPP entities can use to verify the delegation. XMPP services are delegated via SRV records (see Section 3.2.1 of [I-D.ietf-xmpp-3920bis]), so in order for the delegation to be secure, the domain owner MUST sign these records with DNSSEC. In other words, if the delegated domain is example.com, then the zone _xmpp-server._tcp.example.com MUST be signed. Each server that acts for a domain MUST be provisioned with a certificate that contains the target name used by SRV records.

The server on the receiving end of the TLS connection MUST request a client certificate from the originating server during the TLS handshake, and the originating server MUST provide a client certificate. The receiving server can then also initialize an entry in its connection table to which delegated names can be added later:

o Server names: The list of names from the client certificate (from the originating server), if present. Otherwise, empty.

o Delagated names: Empty.

Once the two servers have established a TLS connection, they MUST set up an XMPP stream that will be used for domains that they represent. This process follows the normal stream initiation procedure [I-D.ietf-xmpp-3920bis], except that the ‘to’ and ‘from’ domains MUST be set to the names of the servers themselves: The originating server sends a <stream> stanza with the ‘from’ domain set to a name for itself that is contained in its client certificate, and the ‘to’ domain set to the server name used in the SRV record for this connection. If stream negotiation fails, then the connection fails. If it succeeds, then both sides MUST set the connection identifier in the connection table to be the stream ID for the negotiated stream.

Since server-to-server connections are by default directional, it is RECOMMENDED that servers also request the <bidi> stream feature to enable bidirectional flows on this connection [XEP-0288].
<table>
<thead>
<tr>
<th>Originating Server</th>
<th>DNS Server</th>
<th>Receiving Server</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lookup _xmpp-server</td>
<td>DNS SRV record for target.tld to find delegation of service to Receiving Server. Verify zone signature</td>
<td>'Receiving Server'</td>
</tr>
</tbody>
</table>

```
<stream from='originating.tld' to='receiving.tld'>
  <features><starttls></features>
  <starttls/>
  <proceed/>
</stream>
```

```
<stream from='receiving.tld' to='originating.tld'>
  <features><bidi></features>
</stream>
```
6. Authorizing XMPP Stanzas

Before sending traffic from a Sender Domain to a Target Domain using an established connection, the originating server MUST request permission to do so, and wait until it has received authorization from the remote service. A service receiving traffic MUST verify that the Sender and Target domain pair has been authorized on the connection being used.

An originating server MUST go through the following steps to request authorization to send traffic from a Sender Domain to a Target Domain:

1. Send a `<db:result/>` [XEP-0220] element with Sender Domain as 'from' and Target Domain as 'to'. The server may also include a Dialback Key as part of the element’s character data, to support legacy deployments.

2. Wait for remote service to respond with a `<db:result>` with Target Domain as 'from', Sender Domain as 'to' and 'type' attribute that is either 'valid' or 'invalid'. In case of 'invalid', the originating server SHOULD examine the error cause and take appropriate action and MAY retry requesting authorization on the same connection in the future.

3. If response 'type' was 'valid', the originating server updates its binding table to indicate that Sender Domain (Local) and Target Domain (Remote) is authorized in the sending direction for the connection used.

4. Originating server proceeds with sending traffic from Sender Domain to Target Domain.

Upon receiving a `<db:result/>` stanza, the receiving server MUST take following steps:

1. Verify that the receiving direction is supported for this connection. If not, fail by disconnecting the stream. (By default, connections are one-way)

2. Verify that domain in to-attribute is hosted by the service. If not, fail and respond with an <item-not-found/> error.

3. Verify that domain in from-attribute delegates hosting of their XMPP to the remote Server Domain Name by looking up SRV and verifying that the zone is signed. If not, fail with a <not-authorized/> error. Note: a service MAY accept a less secure delegation mechanism such as a SRV records in a non signed zone,
subject to local policy.

4. Once secure delegation from Sending Domain to remote Server Domain name has been verified, service adds Sending Domain to list of Delegated Domain Names in the Connection Table, and updates the Binding Table indicating that the Sending Domain (remote) is allowed to send traffic to Target Domain (local) on the connection.

5. Respond to remote service with a <db:result/> stanza with ‘type’ set to ‘valid’.

A service may revoke authorization for a domain pair at any time by sending a <db:result> with ‘type’ set to invalid. Once authorization has been revoked, the remote side MUST re-secure authorization before sending any further traffic for the domain pair.

If a server receives a stanza for a to/from pair that it does not consider authorized, then it MUST return a <not-authorized/> error and MAY terminate the TCP connection.
7. Backward Compatibility

Using Server Domain Names as to/from attributes in <stream> stanzas is incompatible with XMPP services that do not support this protocol, because it was previously assumed that when receiving a connection the stream to attribute will contains an XMPP domain hosted by the receiving service. It is RECOMMENDED that if the connection fails, the service tries again using the Remote Domain as stream to-attribute.
Presenting a certificate for the Server Domain Name is incompatible with XMPP services that do not support this protocol, because those will expect the Remote Domain in the certificate. It is RECOMMENDED that if the authorization fails, the service tries again presenting the certificate for the Remote Domain. A service may also choose to fall back on a weaker identification mechanism such as Server Dialback, subject to local policy.

8. Operational Considerations

[[ What names to put in certs for servers in a cluster, i.e., all of them. ]]

[[ Do TLS clients support multiple names in certs? ]]

[[ How DNSSEC validation is done can vary depending on deployment scenario. ]]

[[ Since SNI is used to signal support for this extension, recommended not to serve end users on the same domain as hosting services. ]]

[[ Load balancing thoughts, since each connection will handle a lot more traffic? ]]

9. IANA Considerations

[[ Register XML schema for assertions, if necessary ]]

[[ Define invalid-connection error element ]]

10. Security Considerations

[[ This document simplifies authentication and authorization of XMPP servers in certain scenarios. When used together with DNSSEC-protected delegations, it does not introduce any new security risks. ]]

[[ If a provider chooses to omit DNSSEC checks or ]]

11. Acknowledgements

Thanks to Joe Hildebrand and Sean Turner for prompting the original work on this problem, and to Stephen Farrell for his work on initial
versions of this draft.

12. Normative References

[I-D.ietf-xmpp-3920bis]

[I-D.saintandre-tls-server-id-check]


Authors' Addresses

Richard L. Barnes
BBN Technologies

Email: rbarnes@bbn.com

Jonas Lindberg
Google

Email: jonasl@google.com
Abstract

This document describes current practices for combined use of the Session Initiation Protocol (SIP) and the eXtensible Messaging and Presence Protocol (XMPP). Such practices aim to provide a single fully featured real-time communication service by using complimenting subsets of features from each of the protocols. Typically such subsets would include telephony oriented from SIP and instant messaging and presence capabilities from XMPP. This specification does not define any new protocols or syntax for neither SIP nor XMPP. However, implementing it may require modifying or at least reconfiguring existing client and server-side software. Also, it is not the purpose of this document to make recommendations as to whether or not such combined use should be preferred to the mechanisms provided natively by each protocol like for example SIP’s SIMPLE or XMPP’s Jingle. It merely aims to provide guidance to those who are interested in such a combined use.

Copyright Notice

Ivov & Marocco          Expires September 6, 2012               [Page 1]
Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Introduction</td>
<td>3</td>
</tr>
<tr>
<td>2. Terminology</td>
<td>4</td>
</tr>
<tr>
<td>3. Client Bootstrap</td>
<td>4</td>
</tr>
<tr>
<td>4. Operation</td>
<td>5</td>
</tr>
<tr>
<td>5. Security Considerations</td>
<td>6</td>
</tr>
<tr>
<td>6. Acknowledgements</td>
<td>6</td>
</tr>
<tr>
<td>7. References</td>
<td>6</td>
</tr>
<tr>
<td>7.1. Normative References</td>
<td>6</td>
</tr>
<tr>
<td>7.2. Informative References</td>
<td>6</td>
</tr>
<tr>
<td>Authors’ Addresses</td>
<td>8</td>
</tr>
</tbody>
</table>
1. Introduction

Historically SIP [RFC3261] and XMPP [RFC6120] have often been implemented and deployed with different purposes: from its very start SIP’s primary goal has been to provide a means of conducting "Internet telephone calls". XMPP on the other hand, has from its Jabber days been mostly used for its instant messaging and presence capabilities.

For various reasons, these trends have continued through the years even after each of the protocols had been equipped to provide the features it was initially lacking.

Today, in the context of the SIMPLE working group, the IETF has defined a number of protocols and protocol extensions that not only allow for SIP to be used for regular instant messaging and presence but that also provide mechanisms for elaborated features such as multi-user chats, server-stored contact lists, file transfer and others.

Similarly, the XMPP community and the XMPP Standards Foundation have worked on defining a number of XMPP Extension Protocols (XEPs) that provide XMPP implementations with the means of establishing end-to-end sessions. These extensions are often jointly referred to as Jingle and their arguably most popular use case are audio and video calls.

Yet, despite these advances SIP remains the protocol of choice for telephony-like services, especially in enterprises where users are accustomed to features such as voice mail, call park, call queues, conference bridges and many others that are rarely (if at all) available in Jingle servers. XMPP implementations on the other hand, greatly outnumber and outperform those available for protocols recommended by SIMPLE, such as [MSRP] and [XCAP].

For these reasons in a number of cases, adopters may find themselves needing a set of features that are not offered by any single-protocol solution but that separately exist in SIP and XMPP products. The idea of seamlessly using both protocols together would hence often appeal to service providers.

Most often such combined use would employ SIP exclusively for audio, video and telephony services and it would rely on XMPP for anything else varying from chat, roster management and presence to exchanging files.

This document explains how the above could be achieved with a minimum amount of modifications on existing software while providing an
optimal user experience. It tries to cover points such as server
discovery, determining a SIP AOR while using XMPP and an XMPP JID
from incoming SIP requests. Most of the text here pertains to client
behavior but it also recommends certain server-side configurations.

2. Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT",
"SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this
document are to be interpreted as described in [RFC2119].

3. Client Bootstrap

One of the main problems of using two distinct protocols when
providing one service is how it affects usability. E-mail services
for example have long been affected by the mixed use of SMTP on for
outgoing mail and POP3 and IMAP for incoming, making it rather
complicated for inexperienced users to configure a mail client and
start using it with a new service. As a result mailing list services
often need to provide configuration instructions for various mail
clients. Client developers and communications device manufacturers
on the other hand often ship with a number of wizards that allow to
easily set up a new account for a number of popular e-mail services.
While this may improve the situation to some extent, user experience
is still clearly sub-optimal.

While it should be possible for CUSAX users to manually configure
their separate SIP and XMPP accounts, it is RECOMMENDED that dual
stack SIP/XMPP clients provide means of online provisioning. While
the specifics of such mechanisms are not in the scope of this
specification, they should make it possible for service providers to
remotely configure the clients based on minimal user input (e.g. user
id and password).

Given that many of the features that CUSAX would privilege in one
protocol would also be available in the other, clients should make it
possible for such features to be disabled for a specific account.
Specifically it is RECOMMENDED that clients allow for audio/video
calling features to be disabled for XMPP accounts. Additionally
instant messaging and presence features MAY also be made optional for
SIP accounts.

The main advantage of the above would be that clients would be able
to continue to function properly and use the complete feature set of
stand-alone SIP and XMPP accounts.
Once client bootstrap has completed, clients SHOULD log independently to the SIP and XMPP accounts that make up the CUSAX service and should maintain both these connections. In order to improve user experience, when reporting connection status clients may also wish to present the CUSAX XMPP connection as an "instant messaging" or a "chat" account. Similarly they could also depict the SIP CUSAX connection as a "Voice and Video" or a "Telephony" connection. The exact naming is of course entirely up to implementors. The point is that such presentation could help users better understand why they are being shown two different connections for a single service. It could even alleviate especially situations where one of these connections is disrupted while the other one is successfully maintained.

4. Operation

Once a CUSAX client has been provisioned/configured to connect to the corresponding SIP and XMPP services it would proceed by retrieving its XMPP roster. In order for CUSAX to function properly, XMPP service administrators should make sure that at least one of the [VCARD] "tel" fields for each contact is properly populated with a SIP URI or a phone number. There are no limitations as to the form of that number (e.g. it does not need to respect any equivalence with the XMPP JID). It SHOULD however be reachable through the SIP counterpart of this CUSAX service.

In order to make sure that the above is always respected, service maintainers MAY prevent clients (and hence users) from modifying the VCARD "tel" fields or they MAY apply some form of validation before recording changes.

When rendering the XMPP roster CUSAX clients should make sure that users are presented with a "Call" option for each roster entry that has a properly set "tel" field even if calling has been disabled for that particular XMPP account. The usefulness of such a feature is not limited to CUSAX. After all, numbers are entered in VCARDs in order to be dialed and called. Hence, as long as an XMPP client is equipped with accounts that have calling features it may wish to present the user with the option of using these accounts to reach numbers from an XMPP VCARD. In order to improve usability, in cases where clients are provisioned with only a single telephony capable account they SHOULD do so immediately upon user request without asking for confirmation. This way CUSAX users whose only account with calling capabilities would often be the SIP part of their service would be having better user experience. If on the other hand, the CUSAX client is aware of multiple telephony-capable accounts, it SHOULD present the user with the choice of reaching the
phone number through any of them (including the source XMPP account where the VCARd was obtained) in order to guarantee proper operation for XMPP accounts that are not part of a CUSAX deployment.

The client should use XMPP for all other forms of communication with the contacts from its roster so it should and this should occur naturally given that they were retrieved through XMPP.

When receiving SIP calls, clients may wish to determine the identity of the caller and bind it to a roster entry so that users could revert to chatting or other forms of communication that require XMPP. To do so clients could search their roster for an entry whose VCARd has a "tel" field matching the originator of the call.

An alternate mechanism would be for CUSAX clients to add to their SIP invite requests a contact header containing their XMPP JID, but at this point we are not really sure if that’s such a good idea. (After all Contact headers carry URIs and JIDs are not URIs).

5. Security Considerations

TBD

6. Acknowledgements

This draft is inspired by work from Markus Isomaki and Simo Veikkolainen.

7. References

7.1. Normative References


7.2. Informative References


[XEP-0177]

Authors’ Addresses

Emil Ivov
Jitsi
Strasbourg  67000
France

Email: emcho@jitsi.org

Enrico Marocco (editor)
Telecom Italia
Via G. Reiss Romoli, 274
Turin  10148
Italy

Email: enrico.marocco@telecomitalia.it
Abstract


Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on August 14, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction .................................................. 2
2. Terminology .................................................. 2
3. Encrypting XMPP Stanzas ...................................... 2
   3.1. Prerequisites ............................................. 3
   3.2. Process .................................................. 3
1. Introduction

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

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

2. Terminology

This document inherits terminology defined in [RFC6120].

Security-related terms are to be understood in the sense defined in [SECTERMS].

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 [KEYWORDS].

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

3.1. Prerequisites

First, the sending agent prepares and retains the following:

- The JID of the sender (i.e., its own JID). This SHOULD be the bare JID (localpart@domainpart).
- The JID of the recipient. This SHOULD be the bare JID (localpart@domainpart).
- A random Content Encryption Key (CEK). The CEK MUST have a length at least equal to that of the required encryption keys and MUST be generated randomly. See [RFC4086] for considerations on generating random values.
- A CEK identifier (CID). The CID MUST be unique for a given (sender, recipient, CEK) tuple, and MUST NOT be derived from CEK itself.

3.2. Process

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

1. Notes the current UTC date and time N when this stanza is constructed, formatted as described under [timestamps].
2. Constructs a forwarding envelope P using a <forwarded/> element qualified by the "urn:xmpp:forward:0" namespace (as defined in [MSG-FWD]) as follows:

   * The child element <delay/> qualified by the "urn:xmpp:delay" namespace (as defined in [DELAY]) with the attribute 'stamp' set to the UTC date and time value N
   * The plaintext stanza S
3. Generates any additional unprotected block cipher factors (IV); e.g. initialization vector, nonce, and/or associated authentication data. A sending agent MUST ensure that no two sets of factors are used with the same CEK, and SHOULD NOT reuse such factors for other stanzas.
4. Constructs a partial [JOSE-JWE] header (H) with the following information:

   * The property 'enc' indicating the algorithm used to encrypt the content.
* The property 'iv' indicating the initialization vector, if required by the algorithm.

NOTE: this header is intentionally lacking required attributes. The receiving agent is responsible for assembling a final, valid JWE header.

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

6. Encrypts P' using the intended block cipher, i.e. T = block-encrypt(CEK, IV, P').

7. Constructs an <e2e/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:1" namespace as follows:

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

* The child element <header/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:1" namespace and with XML character data as the base64url-encoded form of H.

* The child element <data/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:1" namespace and with XML character data as the base64url-encoded form of T.

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

3.3. Example Securing a Message

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

The sending agent begins with the plaintext version of the <message/> stanza 'S':
Internet-Draft                  XMPP E2E                   February 2012

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

and the following prerequisites:

- Sender JID as "juliet@capulet.lit"
- Recipient JID as "romeo@montegue.lit"
- Content encryption key 'CEK' as (base64 encoded) 
  "-ElMo6FndEkMxWP3TIkp1dDFvKqmqAAgr1cvVnUVP0c="
- CEK identifier CID as "835c92a8-94cd-4e96-b3f3-b2e75a438f92"

The sending agent performs steps 1 and 2 to generate the envelope:

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

Then the sending agent performs steps 3 and 4 to generate the [JOSE-JWE] header:

{  
    "enc":"A256CCM",
    "iv":"B7waCj2vF_sLaJfe-1GHRa==",
    "adata":"1492-05-12T20:07:37.012Z",
    "msize":"16"  
}
Then the sending agent performs steps 5 and 6 to generate the protected content:

7LlMXd-qqPAQ_Lzm6u9AR2csyDgT09z5DWdn8K5GLr_qbWRDKw2ufZrzm09YZ-jHl
1lDexE9aqZbbNNViv8gpa-prDYkX0o3QoqY0j1A0RAkP0-UJn41wqVvV62gad_OB
Dd9q2xSnK1IP15frlGTcZSxO1eS3Ec0cDI_0MzMEKqPVnByQDkWQNrtxfPs2b
1E15kCqXbHvaxARez7yAa-ITXruV_fOXGftkRVDuiFyVh2xNRP-a-TQxDeg2h1D_u_c2mwPL06ED_12l1V705_VL0D0T1YGTDDqeyrzQWnQNEBj4G5jFpCyqtCszbgx
9kjWjlxYNLLGxbMOTwF450LCd8JFUUQAHoLeKp4a1wr5yp7aATX8dKvM5_TFIcT
nLonaMe5mPnRgg5zNeMEx6FFkgwO1ih7hV1_QHF8Ofoy99cmKwIV3Tq5n5gL
74Xm4CtiJMNhAEn2Q-10-fWoUHIEAOu4GwbAx0ToBw4uCM2hG1FSKYCkkJxm
71L-5jwuSuuQHX6ef8Hhi7fuqmgw9UXekeubsY2bt15put0SWT8_0S8ZBDMrkJx
y7i15NUOhQMms3ou1R1NzdvNKvcqix5q3zeB1FLWmGynmnj_gE_HalWUL0HoqL
93Fr1nFFLNhXLCZHy7By6T9NN8omp4ZY92HMpPzgo-eCGP

Then the sending agent performs step 7 and sends the following:

```xml
<message xmlns='jabber:client'
  from='juliet@capulet.lit/balcony'
  id='fJZd9WFIIWnjFctT'
  to='romeo@montegue.lit/garden'
  type='chat'>
  <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:1'
   id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <header>
      <eyJlbmMiOiJBMjU2Q0NNIiwiaXYiOiJCN3dhQ2oydZfc0xhSmZlLTFHSHJBPT0i
       LChzhGF0YSi6IjQ5M10wN50wMlQyMDowNzozN4wMTJaIiwibXNpemU0I1xNIj9
    </header>
    <data>
      7LlMXd-qqPAQ_Lzm6u9AR2csyDgT09z5DWdn8K5GLr_qbWRDKw2ufZrzm09YZ-jHl
    </data>
  </e2e>
</message>
```

4. Requesting Content Keys

A receiving agent might not have the content encryption key to decrypt the stanza.
4.1. Request Process

Before a CEK can be requested, the receiving agent MUST have at least one public key for which it also has the private key.

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

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

2. Constructs a <keyreq/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:1" namespace as follows:

   * The attribute 'id' set to the CEK identifier value CID.
   * The child element <pkey/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:1" namespace and with XML character data as the base64url-encoded form of KS.

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

4.2. Accept Process

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

1. Chooses a key (PK) from the list provided via KS, and notes its identifier value 'kid'.
2. Constructs a partial [JOSE-JWE] header (H) as follows:

   * The property 'alg' set to the cryptographic algorithm for PK, which is used to secure the content encryption key CEK.
   * The property 'kid' set to the identifier matching PK.

   NOTE: this header is intentionally lacking required attributes. The receiving agent is responsible for assembling a final, valid JWE header.

3. Encrypts the content encryption key CEK using the key PK, i.e. $\text{CEK'} = \text{pki-encrypt}(\text{PK, CEK})$.

4. Constructs a <keyreq/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:1" namespace as follows:
* The attribute 'id' set to the CEK identifier CID.

* The child element <header/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:1" namespace and with XML character data as the base64url-encoded form of H.

* The child element <cek/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:1" namespace and with XML character data as the base64url-encoded form of CEK'.

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

4.3. Error Conditions

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

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

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

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

4.4. Example of Successful Key Request

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

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

```
[{
    "alg":"RSA",
    "mod":"AL7ano5DBdNYfKChGh3xxGDTatozO3iEm20rzbNEjSLWFjuhm46SXzeJXV8Nr6VUXcrJ8FN85qV0g9GAh9VoHrIaECtZJzEw3U18edntXOAu81ZQPi7r7ru1pqQanrJ9rJ2Zgd42LCTpFQWMAib2AfQD2gb5NIzZUbys4wn9Rx52ru5xkrcrIYLPVrK8Yv3fKwuU0Y0qf809Yunz_H8A5aHplzrQtFOuW8Ut_tWR8Wn_2w_H6Y6bOoMEoHG5J3CQCNGz1mVrjKadp7amtMCwesXcW9iI63qIP70bLAKEolw1XaLME6d-mT_tZZwpNo_r2eKm4WhYOW2bYLoME=",
    "exp":"AQAB",
    "kid":"romeo@montegue.lit/garden"
  }]```
Then the receiving agent performs step 2 to generate the `<keyreq/>`:

```xml
<keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:1'
   id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
  <pkey>
    W3siYWxnIjoiUlNBIiwibW9kIjoiqW3YW5vNURCZE5ZktDaEdoM3h4R0R0QXR
    ve8zaUVtmJByemJORwpTTFdGanVobTQ2U1h6ZUpYVjhOcjZWVWjck04Rk44NU
    9xTnYw2zlHQwg5V9iCml0R0UN0wp6RXczVFVSoGVKbnyRT0VQV0aU0BJadyc
    luXcHFRY5Y5jslal dacWdhKNDJMQL1RjXUT4pY1pBZlFEMmd1NU5JcIpZY1lz
    NHduOVJ4NTJydTV4a3Jjck12THZQcks4WXYzZkt3dUweTBN2jgwOvllbnpsDh
    BNWFICG6c1F0R91v90I14V25fMndfsDZzNmJpb01Fb0haR1NgQ1FDtk
    d6bg1CmpLYWRwN2FtdE1d2V6WGNXOW1JnNdxSVA3T2JMQtFb2x3MVhTE1Fb
    T2kLWIUtX3RaWntW7fCJ1S200V2h2TicyY11Mz0nNRT01LCJ1cHAIoiJBUUFc
    I1wia2lkIjoicm9tZ9WAbW9udGVndWUubGI0L2dhcmRlbj9XQ==
  </pkey>
</keyreq>
```

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

```xml
<iq xmlns='jabber:client'
    from='romeo@montegue.lit/garden'
    id='xdJbWMA+'
    to='juliet@capulet.lit/balcony'
    type='get'>
  <keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:1'
   id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <pkey>
      W3siYWxnIjoiUlNBIiwibW9kIjoiqW3YW5vNURCZE5ZktDaEdoM3h4R0R0QXR
      ve8zaUVtmJByemJORwpTTFdGanVobTQ2U1h6ZUpYVjhOcjZWVWjck04Rk44NU
      9xTnYw2zlHQwg5V9iCml0R0UN0wp6RXczVFVSoGVKbnyRT0VQV0aU0BJadyc
      luXcHFRY5Y5jslal dacWdhKNDJMQL1RjXUT4pY1pBZlFEMmd1NU5JcIpZY1lz
      NHduOVJ4NTJydTV4a3Jjck12THZQcks4WXYzZkt3dUweTBN2jgwOvllbnpsDh
      BNWFICG6c1F0R91v90I14V25fMndfsDZzNmJpb01Fb0haR1NgQ1FDtk
      d6bg1CmpLYWRwN2FtdE1d2V6WGNXOW1JnNdxSVA3T2JMQtFb2x3MVhTE1Fb
      T2kLWIUtX3RaWntW7fCJ1S200V2h2TicyY11Mz0nNRT01LCJ1cHAIoiJBUUFc
      I1wia2lkIjoicm9tZ9WAbW9udGVndWUubGI0L2dhcmRlbj9XQ==
    </pkey>
  </keyreq>
</iq>
```

If the sending agent accepts this key request, it performs steps 1 and 2 from accept process [keyreq-process-accept] to generate the partial [JOSE-JWE] header:

```json
{  "alg":"RSA-OEAP",  "kid":"romeo@montegue.lit/garden"}
```

Then the sending agent performs step 3 to generate the encrypted CEK:
Then the sending agent performs step 4 to generate the <keyreq/> response:

```
<k request xmlns='urn:ietf:params:xml:ns:xmpp-e2e:1'
     id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <header>
    eyJhbGciOiJSU0EtT0VBUCIsImtpZCI6InJvbWVvQG1vbnRlZ3VlLmxpc3QvZ2FyZGVuIiwicm9sZSI6Ikpvc3Rlc3RlciJ9
    </header>
    <cek>
        DCKrJpLd8XYze7joNKsyvpyzGD4MffVQF7apQMKPK8_vKx4J3xEqWB3tXxp6oxJF4YH1a48F1Mp44LccMed6JchPp_230XrBo8i1Fwo8jVH6QoQa71T_1rQj6C6WQ1eFvXiU4Qmp52F-gjuUh1Kfe3qccldp3kJy8Mpq5Mdpq6deOq0_qusqr8QTqrT5LNlgsXYyrdlDdxK8JJKnI4FBB_Hlx-A1C97DfJk9s5H7GzKvM2BRWVXXj0f41ckE2452JKa3CQk86Yh3YBetoScPVlhRo02qqBg4gR9dDO5mXnOgPsVfXWvegetH
    </cek>
</k>
```

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

```
<i xmlns='jabber:client'
    from='juliet@capulet.lit/balcony'
    id='iddJbWMA+'
    to='romeo@montegue.lit/garden'
    type='result'>
    <k xmlns='urn:ietf:params:xml:ns:xmpp-e2e:1'
        id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
        <header>
        eyJhbGciOiJSU0EtT0VBUCIsImtpZCI6InJvbWVvQG1vbnRlZ3VlLmxpc3QvZ2FyZGVuIiwicm9sZSI6Ikpvc3Rlc3RlciJ9
        </header>
        <cek>
            DCKrJpLd8XYze7joNKsyvpyzGD4MffVQF7apQMKPK8_vKx4J3xEqWB3tXxp6oxJF4YH1a48F1Mp44LccMed6JchPp_230XrBo8i1Fwo8jVH6QoQa71T_1rQj6C6WQ1eFvXiU4Qmp52F-gjuUh1Kfe3qccldp3kJy8Mpq5Mdpq6deOq0_qusqr8QTqrT5LNlgsXYyrdlDdxK8JJKnI4FBB_Hlx-A1C97DfJk9s5H7GzKvM2BRWVXXj0f41ckE2452JKa3CQk86Yh3YBetoScPVlhRo02qqBg4gR9dDO5mXnOgPsVfXWvegetH
        </cek>
    </k>
</i>
```

5. Handling of Inbound Encrypted Stanzas

Several scenarios are possible when an entity receives an encrypted stanza:
o The receiving agent does not understand the protocol.

o The receiving agent understands the protocol but does not have enough information to decrypt the payload.

o The receiving agent understands the protocol but is unable to decrypt the payload.

o The receiving agent understands the protocol and is able to decrypt the payload, but the timestamps fail the checks specified under [timestamps].

o The receiving agent understands the protocol and is able to decrypt the payload (success case).

5.1. Protocol Not Understood

In the case where the receiving agent does not understand the protocol, it MUST do one and only one of the following: (1) ignore the <e2e/> extension, (2) ignore the entire stanza, or (3) return a <service-unavailable/> error to the sender, as described in [RFC6120].

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

5.2. Insufficient Information

In the case where the protocol is understood but the receiving agent does not have enough information to decrypt the payload, it SHOULD request the additional information as described in [keyreq].

If the key request fails, or the receiving agent could not otherwise determine the additional information, it MAY return a <bad-request/> error to the sending agent (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <insufficient-information/>:

<message xmlns='jabber:client'
    from='juliet@capulet.lit/balcony'
    id='fJZd9WPIwNjFctT'
    to='romeo@montegue.lit/garden'
    type='chat'>
    <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:1'
        id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
        <header>[XML character data]</header>
        <data>[XML character data]</data>
    </e2e>
    <error type='modify'>
        <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
        <insufficient-information xmlns='urn:ietf:params:xml:ns:xmpp-e2e'/>
    </error>
</message>
In addition to returning an error, the receiving agent SHOULD NOT present the stanza to the intended recipient (human or application) and SHOULD provide some explicit alternate processing of the stanza (which MAY be to display a message informing the recipient that it has received a stanza that cannot be decrypted).

5.3. Failed Decryption

In the case where the protocol is understood but the receiving agent is unable to decrypt the payload, the receiving agent SHOULD return a <bad-request/> error to the sending agent (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <decryption-failed/> (previously defined in [RFC3923]):

```
<message xmlns='jabber:client'
    from='juliet@capulet.lit/balcony'
    id='fJZd9WFIIwNjFctT'
    to='romeo@montegue.lit/garden'
    type='chat'>
    <e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:1'
        id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
        <header>[XML character data]</header>
        <data>[XML character data]</data>
    </e2e>
    <error type='modify'>
        <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
        <decryption-failed xmlns='urn:ietf:params:xml:ns:xmpp-e2e'/>
    </error>
</message>
```

In addition to returning an error, the receiving agent SHOULD NOT present the stanza to the intended recipient (human or application) and SHOULD provide some explicit alternate processing of the stanza (which MAY be to display a message informing the recipient that it has received a stanza that cannot be decrypted).

5.4. Timestamp Not Acceptable

In Case #4, the receiving agent MAY return a <not-acceptable/> error to the sender (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <bad-timestamp/> (previously defined in [RFC3923]):
5.5. Successful Decryption

In the case where the protocol is understood and the receiving agent successfully decrypted the payload, it MUST NOT return a stanza error.

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

6. Inclusion and Checking of Timestamps

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

- It MUST verify that the timestamp received is within five minutes of the current time, except as described below for offline messages.
- It SHOULD verify that the timestamp received is greater than any timestamp received in the last 10 minutes which passed the previous check.
- If any of the foregoing checks fails, the timestamp SHOULD be presented to the receiving entity (human or application) marked as "old timestamp", "future timestamp", or "decreasing timestamp", and the receiving entity MAY return a stanza error to the sender.
The foregoing timestamp checks assume that the recipient is online when the message is received. However, if the recipient is offline then the server will probably store the message for delivery when the recipient is next online (offline storage does not apply to <iq/> or <presence/> stanzas, only <message/> stanzas). As described in [OFFLINE], when sending an offline message to the recipient, the server SHOULD include delayed delivery data as specified in [DELAY] so that the recipient knows that this is an offline message and also knows the original time of receipt at the server. In this case, the recipient SHOULD verify that the timestamp received in the encrypted message is within five minutes of the time stamped by the recipient’s server in the <delay/> element.

7. Interaction with Stanza Semantics

The following limitations and caveats apply:

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

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

8. Mandatory-to-Implement Cryptographic Algorithms

All algorithms that MUST be implemented for [JOSE-JWE] also MUST be implemented for this specification.

9. Security Considerations

9.1. Storage of Encrypted Stanzas

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

9.2. Re-use of Content Encryption Keys

A sender SHOULD NOT use the same CEK for stanzas intended for different recipients.

A sender MAY re-use a CEK for several stanzas to the same recipient. In this case, the CID remains the same, but MUST generate a new IV (and other data) for each encrypted stanza. The sender SHOULD periodically generate a new CEK; however, this specification does not mandate any specific algorithms or processes.

10. IANA Considerations

10.1. XML Namespace Name for e2e Data in XMPP
A URN sub-namespace of encrypted content for the Extensible Messaging and Presence Protocol (XMPP) is defined as follows.

URI: urn:ietf:params:xml:ns:xmpp-e2e:1
Specification: RFC XXXX
Description: This is an XML namespace name of encrypted content for the Extensible Messaging and Presence Protocol as defined by RFC XXXX.
Registrant Contact: IESG, <iesg@ietf.org>

11. References

11.1. Normative References


[JOSE-JWK] Jones, M., "JSON Web Key (JWK)", Internet-Draft draft-ietf-jose-json-web-key-00, January 2012.
11.2. Informative References


Appendix A. Schema for urn:ietf:params:xml:ns:xmpp-e2e:1

The following XML schema is descriptive, not normative.
<?xml version='1.0' encoding='UTF-8'?>
<xs:schema
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  targetNamespace='urn:ietf:params:xml:ns:xmpp-e2e:1'
  xmlns='urn:ietf:params:xml:ns:xmpp-e2e:1'
  elementFormDefault='qualified'>
  <xs:element name='e2e'>
    <xs:complexType>
      <xs:attribute name='id' type='xs:string' use='required'/>
      <xs:sequence>
        <xs:element ref='header' minOccurs='1' maxOccurs='1'/>
        <xs:element ref='data' minOccurs='1' maxOccurs='1'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  
  <xs:element name='keyreq'>
    <xs:complexType>
      <xs:attribute name='id' type='xs:string' use='required'/>
      <xs:sequence>
        <xs:element ref='header' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='pkey' minOccurs='0' maxOccurs='1'/>
        <xs:element ref='cek' minOccurs='0' maxOccurs='1'/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  
  <xs:element name='cek'>
    <xs:complexType>
      <xs:simpleType>
        <xs:extension base='xs:string'>
        </xs:extension>
      </xs:simpleType>
    </xs:complexType>
  </xs:element>
  
  <xs:element name='data'>
    <xs:complexType>
      <xs:simpleType>
        <xs:extension base='xs:string'>
        </xs:extension>
      </xs:simpleType>
    </xs:complexType>
  </xs:element>
  
  <xs:element name='header'>
    <xs:complexType>
      <xs:simpleType>
        <xs:extension base='xs:string'>
        </xs:extension>
      </xs:simpleType>
    </xs:complexType>
  </xs:element>
</xs:schema>
<xs:element name='pkey' type='empty'/>
<xs:element name='bad-timestamp' type='empty'/>
<xs:element name='decryption-failed' type='empty'/>
<xs:element name='insufficient-information' type='empty'/>

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

Author’s Address

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

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

This document describes how to prepare and compare Unicode strings representing nicknames, primarily as used within textual chatrooms. This profile is intended to be used by chatroom technologies based on both the Extensible Messaging and Presence Protocol (XMPP) and the Message Session Relay Protocol (MSRP).

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on September 6, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
Table of Contents

1. Introduction ................................................. 3
   1.1. Overview ............................................. 3
   1.2. Terminology ......................................... 3
2. Rules ......................................................... 3
3. Security Considerations ................................. 4
   3.1. Reuse of PRECIS .................................... 4
   3.2. Reuse of Unicode .................................. 4
   3.3. Visually Similar Characters ...................... 5
4. IANA Considerations ........................................ 5
5. References .................................................. 5
   5.1. Normative References ............................... 5
   5.2. Informative References ............................ 6

Author’s Address ............................................ 6
1. Introduction

1.1. Overview

Technologies for textual chatrooms customarily enable participants to specify a nickname for use in the room; e.g., this is true of Internet Relay Chat [RFC2811], Multi-User Chat (MUC) based on the Extensible Messaging and Presence Protocol (XMPP) [XEP-0045], and multi-party chat based on the Message Session Relay Protocol (MSRP) [I-D.ietf-simple-chat]. Recent chatroom technologies also allow internationalized nicknames because they support characters from the outside the ASCII range, typically by means of the Unicode character set [UNICODE]. Although such nicknames are often used primarily for display purposes, they are sometimes used for programmatic purposes as well (e.g., kicking users or avoiding nickname conflicts).

To increase the likelihood that nickname input and comparison will work in ways that make sense for typical users throughout the world, this document defines rules for preparing and comparing internationalized nicknames.

1.2. Terminology

Many important terms used in this document are defined in [I-D.ietf-precis-framework], [RFC6365], and [UNICODE]. Relevant XMPP terms are defined in [RFC6120] and [XEP-0045], and relevant MSRP terms in [RFC4975] and [I-D.ietf-simple-chat].

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Rules

A nickname MUST NOT be zero bytes in length and MUST NOT be more than 1023 bytes in length (the latter restriction is derived from the length restriction on XMPP resourceparts, see [RFC6122]). This rule is to be enforced after any mapping or normalization of code points.

A nickname MUST consist only of Unicode code points that conform to the "FreeClass" base string class defined in [I-D.ietf-precis-framework].

For preparation purposes (e.g., when a chatroom client generates a nickname from user input for inclusion as a nickname protocol element), an application MUST only ensure that the string conforms to
the "FreeClass" base string class defined in [I-D.ietf-precis-framework]; however, it MAY also perform the mapping and normalization operations specified below for comparison.

For comparison purposes (e.g., when a chatroom server determines if two nicknames match during the authorization process), an application MUST treat a nickname as follows, where the operations specified MUST be completed in the order shown:

1. Non-ASCII space characters from the "N" category defined under Section 6.14 of [I-D.ietf-precis-framework] MUST be mapped to SPACE [U+0020].

2. Uppercase and titlecase characters MUST be mapped to their lowercase equivalents. In applications that prohibit matching nicknames, this rule helps to reduce the possibility of confusion by ensuring that nicknames differing only by case (e.g., "stpeter" vs. "StPeter") would not be allowed in a room at the same time.

3. All characters MUST be mapped using Unicode Normalization Form KC (NFKC). Because NFKC is more "aggressive" in finding matches than other normalization forms (in the language of Unicode, it performs both canonical and compatibility decomposition before recomposing code points), this rule helps to reduce the possibility of confusion by increasing the number of characters that would match (e.g., ROMAN NUMERAL FOUR [U+2163] would match the combination of LATIN CAPITAL LETTER I [U+0049] and LATIN CAPITAL LETTER V [U+0056]).

For both preparation and comparison, the "Bidi Rule" provided in [RFC5893] applies to the directionality of a nickname.

3. Security Considerations

3.1. Reuse of PRECIS

The security considerations described in [I-D.ietf-precis-framework] apply to the "FreeClass" base string class used in this document for nicknames, respectively.

3.2. Reuse of Unicode

The security considerations described in [UTR39] apply to the use of Unicode characters in nicknames.
3.3. Visually Similar Characters

[I-D.ietf-precis-framework] describes some of the security considerations related to visually similar characters, also called "confusable characters" or "confusables".

Although the mapping rules under Section 2 are designed in part to reduce the possibility of confusion about nicknames, this document does not yet provide more detailed recommendations regarding the handling of visually similar characters, such as those in [UTR39]. However, a future version of this document might provide such recommendations.

4. IANA Considerations

The IANA shall add an entry to the PRECIS Usage Registry for reuse of the PRECIS FreeClass for preparation and comparision of nicknames, as follows:

Application Protocol: MSRP and XMPP.
Base Class: FreeClass
Subclassing: No.
Directionality: The "Bidi Rule" defined in RFC 5893 applies.
CaseMapping: None.
Normalization: NFC.
Specification: RFC XXXX.

5. References

5.1. Normative References

[I-D.ietf-precis-framework]

[I-D.ietf-simple-chat]


Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.


5.2. Informative References


Author’s Address

Peter Saint-Andre
Cisco Systems, Inc.
1899 Wynkoop Street, Suite 600
Denver, CO  80202
USA

Phone: +1-303-308-3282
Email: psaintan@cisco.com