

Network Working Group
Internet-Draft
Updates: 6120 (if approved)
Intended status: Standards Track
Expires: August 13, 2014

T. Alkemade
February 9, 2014

Validating Info/Query (IQ) stanzas in the Extensible Messaging and
Presence Protocol (XMPP)
draft-alkemade-xmpp-iq-validation-00

Abstract

This document provides security recommendations for the validation and generation of Info/Query (IQ) stanzas in the Extensible Messaging and Presence Protocol (XMPP). This document updates RFC 6120.

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 13, 2014.

Copyright Notice

Copyright (c) 2014 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
2. Terminology	3
3. Discussion Venue	3
4. Recommendations	3
4.1. No 'from'	3
4.2. Rejecting spoofed replies to queries	4
4.3. Preventing presence leaks	4
5. IANA Considerations	4
6. Security Considerations	5
7. Normative References	5
Author's Address	5

1. Introduction

The Extensible Messaging and Presence Protocol (XMPP) [RFC6120] uses Info/Query (IQ) stanzas as a "request-response" mechanism. The semantics of IQ enable an entity to make a request of, and receive a response from, another entity. The interaction is tracked by the requesting entity through use of the 'id' attribute. Thus, IQ interactions follow a common pattern of structured data exchange such as get/result or set/result (although an error can be returned in reply to a request if appropriate).

However, it was found not all implementations properly verify the origin of IQ responses. This document provides recommendations on how to avoid spoofed responses.

2. Terminology

Various security-related terms are to be understood in the sense defined in [RFC4949].

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 [RFC2119].

3. Discussion Venue

The discussion venue for this document is the mailing list of the XMPP Working Group, for which archives and subscription information can be found at <<https://www.ietf.org/mailman/listinfo/xmpp>>.

4. Recommendations

4.1. No 'from'

Section 8.1.2.1. of [RFC6120] specifies that IQ stanzas sent on behalf of the user's own account MUST either (a) include no 'from' attribute or (b) use the user's bare JID as 'from' address (i.e., <localpart@domainpart>). For compatibility with incorrect servers and servers still following [RFC3920], implementations MAY additionally accept a reply with either (a) a 'from' address equal to the full JID of the client (i.e., <localpart@domainpart/resourcepart>) or (b) a 'from' address equal to the domainpart of the JID of the account (i.e., <domainpart>).

4.2. Rejecting spoofed replies to queries

'id' attributes are used as end-to-end identifiers of stanzas: the same 'id' attribute is used across every hop. When delivered, the receiver will see the same 'id' value as the sender specified. As described in Section 8.1.3. of [RFC6120], it is REQUIRED to include 'id' values on IQ stanzas and RECOMMENDED for all other stanza types. The entity creating a stanza needs to ensure the 'id' values it generates are unique.

After sending an IQ stanza with type "get" or "set", an entity may store the 'id' and 'to' of the outgoing <iq/> element to identify the response. When an IQ stanza comes in with a matching 'id' and type "result" or "error", the entity MUST verify that the 'from' attribute on the <iq/> matches the 'to' of the outgoing stanza. If the 'from' and 'to' attributes do not match, the entity MUST ignore the stanza. As is mandatory in response to IQ stanzas of type "result" or "error", the entity MUST NOT return an error.

For queries where the intended recipient is the server acting on behalf of the user's own account entities MAY apply the exceptions in Section 4.1.

4.3. Preventing presence leaks

Many implementations use a counter to generate new 'id' attributes for stanzas to guarantee their uniqueness. However, this may leak presence information of the user: it can give an approximation of how long a client has been running and how many stanzas it has sent since a previous stanza. This could, for example, give an indication of how many messages a user has sent in a certain time. Therefore clients SHOULD NOT include a counter in the 'id' attribute.

Additionally, it is RECOMMENDED to use randomly or pseudo-randomly generated 'id' attributes. Implementations using [RFC4122] to generate Universally Unique Identifiers (UUIDs) to use as 'id' attributes MUST use version 4 UUIDs, which are randomly or pseudo-randomly generated and carry no identifying information. Implementations using a pseudo-random generator, either directly or for generating UUIDs, SHOULD make sure that future values are hard to predict. For more information see [RFC4086].

5. IANA Considerations

This document requests no actions of the IANA.

6. Security Considerations

This document covers security.

7. Normative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.
- [RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

Author's Address

Thijs Alkemade

Email: me@thijsalkema.de

DNS-Based Authentication of Named Entities (DANE)
Internet-Draft

Intended status: Standards Track

Expires: August 17, 2014

T. Finch
University of Cambridge

M. Miller

Cisco Systems, Inc.

P. Saint-Andre

&yet

February 13, 2014

Using DNS-Based Authentication of Named Entities (DANE) TLSA records
with SRV and MX records.
draft-ietf-dane-srv-05

Abstract

The DANE specification (RFC 6698) describes how to use TLSA resource records in the DNS to associate a server's host name with its TLS certificate. The association is secured with DNSSEC. Some application protocols use SRV records (RFC 2782) to indirectly name the server hosts for a service domain (SMTP uses MX records for the same purpose). This specification gives generic instructions for how these application protocols locate and use TLSA records when technologies such as SRV records are used. Separate documents give the details that are specific to particular application protocols.

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 17, 2014.

Copyright Notice

Copyright (c) 2014 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	3
3. Relation between SRV and MX records	3
4. DNS Checks for TLSA and SRV Records	4
4.1. SRV Query	4
4.2. TLSA Queries	5
5. TLS Checks for TLSA and SRV Records	6
6. Guidance for Application Protocols	7
7. Guidance for Server Operators	7
8. Internationalization Considerations	8
9. IANA Considerations	8
10. Security Considerations	8
10.1. Mixed Security Status	8
10.2. A Service Domain Trusts its Servers	8
10.3. Certificate Subject Name Matching	9
11. Acknowledgements	9
12. References	9
12.1. Normative References	9
12.2. Informative References	10
Appendix A. Mail Example	11
Appendix B. XMPP Example	11
Appendix C. Rationale	12
Authors' Addresses	13

1. Introduction

The base DANE specification [RFC6698] describes how to use TLSA resource records in the DNS to associate a server's host name with its TLS certificate. The association is secured using DNSSEC. That document "only relates to securely associating certificates for TLS and DTLS with host names" (see the last paragraph of section 1.2 of [RFC6698]).

Some application protocols do not use host names directly; instead, they use a service domain and the relevant host names are located indirectly via SRV records [RFC2782], or MX records in the case of

SMTP [RFC5321] (Note: in the "CertID" specification [RFC6125], the source domain and host name are referred to as the "source domain" and the "derived domain"). Because of this intermediate resolution step, the normal DANE rules specified in [RFC6698] do not directly apply to protocols that use SRV or MX records.

This document describes how to use DANE TLSA records with SRV and MX records. To summarize:

- o We rely on DNSSEC to secure the association between the service domain and the target server host names (i.e., the host names that are discovered by the SRV or MX query).
- o The TLSA records are located using the port, protocol, and target host name fields (not the service domain).
- o Clients always use TLS when connecting to servers with TLSA records.
- o Assuming that the association is secure, the server's certificate is expected to authenticate the target server host name, rather than the service domain.

Separate documents give the details that are specific to particular application protocols, such as SMTP [I-D.ietf-dane-smtp-with-dane] and XMPP [I-D.ietf-xmpp-dna].

2. Terminology

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

This draft uses the definitions for "secure", "insecure", "bogus", and "indeterminate" from [RFC4035]. This draft uses the acronyms from [I-D.ietf-dane-registry-acronyms] for the values of TLSA fields where appropriate.

3. Relation between SRV and MX records

For the purpose of this specification (to avoid cluttering the description with special cases) we treat each MX record ([RFC5321] section 5) as being equivalent to an SRV record [RFC2782] with corresponding fields copied from the MX record and the remaining fields having fixed values as follows:

Table 1: SRV Fields and MX Equivalents

DNS SRV Field	Equivalent MX Value
Service	smtp
Proto	tcp
Name	MX owner name (mail domain)
TTL	MX TTL
Class	MX Class
Priority	MX Priority
Weight	0
Port	25
Target	MX Target

Thus we can treat the following MX record as if it were the SRV record shown below:

```
example.com.          86400 IN MX  10      mx.example.net.
_smtptcp.example.com. 86400 IN SRV 10 0 25 mx.example.net.
```

Other details that are specific to SMTP are described in [I-D.ietf-dane-smtp-with-dane].

4. DNS Checks for TLSA and SRV Records

4.1. SRV Query

When the client makes an SRV query, a successful result will be a list of one or more SRV records (or possibly a chain of CNAME / DNAME aliases referring to such a list).

For this specification to apply, all of these DNS RRsets MUST be "secure" according to DNSSEC validation ([RFC4033] section 5). In the case of aliases, the whole chain of CNAME and DNAME RRsets MUST be secure as well. This corresponds to the AD bit being set in the response(s); see [RFC4035] section 3.2.3.

If they are not all secure, this protocol has not been correctly deployed. The client SHOULD fall back to its non-DNSSEC non-DANE behavior (this corresponds to the AD bit being unset).

If any of the responses are "bogus" or "indeterminate" according to DNSSEC validation, the client MUST abort (This usually corresponds to a "server failure" response).

In the successful case, the client now has an authentic list of server host names with weight and priority values. It performs server ordering and selection using the weight and priority values without regard to the presence or absence of DNSSEC or TLSA records. It takes note of the DNSSEC validation status of the SRV response for use when checking certificate names (see Section 5).

4.2. TLSA Queries

If the SRV response was insecure, the client MUST NOT perform any TLSA queries. If the SRV response is "secure" according to DNSSEC validation, the client performs a TLSA query for each SRV target as described in this section.

For each SRV target host name, the client performs DNSSEC validation on the address (A, AAAA) response and continues based on the results:

- o if the response is "insecure", the client MUST NOT perform a TLSA query for that target; the TLSA query will most likely fail.
- o If the response is "bogus" or "indeterminate", the client MUST NOT connect to this host name; instead it uses the next most appropriate SRV target.

The client SHALL construct the TLSA query name as described in [RFC6698] section 3, based on fields from the SRV record: the port from the SRV RDATA, the protocol from the SRV query name, and the TLSA base domain set to the SRV target host name.

For example, the following SRV record leads to the TLSA query shown below:

```
_imap._tcp.example.com. 86400 IN SRV 10 0 143 imap.example.net.
```

```
_143._tcp.imap.example.net. IN TLSA ?
```

The client SHALL determine if the TLSA record(s) are usable according to section 4.1 of [RFC6698]. This affects SRV handling as follows:

If the TLSA response is "secure", the client MUST use TLS when connecting to the server. The TLSA records are used when validating the server's certificate as described under Section 5.

If the TLSA response is "insecure", the client SHALL proceed as if this server has no TLSA records. It MAY connect to the server with or without TLS.

If the TLSA response is "bogus" or "indeterminate", then the client MUST NOT connect to this server (the client can still use other SRV targets).

5. TLS Checks for TLSA and SRV Records

When connecting to a server, the client MUST use TLS if the responses to the SRV and TLSA queries were "secure" as described above. If the client received zero usable TLSA certificate associations, it SHALL validate the server's TLS certificate using the normal PKIX rules [RFC5280] or protocol-specific rules (e.g., following [RFC6125]) without further input from the TLSA records. If the client received one or more usable TLSA certificate associations, it SHALL process them as described in [RFC6698] section 2.1.

If the TLS server's certificate -- or the public key of the server's certificate -- matches a usable TLSA record with Certificate Usage "DANE-EE", the client MUST consider the server to be authenticated. Because the information in such a TLSA record supersedes the non-key information in the certificate, all other [RFC5280] and [RFC6125] authentication checks (e.g., reference identifier, key usage, expiration, issuance, etc.) MUST be ignored or omitted.

Otherwise, the client uses the information in the server certificate and DNSSEC validation status of the SRV query in its authentication checks. It SHOULD use the Server Name Indication extension (TLS SNI) [RFC6066] or its functional equivalent in the relevant application protocol (e.g., in XMPP [RFC6120] this is the 'to' address of the initial stream header). The preferred name SHALL be chosen as follows, and the client SHALL verify the identity asserted by the server's certificate according to [RFC6125] section 6, using a list of reference identifiers constructed as follows (note again that in RFC 6125 the terms "source domain" and "derived domain" refer to the same things as "service domain" and "target host name" in this document).

SRV is insecure: The reference identifiers SHALL include the service domain and MUST NOT include the SRV target host name. The service domain is the preferred name for TLS SNI or its equivalent.

SRV is secure: The reference identifiers SHALL include both the service domain and the SRV target host name. The target host name is the preferred name for TLS SNI or its equivalent.

In the latter case, the client will accept either identity so that it is compatible with servers that do and do not support this specification.

6. Guidance for Application Protocols

Separate documents describe how to apply this specification to particular application protocols. Such documents ought to cover the following points:

- o Fallback logic in the event of bogus replies and the like.
- o The use of TLS SNI or its functional equivalent.
- o Appropriate mappings for non-SRV technologies such as MX.
- o Compatibility with clients that do not support SRV lookups.

7. Guidance for Server Operators

To conform to this specification, the published SRV records and subsequent address (A, AAAA) records MUST be secured with DNSSEC. There SHOULD also be at least one TLSA record published that authenticates the server's certificate.

When using TLSA records with Certificate Usage "DANE-EE", the deployed certificate does not need to contain any of the possible reference identifiers discussed below. Indeed, none of the certificate's information is necessary for such certificates. However, servers that rely solely on validation using Certificate Usage "DANE-EE" TLSA records might prevent clients that do not support this specification from successfully connecting with TLS.

For TLSA records with Certificate Usage types other than "DANE-EE", the certificate(s) MUST contain a reference identifier that matches:

- o the service domain name (the "source domain" in [RFC6125] terms, which is the SRV query domain); and/or

- o the server host name (the "derived domain" in [RFC6125] terms, which is the SRV target).

Servers that support multiple service domains (i.e., multi-tenant) can implement Server Name Indicator (TLS SNI) [RFC6066] or its functional equivalent to determine which certificate to offer. Clients that do not support this specification will indicate a preference for the service domain name, while clients that support this specification will indicate the server host name. However, the server determines what certificate to present in the TLS handshake; e.g., the presented certificate might only authenticate the server host name.

8. Internationalization Considerations

If any of the DNS queries are for an internationalized domain name, then they need to use the A-label form [RFC5890].

9. IANA Considerations

No IANA action is required.

10. Security Considerations

10.1. Mixed Security Status

We do not specify that clients checking all of a service domain's server host names are consistent in whether they have or do not have TLSA records. This is so that partial or incremental deployment does not break the service. Different levels of deployment are likely if a service domain has a third-party fallback server, for example.

The SRV and MX sorting rules are unchanged; in particular they have not been altered in order to prioritize secure servers over insecure servers. If a site wants to be secure it needs to deploy this protocol completely; a partial deployment is not secure and we make no special effort to support it.

10.2. A Service Domain Trusts its Servers

By signing their zone with DNSSEC, service domain operators implicitly instruct their clients to check their server TLSA records. This implies another point in the trust relationship between service domain holders and their server operators. Most of the setup requirements for this protocol fall on the server operator: installing a TLS certificate with the correct name (where necessary), and publishing a TLSA record for that certificate. If these are not correct then connections from TLSA-aware clients might fail.

10.3. Certificate Subject Name Matching

Section 4 of the TLSA specification [RFC6698] leaves the details of checking names in certificates to higher level application protocols, though it suggests the use of [RFC6125].

Name checks are not necessary if the matching TLSA record is of Certificate Usage "DANE-EE". Because such a record identifies the specific certificate (or public key of the certificate), additional checks are superfluous and potentially conflicting.

Otherwise, while DNSSEC provides a secure binding between the server name and the TLSA record, and the TLSA record provides a binding to a certificate, this latter step can be indirect via a chain of certificates. For example, a Certificate Usage "PKIX-TA" TLSA record only authenticates the CA that issued the certificate, and third parties can obtain certificates from the same CA. Therefore, clients need to check whether the server's certificate matches one of the expected reference identifiers to ensure the certificate was issued by the CA to the server the client expects.

11. Acknowledgements

Thanks to Mark Andrews for arguing that authenticating the server host name is the right thing, and that we ought to rely on DNSSEC to secure the SRV / MX lookup. Thanks to James Cloos, Viktor Dukhovni, Ned Freed, Olafur Gudmundsson, Paul Hoffman, Phil Pennock, Hector Santos, Jonas Schneider, and Alessandro Vesely for helpful suggestions.

12. References

12.1. Normative References

- [I-D.ietf-dane-registry-acronyms]
Gudmundsson, O., "Adding acronyms to simplify DANE conversations", draft-ietf-dane-registry-acronyms-03 (work in progress), January 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.

- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.
- [RFC4035] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Protocol Modifications for the DNS Security Extensions", RFC 4035, March 2005.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.
- [RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.
- [RFC6066] Eastlake, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, January 2011.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.

12.2. Informative References

- [I-D.ietf-dane-smtp-with-dane]
Dukhovni, V. and W. Hardaker, "SMTP security via opportunistic DANE TLS", draft-ietf-dane-smtp-with-dane-05 (work in progress), February 2014.
- [I-D.ietf-xmpp-dna]
Saint-Andre, P. and M. Miller, "Domain Name Associations (DNA) in the Extensible Messaging and Presence Protocol (XMPP)", draft-ietf-xmpp-dna-05 (work in progress), February 2014.

Appendix A. Mail Example

In the following, most of the DNS resource data is elided for simplicity.

```
; mail domain
example.com.      MX      1 mx.example.net.
example.com.      RRSIG   MX ...

; SMTP server host name
mx.example.net.   A       192.0.2.1
mx.example.net.   RRSIG   A ...

mx.example.net.   AAAA    2001:db8:212:8::e:1
mx.example.net.   RRSIG   ...

; TLSA resource record
_25._tcp.mx.example.net.  TLSA ...
_25._tcp.mx.example.net.  RRSIG TLSA ...
```

Mail for addresses at example.com is delivered by SMTP to mx.example.net. Connections to mx.example.net port 25 that use STARTTLS will get a server certificate that authenticates the name mx.example.net.

Appendix B. XMPP Example

In the following, most of the DNS resource data is elided for simplicity.

```
; XMPP domain
_xmpp-client.example.com. SRV      1 0 5222 im.example.net.
_xmpp-client.example.com. RRSIG    SRV ...

; XMPP server host name
im.example.net.   A       192.0.2.3
im.example.net.   RRSIG   A ...

im.example.net.   AAAA    2001:db8:212:8::e:4
im.example.net.   RRSIG   AAAA ...

; TLSA resource record
_5222._tcp.im.example.net.  TLSA ...
_5222._tcp.im.example.net.  RRSIG TLSA ...
```

XMPP sessions for addresses at example.com are established at im.example.net. Connections to im.example.net port 5222 that use STARTTLS will get a server certificate that authenticates the name im.example.net.

Appendix C. Rationale

The long-term goal of this specification is to settle on TLS certificates that verify the server host name rather than the service domain, since this is more convenient for servers hosting multiple domains (so-called "multi-tenanted environments") and scales up more easily to larger numbers of service domains.

There are a number of other reasons for doing it this way:

- o The certificate is part of the server configuration, so it makes sense to associate it with the server host name rather than the service domain.
- o In the absence of TLS SNI, if the certificate identifies the host name then it does not need to list all the possible service domains.
- o When the server certificate is replaced it is much easier if there is one part of the DNS that needs updating to match, instead of an unbounded number of hosted service domains.
- o The same TLSA records work with this specification, and with direct connections to the host name in the style of [RFC6698].
- o Some application protocols, such as SMTP, allow a client to perform transactions with multiple service domains in the same connection. It is not in general feasible for the client to specify the service domain using TLS SNI when the connection is established, and the server might not be able to present a certificate that authenticates all possible service domains.

- o It is common for SMTP servers to act in multiple roles, for example as outgoing relays or as incoming MX servers, depending on the client identity. It is simpler if the server can present the same certificate regardless of the role in which it is to act. Sometimes the server does not know its role until the client has authenticated, which usually occurs after TLS has been established.

This specification does not provide an option to put TLSA records under the service domain because that would add complexity without providing any benefit, and security protocols are best kept simple. As described above, there are real-world cases where authenticating the service domain cannot be made to work, so there would be complicated criteria for when service domain TLSA records might be used and when they cannot. This is all avoided by putting the TLSA records under the server host name.

The disadvantage is that clients which do not do DNSSEC validation must, according to [RFC6125] rules, check the server certificate against the service domain, since they have no other way to authenticate the server. This means that SNI support or its functional equivalent is necessary for backward compatibility.

Authors' Addresses

Tony Finch
University of Cambridge Computing Service
New Museums Site
Pembroke Street
Cambridge CB2 3QH
ENGLAND

Phone: +44 797 040 1426
Email: dot@dotat.at
URI: <http://dotat.at/>

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

Email: mamille2@cisco.com

Peter Saint-Andre
&yet

Email: ietf@stpeter.im

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 8, 2014

P. Saint-Andre
&yet
M. Miller
Cisco Systems, Inc.
February 4, 2014

Domain Name Associations (DNA) in the Extensible Messaging and Presence
Protocol (XMPP)
draft-ietf-xmpp-dna-05

Abstract

This document improves the security of the Extensible Messaging and Presence Protocol (XMPP) in two ways. First, it specifies how "prooftypes" can establish a strong association between a domain name and an XML stream. Second, it describes how to securely delegate a source domain to a derived domain, which is especially important in virtual hosting environments.

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 8, 2014.

Copyright Notice

Copyright (c) 2014 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	3
3. Flow Chart	3
4. A Simple Scenario	6
5. One-Way Authentication	7
6. Piggybacking	8
6.1. Assertion	8
6.2. Supposition	9
7. Alternative Proofypes	10
7.1. DANE	11
7.2. POSH	11
8. Secure Delegation and Multi-Tenancy	12
9. Proofype Model	13
10. IANA Considerations	13
10.1. Well-Known URI for xmpp-client Service	13
10.2. Well-Known URI for xmpp-server Service	13
11. Security Considerations	14
12. References	14
12.1. Normative References	14
12.2. Informative References	15
Authors' Addresses	15

1. Introduction

The need to establish a strong association between a domain name and an XML stream arises in both client-to-server and server-to-server communication using the Extensible Messaging and Presence Protocol (XMPP) [RFC6120]. Because XMPP servers are typically identified by DNS domain names, a client or peer server needs to verify the identity of a server to which it connects.

To date, such verification has been established based on information obtained from the Domain Name System (DNS), the Public Key Infrastructure (PKI), or similar sources. In relation to such associations, this document does the following:

1. Generalizes the model currently in use so that additional proofypes can be defined
2. Provides a basis for modernizing some proofypes to reflect progress in underlying technologies such as DNS Security [RFC4033]

3. Describes the flow of operations for establishing a domain name association (DNA)

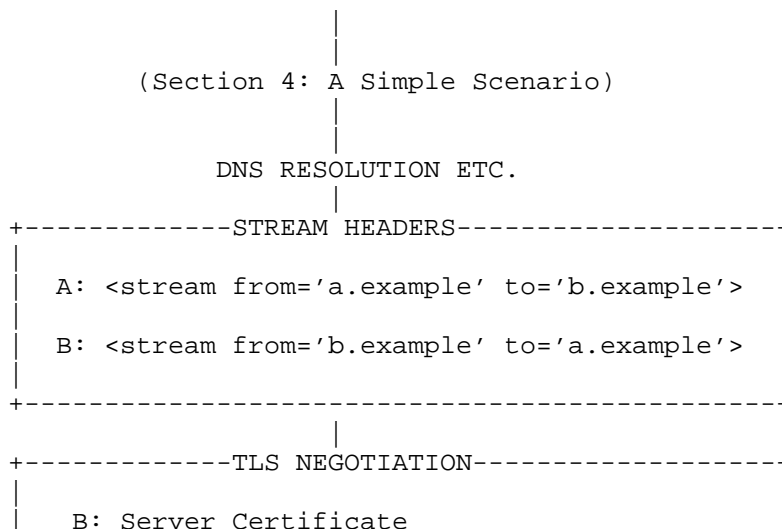
This document also provides guidelines for secure delegation. The need for secure delegation arises because the process for resolving the domain name of an XMPP service into the IP address at which an XML stream will be negotiated (defined in [RFC6120]) can involve delegation of a source domain (say, example.com) to a derived domain (say, hosting.example.net) using technologies such as DNS SRV records [RFC2782]. If such delegation is not done in a secure manner, then the domain name association cannot be authenticated.

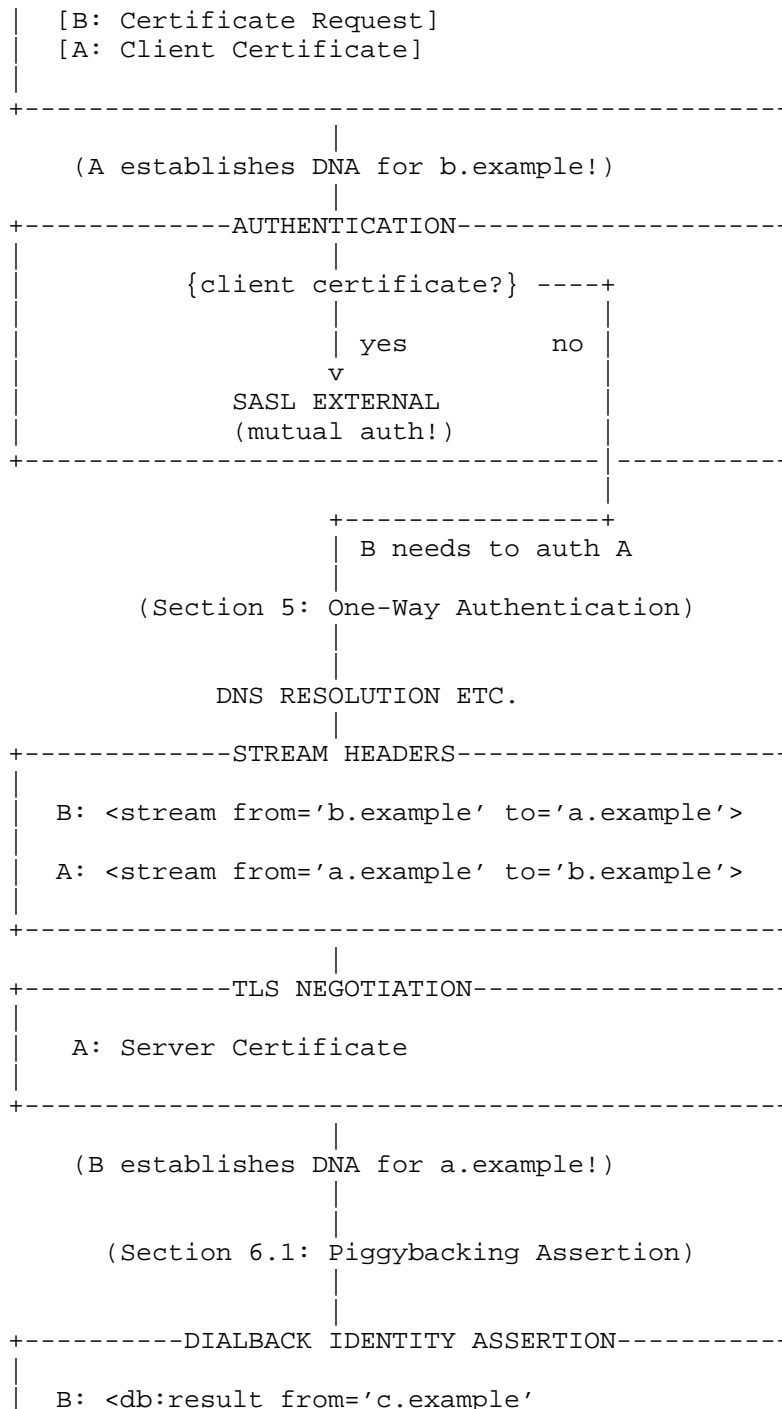
2. Terminology

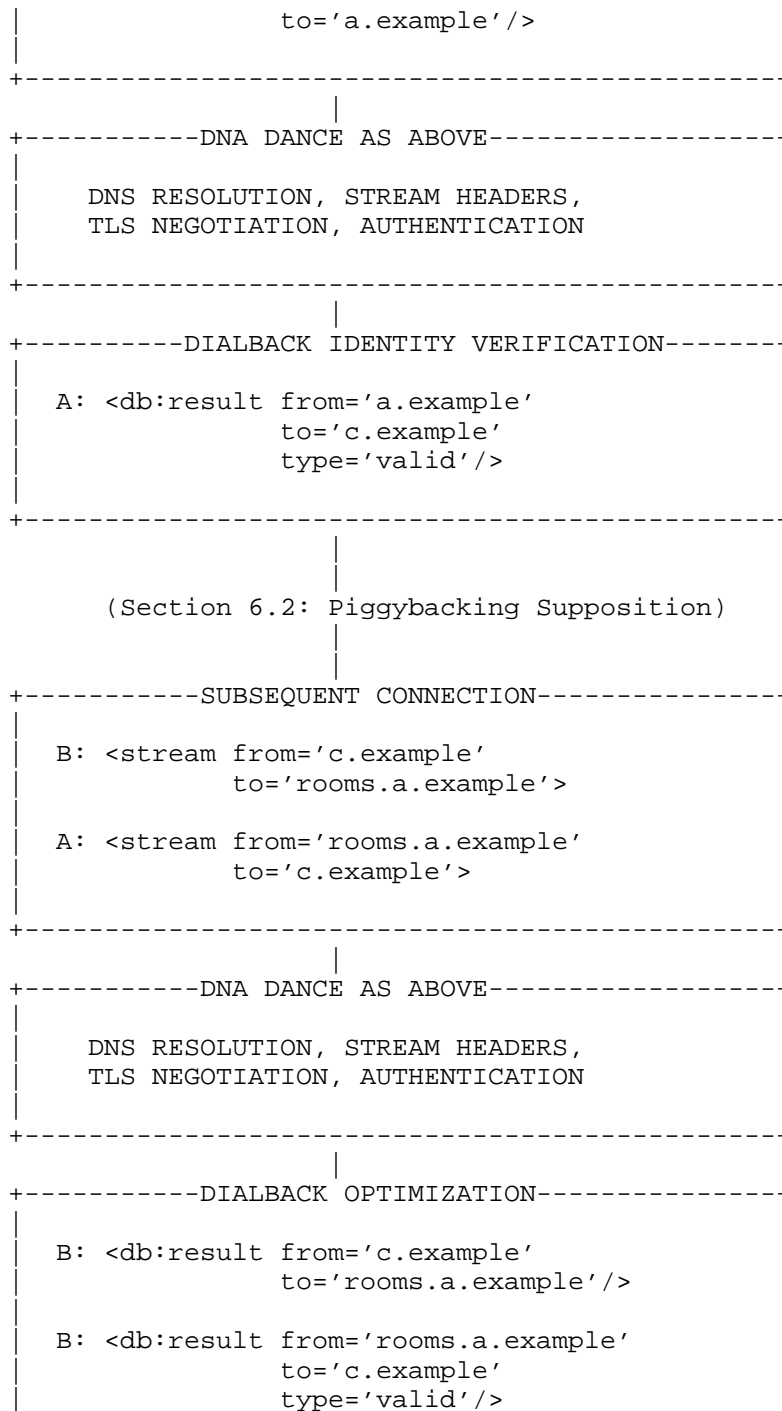
This document inherits XMPP terminology from [RFC6120] and [XEP-0220], DNS terminology from [RFC1034], [RFC1035], [RFC2782] and [RFC4033], and security terminology from [RFC4949] and [RFC5280]. The terms "source domain", "derived domain", "reference identity", and "presented identity" are used as defined in the "CertID" specification [RFC6125]. The terms "permissive federation", "verified federation", and "encrypted federation" are derived from [XEP-0238], although we substitute the term "authenticated federation" for the term "trusted federation" from that document.

3. Flow Chart

The following flow chart illustrates the protocol flow for establishing domain name associations between Server 1 and Server 2, as described in the remaining sections of this document.







```
|
+-----+
```

4. A Simple Scenario

To illustrate the problem, consider the simplified order of events (see [RFC6120] for details) in establishing an XML stream between Server 1 (a.example) and Server 2 (b.example):

1. Server 1 resolves the DNS domain name b.example.
2. Server 1 opens a TCP connection to the resolved IP address.
3. Server 1 sends an initial stream header to Server 2, asserting that it is a.example:

`<stream:stream from='a.example' to='b.example'>`
4. Server 2 sends a response stream header to Server 1, asserting that it is b.example:

`<stream:stream from='b.example' to='a.example'>`
5. The servers attempt TLS negotiation, during which Server 2 (acting as a TLS server) presents a PKIX certificate proving that it is b.example and Server 1 (acting as a TLS client) presents a PKIX certificate proving that it is a.example.
6. Server 1 checks the PKIX certificate that Server 2 provided and Server 2 checks the PKIX certificate that Server 1 provided; if these proofs are consistent with the XMPP profile of the matching rules from [RFC6125], each server accepts that there is a strong domain name association between its stream to the other party and the DNS domain name of the other party.

Several simplifying assumptions underlie the happy scenario just outlined:

- o Server 1 presents a PKIX certificate during TLS negotiation, which enables the parties to complete mutual authentication.

- o There are no additional domains associated with Server 1 and Server 2 (say, a subdomain rooms.a.example on Server 1 or a second domain c.example on Server 2).
- o The server administrators are able to obtain PKIX certificates in the first place.
- o The server administrators are running their own XMPP servers, rather than using hosting services.

Let's consider each of these "wrinkles" in turn.

5. One-Way Authentication

If Server 1 does not present its PKIX certificate during TLS negotiation (perhaps because it wishes to verify the identity of Server 2 before presenting its own credentials), Server 2 is unable to mutually authenticate Server 1. Therefore, Server 2 needs to negotiate and authenticate a stream to Server 1, just as Server 1 has done:

1. Server 2 resolves the DNS domain name a.example.
2. Server 2 opens a TCP connection to the resolved IP address.
3. Server 2 sends an initial stream header to Server 1, asserting that it is b.example:

`<stream:stream from='b.example' to='a.example'>`
4. Server 1 sends a response stream header to Server 2, asserting that it is a.example:

`<stream:stream from='a.example' to='b.example'>`
5. The servers attempt TLS negotiation, during which Server 1 (acting as a TLS server) presents a PKIX certificate proving that it is a.example.
6. Server 2 checks the PKIX certificate that Server 1 provided; if it is consistent with the XMPP profile [RFC6120] of the matching

rules from [RFC6125], Server 2 accepts that there is a strong domain name association between its stream to Server 1 and the DNS domain name a.example.

At this point the servers are using two TCP connections instead of one, which is somewhat wasteful. However, there are ways to tie the authentication achieved on the second TCP connection to the first TCP connection; see [XEP-0288] for further discussion.

6. Piggybacking

6.1. Assertion

Consider the common scenario in which Server 2 hosts not only b.example but also a second domain c.example (a "multi-tenanted" environment). If a user of Server 2 associated with c.example wishes to communicate with a friend at a.example, Server 2 needs to send XMPP stanzas from the domain c.example rather than b.example. Although Server 2 could open a new TCP connection and negotiate new XML streams for the domain pair of c.example and a.example, that too is wasteful. Server 2 already has a connection to a.example, so how can it assert that it would like to add a new domain pair to the existing connection?

The traditional method for doing so is the Server Dialback protocol, first specified in [RFC3920] and since moved to [XEP-0220]. Here, Server 2 can send a <db:result/> element for the new domain pair over the existing stream.

```
<db:result from='c.example' to='a.example'>  
  some-dialback-key  
</db:result>
```

This element functions as Server 2's assertion that it is (also) c.example, and thus is functionally equivalent to the 'from' address of an initial stream header as previously described.

In response to this assertion, Server 1 needs to obtain some kind of proof that Server 2 really is also c.example. It can do the same thing that it did before:

1. Server 1 resolves the DNS domain name c.example.
2. Server 1 opens a TCP connection to the resolved IP address (which might be the same IP address as for b.example).

3. Server 1 sends an initial stream header to Server 2, asserting that it is a.example:

```
<stream:stream from='a.example' to='c.example'>
```

4. Server 2 sends a response stream header to Server 1, asserting that it is c.example:

```
<stream:stream from='c.example' to='a.example'>
```

5. The servers attempt TLS negotiation, during which Server 2 (acting as a TLS server) presents a PKIX certificate proving that it is c.example.
6. Server 1 checks the PKIX certificate that Server 2 provided; if it is consistent with the XMPP profile [RFC6120] of the matching rules from [RFC6125], Server 1 accepts that there is a strong domain name association between its stream to Server 2 and the DNS domain name c.example.

Now that Server 1 accepts the domain name association, it informs Server 2 of that fact:

```
<db:result from='a.example' to='c.example' type='valid'>
```

The parties can then terminate the second connection, since it was used only for Server 1 to associate a stream over the same IP:port combination with the domain name c.example (the dialback key links the original stream to the new association).

6.2. Supposition

Piggybacking can also occur in the other direction. Consider the common scenario in which Server 1 provides XMPP services not only for a.example but also for a subdomain such as a groupchat service at rooms.a.example (see [XEP-0045]). If a user from c.example at Server 2 wishes to join a room on the groupchat service, Server 2 needs to send XMPP stanzas from the domain c.example to the domain rooms.a.example rather than a.example. Therefore, Server 2 needs to negotiate and authenticate a stream to rooms.a.example:

1. Server 2 resolves the DNS domain name rooms.a.example.

2. Server 2 opens a TCP connection to the resolved IP address.
3. Server 2 sends an initial stream header to Server 1 acting as rooms.a.example, asserting that it is b.example:

```
<stream:stream from='b.example' to='rooms.a.example'>
```
4. Server 1 sends a response stream header to Server 2, asserting that it is rooms.a.example:

```
<stream:stream from='rooms.a.example' to='b.example'>
```
5. The servers attempt TLS negotiation, during which Server 1 (acting as a TLS server) presents a PKIX certificate proving that it is rooms.a.example.
6. Server 2 checks the PKIX certificate that Server 1 provided; if it is consistent with the XMPP profile [RFC6120] of the matching rules from [RFC6125], Server 2 accepts that there is a strong domain name association between its stream to Server 1 and the DNS domain name rooms.a.example.

As before, the parties now have two TCP connections open. So that they can close the now-redundant connection, Server 2 sends a dialback key to Server 1 over the new connection.

```
<db:result from='c.example' to='rooms.a.example'>  
  some-dialback-key  
</db:result>
```

Server 1 then informs Server 2 that it accepts the domain name association:

```
<db:result from='rooms.a.example' to='c.example' type='valid' />
```

Server 2 can now close the connection over which it tested the domain name association for rooms.a.example.

7. Alternative Proofatypes

The foregoing protocol flows assumed that domain name associations were proved using the standard PKI prooftype specified in [RFC6120]:

that is, the server's proof consists of a PKIX certificate that is checked according to the XMPP profile [RFC6120] of the matching rules from [RFC6125], the client's verification material is obtained out of band in the form of a trusted root, and secure DNS is not necessary.

However, sometimes XMPP server administrators are unable or unwilling to obtain valid PKIX certificates for their servers. As one example, a certificate authority (CA) might try to send email messages to authoritative mailbox names [RFC2142], but the administrator of a subsidiary service such as `im.cs.podunk.example` can't receive email sent to `mailto:hostmaster@podunk.example`. As another example, a hosting provider such as `hosting.example.net` might not want to take on the liability of holding the certificate and private key for a tenant such as `example.com` (or the tenant might not want the hosting provider to hold its certificate and private key). In these circumstances, prooftypes other than PKIX are desirable. As described below, two alternatives have been defined so far: DNS-Based Authentication of Named Entities (DANE) and and PKIX Over Secure HTTP (POSH).

7.1. DANE

In the DANE prooftype, the server's proof consists of a PKIX certificate that is compared as an exact match or a hash of either the `SubjectPublicKeyInfo` or the full certificate, and the client's verification material is obtained via secure DNS.

The DANE prooftype makes use of the DNS-Based Authentication of Named Entities [RFC6698], specifically the use of DANE with DNS SRV records [I-D.ietf-dane-srv]. For XMPP purposes, the following rules apply:

- o If there is no SRV resource record, pursue the fallback methods described in [RFC6120].
- o Use the 'to' address of the initial stream header to determine the domain name of the TLS client's reference identifier (since use of the TLS Server Name Indication is purely discretionary in XMPP, as mentioned in [RFC6120]).

7.2. POSH

In the POSH prooftype, the server's proof consists of a PKIX certificate that is checked according to the rules from [RFC6120] and [RFC6125], the client's verification material is obtained by retrieving the PKIX certificate over HTTPS at a well-known URI [RFC5785], and secure DNS is not necessary since the HTTPS retrieval mechanism relies on the chain of trust from the public key infrastructure.

POSH is defined in [I-D.ietf-xmpp-posh]. For XMPP purposes, the well-known URIs [RFC5785] to be used are:

- o `"/.well-known/posh._xmpp-client._tcp.json"` for client-to-server connections
- o `"/.well-known/posh._xmpp-server._tcp.json"` for server-to-server connections

8. Secure Delegation and Multi-Tenancy

One common method for deploying XMPP services is multi-tenancy or virtual hosting: e.g., the XMPP service for `example.com` is actually hosted at `hosting.example.net`. Such an arrangement is relatively convenient in XMPP given the use of DNS SRV records [RFC2782], such as the following pointer from `example.com` to `hosting.example.net`:

```
_xmpp-server._tcp.example.com. 0 IN SRV 0 0 5269 hosting.example.net
```

Secure connections with multi-tenancy can work using the PKIX prooftype on a small scale if the provider itself wishes to host several domains (e.g., several related domains such as `jabber-de.example` and `jabber-ch.example`). However, in practice the security of multi-tenancy has been found to be unwieldy when the provider hosts large numbers of XMPP services on behalf of multiple tenants. Typically there are two main reasons for this state of affairs: the service provider (say, `hosting.example.net`) wishes to limit its liability and therefore does not wish to hold the certificate and private key for the tenant (say, `example.com`) and the tenant wishes to improve the security of the service and therefore does not wish to share its certificate and private key with service provider. As a result, server-to-server communications to `example.com` go unencrypted or the communications are TLS-encrypted but the certificates are not checked (which is functionally equivalent to a connection using an anonymous key exchange). This is also true of client-to-server communications, forcing end users to override certificate warnings or configure their clients to accept certificates for `hosting.example.net` instead of `example.com`. The fundamental problem here is that if DNSSEC is not used then the act of delegation via DNS SRV records is inherently insecure.

The specification for use of SRV and MX records with DANE [I-D.ietf-dane-srv] explains how to use DNSSEC for secure delegation with the DANE prooftype, and the POSH specification [I-D.ietf-xmpp-posh] explains how to use HTTPS redirects for secure delegation with the POSH prooftype.

9. Prooftype Model

In general, a domain name association (DNA) prooftype conforms to the following definition:

prooftype: A mechanism for proving an association between a domain name and an XML stream, where the mechanism defines (1) the nature of the server's proof, (2) the matching rules for comparing the client's verification material against the server's proof, (3) how the client obtains its verification material, and (4) whether the mechanism depends on secure DNS.

The PKI, DANE, and POSH prooftypes adhere to this model. In addition, other prooftypes are possible (examples might include PGP keys rather than PKIX certificates, or a token mechanism such as Kerberos or OAuth).

Some prooftypes depend on (or are enhanced by) secure DNS and thus also need to describe how they ensure secure delegation.

10. IANA Considerations

The POSH specification [I-D.ietf-xmpp-posh] provides guidelines for registering the well-known URIs [RFC5785] of protocols that make use of POSH. This specification registers two such URIs, for which the completed registration templates follow.

10.1. Well-Known URI for xmpp-client Service

This specification registers the well-known URI "posh._xmpp-client._tcp.json" in the Well-Known URI Registry as defined by [RFC5785].

URI suffix: posh._xmpp-client._tcp.json

Change controller: IETF

Specification document(s): [[this document]]

10.2. Well-Known URI for xmpp-server Service

This specification registers the well-known URI "posh._xmpp-server._tcp.json" in the Well-Known URI Registry as defined by [RFC5785].

URI suffix: posh._xmpp-server._tcp.json

Change controller: IETF

Specification document(s): [[this document]]

11. Security Considerations

This document supplements but does not supersede the security considerations of [RFC6120] and [RFC6125]. Relevant security considerations can also be found in [I-D.ietf-dane-srv] and [I-D.ietf-xmpp-posh].

12. References

12.1. Normative References

- [I-D.ietf-dane-srv]
Finch, T., Miller, M., and P. Saint-Andre, "Using DNS-Based Authentication of Named Entities (DANE) TLSA records with SRV and MX records.", draft-ietf-dane-srv-03 (work in progress), December 2013.
- [I-D.ietf-xmpp-posh]
Miller, M. and P. Saint-Andre, "PKIX over Secure HTTP (POSH)", draft-ietf-xmpp-posh-00 (work in progress), February 2014.
- [RFC1034] Mockapetris, P., "Domain names - concepts and facilities", STD 13, RFC 1034, November 1987.
- [RFC1035] Mockapetris, P., "Domain names - implementation and specification", STD 13, RFC 1035, November 1987.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, May 2005.
- [RFC4949] Shirey, R., "Internet Security Glossary, Version 2", RFC 4949, August 2007.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.

- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.
- [XEP-0220] Miller, J., Saint-Andre, P., and P. Hancke, "Server Dialback", XSF XEP 0220, September 2013.

12.2. Informative References

- [RFC2142] Crocker, D., "MAILBOX NAMES FOR COMMON SERVICES, ROLES AND FUNCTIONS", RFC 2142, May 1997.
- [RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.
- [XEP-0045] Saint-Andre, P., "Multi-User Chat", XSF XEP 0045, February 2012.
- [XEP-0238] Saint-Andre, P., "XMPP Protocol Flows for Inter-Domain Federation", XSF XEP 0238, March 2008.
- [XEP-0288] Hancke, P. and D. Cridland, "Bidirectional Server-to-Server Connections", XSF XEP 0288, September 2013.

Authors' Addresses

Peter Saint-Andre
&yet

Email: ietf@stpeter.im

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

Email: mamille2@cisco.com

XMPP Working Group
Internet-Draft
Intended status: Standards Track
Expires: August 8, 2014

M. Miller
Cisco Systems, Inc.
P. Saint-Andre
&yet
February 4, 2014

PKIX over Secure HTTP (POSH)
draft-ietf-xmpp-posh-00

Abstract

Experience has shown that it is extremely difficult to deploy proper PKIX certificates for TLS in multi-tenanted environments, since certification authorities will not issue certificates for hosted domains to hosting services, hosted domains do not want hosting services to hold their private keys, and hosting services wish to avoid liability for holding those keys. As a result, domains hosted in multi-tenanted environments often deploy non-HTTP applications such as email and instant messaging using certificates that identify the hosting service, not the hosted domain. Such deployments force end users and peer services to accept a certificate with an improper identifier, resulting in obvious security implications. This document defines two methods that make it easier to deploy certificates for proper server identity checking in non-HTTP application protocols. The first method enables the TLS client associated with a user agent or peer application server to obtain the end-entity certificate of a hosted domain over secure HTTP as an alternative to standard PKIX techniques. The second method enables a hosted domain to securely delegate a non-HTTP application to a hosting service using redirects provided by HTTPS itself or by a pointer in a file served over HTTPS at the hosted domain. While this approach is developed for use in the Extensible Messaging and Presence Protocol (XMPP) as a Domain Name Association proofotype, it can be applied to any non-HTTP application protocol.

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 8, 2014.

Copyright Notice

Copyright (c) 2014 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. Discussion Venue	4
3. Terminology	4
4. Obtaining Verification Materials	4
4.1. Source Domain Possesses PKIX Certificate	6
4.2. Source Domain References PKIX Certificate	7
4.3. Performing Verification	8
5. Secure Delegation	9
6. Order of Operations	9
7. Caching Results	10
8. Alternates and Roll-over	11
9. IANA Considerations	12
10. Security Considerations	12
11. References	13
11.1. Normative References	13
11.2. Informative References	14
Appendix A. Acknowledgements	15
Authors' Addresses	15

1. Introduction

We start with a thought experiment.

Imagine that you work on the operations team of a hosting company that provides the "foo" service (or email or instant messaging or social networking service) for ten thousand different customer

organizations. Each customer wants their service to be identified by the customer's domain name (e.g., foo.example.com), not the hosting company's domain name (e.g., hosting.example.net).

In order to properly secure each customer's "foo" service via Transport Layer Security (TLS) [RFC5246], you need to obtain PKIX certificates [RFC5280] containing identifiers such as foo.example.com, as explained in the "CertID" specification [RFC6125]. Unfortunately, you can't obtain such certificates because:

- o Certification authorities won't issue such certificates to you because you work for the hosting company, not the customer organization.
- o Customers won't obtain such certificates and then give them (plus the associated private keys) to you because their legal department is worried about liability.
- o You don't want to install such certificates (plus the associated private keys) on your servers anyway because your legal department is worried about liability, too.

Given your inability to deploy public keys / certificates containing the right identifiers, your back-up approach was always to use a certificate containing hosting.example.net as the identifier. However, more and more customers and end users are complaining about warning messages in user agents and the inherent security issues involved with taking a "leap of faith" to accept the identity mismatch between the source domain (foo.example.com) and the delegated domain (hosting.example.net).

This situation is both insecure and unsustainable. You have investigated the possibility of using DNS Security [RFC4033] and DNS-Based Authentication of Named Entities (DANE) [RFC6698] to solve the problem. However, your customers and your operations team have told you that they will not be able to deploy DNSSEC and DANE for several years at least. The product managers in your company are pushing you to find a method that can be deployed more quickly to overcome the lack of proper server identity checking for your hosted customers.

One possible approach is to ask each customer to provide the public key / certificate for the "foo" service at a special HTTPS URI on their website ("https://foo.example.com/.well-known/posh.foo.json" is one possibility). This could be a public key that you generate for the customer, but because the customer hosts it via HTTPS, any user

agent can find that public key and check it against the public key you provide during TLS negotiation for the "foo" service (as one added benefit, the customer never needs to hand you a private key). Alternatively, the customer can redirect requests for that special HTTPS URI to an HTTPS URI at your own website, thus making it explicit that they have delegated the "foo" service to you.

The approach sketched out above, called POSH ("PKIX Over Secure HTTP"), is explained in the remainder of this document. While this approach is developed for use in the Extensible Messaging and Presence Protocol (XMPP) as a proofotype for Domain Name Associations (DNA) [XMPP-DNA], it can be applied to any non-HTTP application protocol.

2. Discussion Venue

The discussion venue for this document is the posh@ietf.org mailing list; visit <https://www.ietf.org/mailman/listinfo/posh> for subscription information and discussion archives.

3. Terminology

This document inherits security terminology from [RFC5280]. The terms "source domain", "derived domain", "reference identifier", and "presented identifier" are used as defined in the "CertID" specification [RFC6125].

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 [RFC2119].

4. Obtaining Verification Materials

Server identity checking (see [RFC6125]) involves three different aspects:

1. A proof of the TLS server's identity (in PKIX, this takes the form of a PKIX certificate [RFC5280]).
2. Rules for checking the certificate (which vary by application protocol, although [RFC6125] attempts to harmonize those rules).
3. The materials that a TLS client uses to verify the TLS server's identity or check the TLS server's proof (in PKIX, this takes the form of chaining the end-entity certificate back to a trusted

root and performing all validity checks as described in [RFC5280], [RFC6125], and the relevant application protocol specification).

When POSH is used, the first two aspects remain the same: the TLS server proves its identity by presenting a PKIX certificate [RFC5280] and the certificate is checked according to the rules defined in the appropriate application protocol specification (such as [RFC6120] for XMPP). However, the TLS client obtains the materials it will use to verify the server's proof by retrieving a JSON Web Key (JWK) set [JOSE-JWK] over HTTPS ([RFC2616] and [RFC2818]) from a well-known URI [RFC5785].

The process for retrieving a PKIX certificate over secure HTTP is as follows.

1. The TLS client performs an HTTPS GET at the source domain to the path `"/.well-known/posh.{servicedesc}.json"`. The value of `"{servicedesc}"` is application-specific; see Section 9 of this document for more details. For example, if the application protocol is some hypothetical "Foo" service, then `"{servicedesc}"` could be `"foo"`; thus if a Foo client were to use POSH to verify a Foo server for the domain `"foo.example.com"`, the HTTPS GET request would be as follows:

```
GET /.well-known/posh.foo.json HTTP/1.1
Host: foo.example.com
```

2. The source domain HTTPS server responds in one of three ways:
 - * If it possesses a PKIX certificate for the requested path, it responds as detailed in Section 4.1.
 - * If it has a reference to where the PKIX certificate can be obtained, it responds as detailed in Section 4.2.
 - * If it does not have any PKIX certificate for the requested path, it responds with a client error status code (e.g., 404).

4.1. Source Domain Possesses PKIX Certificate

If the source domain HTTPS server possesses the certificate information, it responds to the HTTPS GET with a success status code and the message body set to a JSON Web Key (JWK) set [JOSE-JWK]. The JWK set MUST contain at least one JWK object, and MUST contain an "expires" field whose value is the number of seconds after which the TLS client ought to consider the key information to be stale (further explained under Section 7).

Each included JWK object MUST possess the following information:

- o The "kty" field set to the appropriate key type used for TLS connections (e.g., "RSA" for a certificate using an RSA key).
- o The required public parameters for the key type (e.g., "n" and "e" for a certificate using an RSA key).
- o The "x5t" field set to the certificate thumbprint, as described in section 3.6 of [JOSE-JWK].

Each JWK object MUST NOT possess the private parameters for the key type (e.g., "d", "p", "q" for a certificate using an RSA key).

Each JWK object MAY possess other parameters as desired by application servers (e.g., the "x5c" field containing the entire X.509 certificate chain, as per section 3.7 of [JOSE-JWK]).

The following example illustrates the usage described above.

Example Content Response

```
HTTP/1.1 200 OK
Content-Type: application/jwk-set+json
Content-Length: 2785
```

```
{
  "keys": [
    {
      "kty": "RSA",
      "kid": "c8fb8b80-1193-11e3-b2b1-835742119fe8",
      "n": "ANxwssdcU3LbODErec3owrwUhlzjtuskAn8rAcBMRPImn5xA
        JRX-1T5g2D7MTozWWFk4TlpgzAR5slvM0tc35qAI9I0Cqk4Z
        LChQrYsWuY7alTrnNXdusHUYc6Eq89DZaH2knTcp57wAXzJP
        IG_tpBi5F7ck9LVRvRjybix0HJ7i4YrL-GeLuSgrj04-GDcX
        Ip8oV0FMKZH-NoMfUITlWYl_JcXlD0WUAiuAnvWtD4Kh_qMJ
        U6FZuupZGHqPdc3vrXtp27LWgxzxjFa9qnOU6y53vCCJXLLI
        5sy2fCwEDzLJqh2T6UitIzjrSUZMIsK8r2pXkroI0uYuNn3W
        y-jAzK8",
      "e": "AQAB",
      "x5t": "UpjRI_A3afKE8_AIeTZ5oldECTY"
    }
  ],
  "expires": 604800
}
```

The "expires" value is a hint regarding the expiration of the keying materials. If no "expires" field is included, a TLS client SHOULD consider these verification materials invalid. See Section 7 for how to reconcile this "expires" field with the reference's "expires" field.

4.2. Source Domain References PKIX Certificate

If the source domain HTTPS server has a reference to the certificate information, it responds to the HTTPS GET with a JSON document. The document MUST contain a "url" field whose value is the HTTPS URL where TLS clients can obtain the actual JWK set, and MUST contain an "expires" field whose value is the number of seconds after which the TLS client ought to consider the delegation to be stale (further explained under Section 7).

Example Reference Response

```
HTTP/1.1 200 Ok
Content-Type: application/json
Content-Length: 78
```

```
{
  "url": "https://hosting.example.net/.well-known/posh.foo.json",
  "expires": 86400
}
```

The client performs an HTTPS GET for the URL specified in the "url" field value. The HTTPS server for the URI to which the client has been redirected responds to the request with a JWK set. The content retrieved from the "url" location MUST NOT itself be a reference (i.e., containing a "url" fields instead of a "keys" field), in order to prevent circular delegations.

Note: The JSON document returned by the source domain HTTPS server MUST contain either a reference or a JWK-set, but MUST NOT contain both.

Note: See Section 10 for discussion about HTTPS redirects.

The "expires" value is a hint regarding the expiration of the source domain's delegation of service to the delegated domain. If no "expires" field is included, a TLS client SHOULD consider the delegation invalid. See Section 7 for guidelines about reconciling this "expires" field with the JWK-set's "expires" field.

4.3. Performing Verification

The TLS client compares the PKIX information obtained from the TLS server against each JWK object in the POSH results, until a match is found or the collection of POSH verification materials is exhausted. If none of the JWK objects match the TLS server PKIX information, the TLS client SHOULD reject the connection (the TLS client might still accept the connection if other verification schemes are successful).

The TLS client SHOULD compare the fingerprint of the PKIX certificate from the TLS server against the "x5t" field of the JWK object (note the "x5t" field is the base64url encoding of the fingerprint).

The TLS client MAY verify the certificate chain provided in the "x5c" field of the JWK object (if present), but it MUST NOT implicitly consider the final certificate in the "x5c" field to be a trust anchor itself; the TLS client only uses the end entity certificate information for verification.

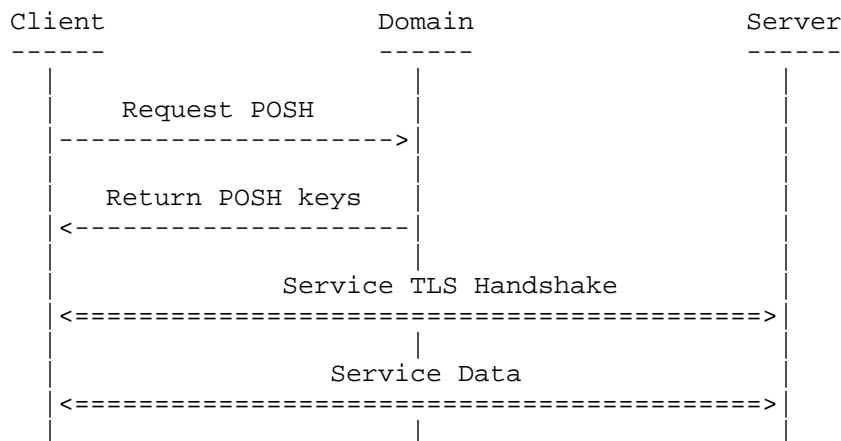
5. Secure Delegation

The delegation from the source domain to the delegated domain can be considered secure if the certificate offered by the TLS server matches the POSH certificate, regardless of how the POSH certificates are obtained.

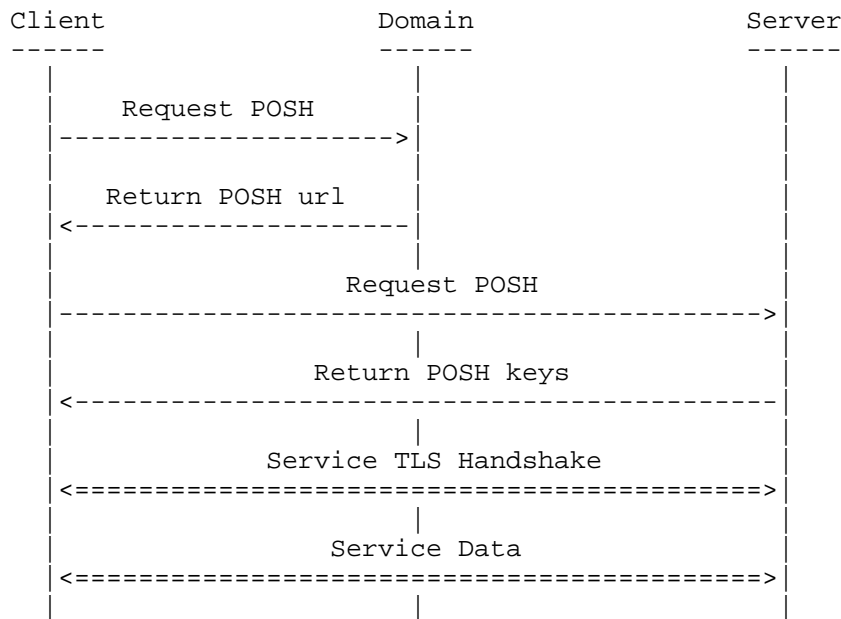
6. Order of Operations

In order for the TLS client to perform verification of reference identifiers without potentially compromising data, POSH processes MUST be complete before any application-level data is exchanged for the source domain. The TLS client SHOULD perform all POSH retrievals before opening any socket connections to the application protocol server. For application protocols that use DNS SRV, the POSH processes ideally ought to be done in parallel with resolving the SRV records and the addresses of any targets, similar to the "happy eyeballs" approach for IPv4 and IPv6 [RFC6555].

The following diagram illustrates the possession flow:



While the following diagram illustrates the reference flow:



7. Caching Results

The TLS client **MUST NOT** cache results (reference or JWK-set) indefinitely. If the source domain returns a reference, the TLS client **MUST** use the lower of the two "expires" values when determining how long to cache results (i.e., if the reference "expires" value is lower than the JWK-set "expires" value, honor the reference "expires" value). Once the TLS client considers the results stale, it **SHOULD** perform the entire POSH process again starting with the HTTPS GET to the source domain. The TLS client **MAY** use a lower value than any provided in the "expires" field(s), or not cache results at all.

The TLS client **SHOULD NOT** rely on HTTP caching mechanisms, instead using the expiration hints provided in the POSH reference or JWK-set documents. To that end, the HTTPS servers for source and derived domains **SHOULD** specify a 'Cache-Control' header indicating a very short duration (e.g., max-age=60) or "no-cache" to indicate that the response (redirect, reference, or content) is not appropriate to cache at the HTTP level.

8. Alternates and Roll-over

To indicate alternate PKIX certificates (such as when an existing certificate will soon expire), the returned JWK set MAY contain multiple JWK objects. The JWK set SHOULD be ordered with the most relevant certificate first as determined by the application service operator (e.g., the renewed certificate), followed by the next most relevant certificate (e.g., the certificate soonest to expire). Here is an example:

```
{
  "keys":[
    {
      "kty": "RSA",
      "kid": "cfc0ca70-1193-11e3-b2b1-835742119fe8",
      "n": "AM-ktWkQ8btj_HEdAA6kOpzJGgoHNZsJmxjh_PifpgAUfQeq
MO_YBR100IdJZRzJfULyhRwn9bikCq87WToxgPWond3sH3qT
YiAcIR5S6tBbsyp6WYmwMlyuC0vLCo6SoDzdK1SvkQKM3QWk
0GFNU4l4qXYAMxaSw83i6yv5DBVbST7E92vS6Gq_4pgI26l1
0JhybZuTEVPRUCG6pTKAXQpLxmJ5oG9M91RP17nsuQeE7Ng
0Ap4BBn5hocojkfthwgbX4lqBMecpBAnky5jn6slmzS_rL-L
w-_8hUldaTPD9MHlHPrvcsRV5uw8wK5MB6Qyfs6wF4b0Kj2T
vYceN1E",
      "e": "AQAB",
      "x5t": "Ae0sLVtm78VT-mQXJQop-ENOM6o"
    },
    {
      "kty": "RSA",
      "kid": "dbc28570-1193-11e3-b2b1-835742119fe8",
      "n": "AM-ktWkQ8btj_HEdAA6kOpzJGgoHNZsJmxjh_PifpgAUfQeq
MO_YBR100IdJZRzJfULyhRwn9bikCq87WToxgPWond3sH3qT
YiAcIR5S6tBbsyp6WYmwMlyuC0vLCo6SoDzdK1SvkQKM3QWk
0GFNU4l4qXYAMxaSw83i6yv5DBVbST7E92vS6Gq_4pgI26l1
0JhybZuTEVPRUCG6pTKAXQpLxmJ5oG9M91RP17nsuQeE7Ng
0Ap4BBn5hocojkfthwgbX4lqBMecpBAnky5jn6slmzS_rL-L
w-_8hUldaTPD9MHlHPrvcsRV5uw8wK5MB6Qyfs6wF4b0Kj2T
vYceN1E",
      "e": "AQAB",
      "x5t": "1YZC2n9TBpOaUsBclEIacQTKToA"
    }
  ]
}
```

9. IANA Considerations

This document registers a well-known URI [RFC5785] for protocols that use POSH. The completed template follows.

URI suffix: posh.

Change controller: IETF

Specification document: [[this document]]

Related information: Because the "posh." string is merely a prefix, protocols that use POSH need to register particular URIs that are prefixed with the "posh." string.

Note that the registered URI is "posh." (with a trailing dot). This is merely a prefix to be placed at the front of well-known URIs [RFC5785] registered by protocols that use POSH, which themselves are responsible for the relevant registrations with the IANA. The URIs registered by such protocols SHOULD match the URI template [RFC6570] path `"/.well-known/posh.{servicedesc}.json"`; that is, begin with "posh." and end with ".json" (indicating a media type of `application/json` [RFC4627] or `application/jwk-set+json` [JOSE-JWK]).

For POSH-using protocols that rely on DNS SRV records [RFC2782], the `"{servicedesc}"` part of the well-known URI SHOULD be `"{service}.{proto}"`, where the `"{service}"` is the DNS SRV "Service" prepended by the underscore character "_" and the `"{proto}"` is the DNS SRV "Proto" also prepended by the underscore character "_". As an example, the well-known URI for XMPP server-to-server connections would be `"posh._xmpp-server._tcp.json"` since XMPP [RFC6120] registers a service name of `"xmpp-server"` and uses TCP as the underlying transport protocol.

For other POSH-using protocols, the `"{servicedesc}"` part of the well-known URI can be any unique string or identifier for the protocol, which might be a service name registered with the IANA in accordance with [RFC6335] or which might be an unregistered name. As an example, the well-known URI for the mythical "Foo" service could be `"posh.foo.json"`.

Note: As explained in [RFC5785], the IANA registration policy [RFC5226] for well-known URIs is Specification Required.

10. Security Considerations

This document supplements but does not supersede the security considerations provided in specifications for application protocols that decide to use POSH (e.g., [RFC6120] and [RFC6125] for XMPP). Specifically, the security of requests and responses sent via HTTPS depends on checking the identity of the HTTP server in accordance with [RFC2818]. Additionally, the security of POSH can benefit from other HTTP hardening protocols, such as HSTS [RFC6797] and key pinning [KEYPIN], especially if the TLS client shares some information with a common HTTPS implementation (e.g., platform-default web browser).

Note well that POSH is used by a TLS client to obtain the public key of a TLS server to which it might connect for a particular application protocol such as IMAP or XMPP. POSH does not enable a hosted domain to transfer private keys to a hosting service via HTTPS. POSH also does not enable a TLS server to engage in certificate enrollment with a certification authority via HTTPS, as is done in Enrollment over Secure Transport [RFC7030].

A web server at the source domain might redirect an HTTPS request to another URL. The location provided in the redirect response MUST specify an HTTPS URL. Source domains SHOULD use only temporary redirect mechanisms, such as HTTP status codes 302 (Found) and 307 (Temporary Redirect). Clients MAY treat any redirect as temporary, ignoring the specific semantics for 301 (Moved Permanently) and 308 (Permanent Redirect) [HTTP-STATUS-308]. To protect against circular references, clients MUST NOT follow an infinite number of redirects. It is RECOMMENDED that clients follow no more than 10 redirects, although applications or implementations can require that fewer redirects be followed.

11. References

11.1. Normative References

- [JOSE-JWK] Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-20 (work in progress), January 2014.
- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.
- [RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000.

- [RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.
- [RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008.
- [RFC5280] Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., and W. Polk, "Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile", RFC 5280, May 2008.
- [RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.
- [RFC6125] Saint-Andre, P. and J. Hodges, "Representation and Verification of Domain-Based Application Service Identity within Internet Public Key Infrastructure Using X.509 (PKIX) Certificates in the Context of Transport Layer Security (TLS)", RFC 6125, March 2011.

11.2. Informative References

- [HTTP-STATUS-308] Reschke, J., "The Hypertext Transfer Protocol (HTTP) Status Code 308 (Permanent Redirect)", draft-reschke-http-status-308-07 (work in progress), March 2012.
- [KEYPIN] Evans, C., Palmer, C., and R. Sleevi, "Public Key Pinning Extension for HTTP", draft-ietf-websec-key-pinning-09 (work in progress), November 2013.
- [XMPP-DNA] Saint-Andre, P. and M. Miller, "Domain Name Associations (DNA) in the Extensible Messaging and Presence Protocol (XMPP)", draft-ietf-xmpp-dna-05 (work in progress), February 2014.
- [RFC2782] Gulbrandsen, A., Vixie, P., and L. Esibov, "A DNS RR for specifying the location of services (DNS SRV)", RFC 2782, February 2000.
- [RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, May 2005.

- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.
- [RFC6335] Cotton, M., Eggert, L., Touch, J., Westerlund, M., and S. Cheshire, "Internet Assigned Numbers Authority (IANA) Procedures for the Management of the Service Name and Transport Protocol Port Number Registry", BCP 165, RFC 6335, August 2011.
- [RFC6555] Wing, D. and A. Yourtchenko, "Happy Eyeballs: Success with Dual-Stack Hosts", RFC 6555, April 2012.
- [RFC6570] Gregorio, J., Fielding, R., Hadley, M., Nottingham, M., and D. Orchard, "URI Template", RFC 6570, March 2012.
- [RFC6698] Hoffman, P. and J. Schlyter, "The DNS-Based Authentication of Named Entities (DANE) Transport Layer Security (TLS) Protocol: TLSA", RFC 6698, August 2012.
- [RFC6797] Hodges, J., Jackson, C., and A. Barth, "HTTPS Strict Transport Security (HSTS)", RFC 6797, November 2012.
- [RFC7030] Pritikin, M., Yee, P., and D. Harkins, "Enrollment over Secure Transport", RFC 7030, October 2013.

Appendix A. Acknowledgements

Many thanks to Philipp Hancke, Joe Hildebrand, and Tobias Markmann for their implementation feedback. Thanks also to Dave Cridland, Chris Newton, Max Pritikin, and Joe Salowey for their input on the specification.

Authors' Addresses

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

Email: mamille2@cisco.com

Peter Saint-Andre
&yet

Email: ietf@stpeter.im

XMPP Working Group
Internet-Draft
Intended status: Standards Track
Expires: March 09, 2014

L. Stout, Ed.
&yet
J. Moffitt
Mozilla
E. Cestari
cstar industries
September 05, 2013

An XMPP Sub-protocol for WebSocket
draft-ietf-xmpp-websocket-00

Abstract

This document defines a binding for the XMPP protocol over a WebSocket transport layer. A WebSocket binding for XMPP provides higher performance than the current HTTP binding for XMPP.

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 March 09, 2014.

Copyright Notice

Copyright (c) 2013 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	3
3. XMPP Sub-Protocol	3
3.1. Handshake	3
3.2. Messages	4
3.3. XMPP Stream Setup	4
3.4. Stream Errors	5
3.5. Closing the Connection	5
3.6. Stanzas	6
3.7. Stream Restarts	6
3.8. Pings and Keepalives	6
3.9. Use of TLS	7
3.10. Stream Management	7
4. Discovering Connection Method	7
5. Security Considerations	8
6. IANA Considerations	8
7. Informative References	8
Authors' Addresses	9

1. Introduction

Applications using XMPP (see [RFC6120] and [RFC6121]) on the Web currently make use of BOSH (see [XEP-0124] and [XEP-0206]), an XMPP binding to HTTP. BOSH is based on the HTTP long polling technique, and it suffers from high transport overhead compared to XMPP's native binding to TCP. In addition, there are a number of other known issues with long polling [RFC6202], which have an impact on BOSH-based systems.

It would be much better in most circumstances to avoid tunneling XMPP over HTTP long polled connections and instead use the XMPP protocol directly. However, the APIs and sandbox that browsers have provided do not allow this. The WebSocket protocol [RFC6455] now exists to solve these kinds of problems. The WebSocket protocol is a bi-directional protocol that provides a simple message-based framing layer over raw sockets and allows for more robust and efficient communication in web applications.

The WebSocket protocol enables two-way communication between a client and a server, effectively emulating TCP at the application layer and therefore overcoming many of the problems with existing long-polling techniques for bidirectional HTTP. This document defines a WebSocket sub-protocol for the Extensible Messaging and Presence Protocol (XMPP).

2. Terminology

The basic unit of framing in the WebSocket protocol is called a message. In XMPP, the basic unit is the stanza, which is a subset of the first-level children of each document in an XMPP stream (see Section 9 of [RFC6120]). XMPP also has a concept of messages, which are stanzas whose top-level element name is message. In this document, the word "message" will mean a WebSocket message, not an XMPP message stanza (see Section 3.2).

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. XMPP Sub-Protocol

3.1. Handshake

The XMPP sub-protocol is used to transport XMPP over a WebSocket connection. The client and server agree to this protocol during the WebSocket handshake (see Section 1.3 of [RFC6455]).

During the WebSocket handshake, the client MUST include the |Sec-WebSocket-Protocol| header in its handshake, and the value |xmpp| MUST be included in the list of protocols. The reply from the server MUST also contain |xmpp| in its own |Sec-WebSocket-Protocol| header in order for an XMPP sub-protocol connection to be established.

Once the handshake is complete, WebSocket messages sent or received will conform to the protocol defined in the rest of this document.

```
C: GET /xmpp-websocket HTTP/1.1
   Host: example.com
   Upgrade: websocket
   Connection: Upgrade
   Sec-WebSocket-Key: dGh1IHhnbXBsZSBub25jZQ==
   Origin: http://example.com
   ...
   Sec-WebSocket-Protocol: xmpp
   Sec-WebSocket-Version: 13

S: HTTP/1.1 101 Switching Protocols
   Upgrade: websocket
   Connection: Upgrade
   ...
   Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
   Sec-WebSocket-Protocol: xmpp
```

[WebSocket connection established]

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

3.2. Messages

Data frame messages in the XMPP sub-protocol MUST be of the text type and contain UTF-8 encoded data. The close control frame's contents are specified in Section 3.5. Control frames other than close are not restricted.

Unless noted in text, the word "message" will mean a WebSocket message composed of text data frames.

3.3. XMPP Stream Setup

The first message sent after the handshake is complete MUST be an XMPP opening stream tag as defined in XMPP [RFC6120] or an XML text declaration (see Section 4.3.1 of [W3C.REC-xml-20081126]) followed by an XMPP opening stream tag. The stream tag MUST NOT be closed (i.e. the closing `</stream:stream>` tag should not appear in the message) as it is the start of the client's outgoing XML. The `'<'` character of the tag or text declaration MUST be the first character of the text payload.

The server MUST respond with a message containing an error (see Section 3.4), its own opening stream tag, or an XML text declaration followed by an opening stream tag.

Except in the case of certain stream errors (see Section 3.4), the opening stream tag, `<stream:stream>`, MUST appear in a message by itself.

3.4. Stream Errors

Stream level errors in XMPP are terminal. Should such an error occur, the server MUST send the stream error as a complete element in a message to the client.

If the error occurs during the opening of a stream, the stream error message MUST start with an opening stream tag (see Section 4.7.1 of [RFC6120]) and end with a closing stream tag.

After the stream error and closing stream tag have been sent, the server MUST close the connection as in Section 3.5.

3.5. Closing the Connection

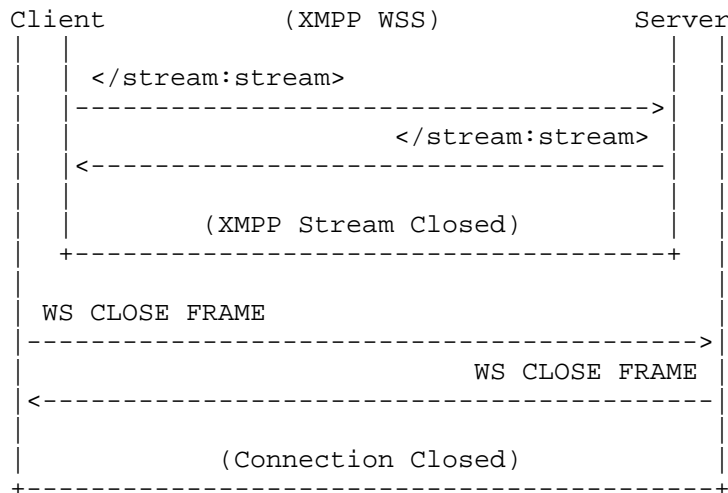
Either the server or the client may close the connection at any time. Before closing the connection, the closing party SHOULD close the XMPP stream, if it has been established, by sending a message with the closing `</stream:stream>` tag. The XMPP stream is considered closed when a corresponding `</stream:stream>` tag is received from the other party.

If a client closes the WebSocket connection without closing the XMPP stream after having enabled stream management (see Section 3.10), the server SHOULD keep the XMPP session alive for a period of time based on server policy, as specified in [XEP-0198].

To initiate closing the WebSocket connection, the closing party MUST send a normal WebSocket close message with an empty body. The connection is considered closed when a matching close message is received (see Section 1.4 of [RFC6455]).

Except in the case of certain stream errors (see Section 3.4), the closing stream tag, `</stream:stream>`, MUST appear in a message by itself.

An example of ending an XMPP over WebSocket session by first closing the XMPP stream layer and then the WebSocket connection layer:



3.6. Stanzas

Each XMPP stanza MUST be sent in its own message. A stanza MUST NOT be split over multiple messages. All first level children of the `<stream:stream>` element MUST be treated the same as stanzas (e.g. `<stream:features>` and `<stream:error>`).

3.7. Stream Restarts

After successful SASL authentication, an XMPP stream needs to be restarted. In these cases, as soon as the message is sent (or received) containing the success indication, both the server and client streams are implicitly closed, and new streams need to be opened. The client MUST open a new stream as in Section 3.3 and MUST NOT send a closing stream tag.

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

[Streams implicitly closed]

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

3.8. Pings and Keepalives

XMPP servers send whitespace pings as keepalives between stanzas, and XMPP clients can do the same as these extra whitespace characters are not significant in the protocol. Servers and clients SHOULD use WebSocket ping control frames instead for this purpose.

In some cases, the WebSocket connection might be served by an intermediary connection manager and not the XMPP server. In these situations, the use of WebSocket ping messages are insufficient to test that the XMPP stream is still alive. Both the XMPP Ping extension [XEP-0199] and the XMPP Stream Management extension [XEP-0198] provide mechanisms to ping the XMPP server, and either extension (or both) MAY be used to determine the state of the connection.

3.9. Use of TLS

TLS cannot be used at the XMPP sub-protocol layer because the sub-protocol does not allow for raw binary data to be sent. Instead, enabling TLS SHOULD be done at the WebSocket layer using secure WebSocket connections via the `wss` URI scheme. (See Section 10.6 of [RFC6455]).

Because TLS is to be provided outside of the XMPP sub-protocol layer, a server MUST NOT advertise TLS as a stream feature (see Section 4.6 of [RFC6120]), and a client MUST ignore any advertised TLS stream feature, when using the XMPP sub-protocol.

3.10. Stream Management

In order to alleviate the problems of temporary disconnections, the XMPP Stream Management extension [XEP-0198] MAY be used to confirm when stanzas have been received by the server.

In particular, the use of session resumption in [XEP-0198] MAY be used to allow for recreating the same stream session state after a temporary network unavailability or after navigating to a new URL in a browser.

4. Discovering Connection Method

The XMPP extension Discovering Alternate XMPP Connection Methods [XEP-0156] provides a mechanism to discover the additional information needed to connect to an XMPP server outside of the procedure defined in in Section 3 of [RFC6120].

For the XMPP over Websocket connection type, the connection method name `"_xmpp-client-websocket"` is used to specify a URI for the server's WebSocket connection endpoint.

An example entry advertising that the URI "wss://example.com/xmpp" is an XMPP over WebSocket endpoint, using a DNS TXT record as specified in [XEP-0156]:

```
_xmppconnect IN TXT "_xmpp-client-websocket=wss://example.com/xmpp"
```

Implementation Note: A server is able to expose both BOSH [XEP-0206] and WebSocket endpoints over the registered port 5280, using the URI path and connection upgrade headers to determine which transport to serve.

5. Security Considerations

Since application level TLS cannot be used (see Section 3.9), applications which need to protect the privacy of the XMPP traffic need to do so at the WebSocket or other appropriate layer.

The Security Considerations for both WebSocket (See Section 10 of [RFC6455] and XMPP (See Section 13 of [RFC6120]) apply to the WebSocket XMPP sub-protocol.

6. IANA Considerations

This specification requests IANA to register the WebSocket XMPP sub-protocol under the "WebSocket Subprotocol Name" Registry with the following data:

Subprotocol Identifier: xmpp

Subprotocol Common Name: WebSocket Transport for the Extensible Messaging and Presence Protocol (XMPP)

Subprotocol Definition: RFC XXXX

[[NOTE TO RFC EDITOR: Please change XXXX to the number assigned to this document upon publication.]]

7. Informative References

- [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.
- [RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 6120, March 2011.

- [RFC6121] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence", RFC 6121, March 2011.
- [RFC6202] Loreto, S., Saint-Andre, P., Salsano, S., and G. Wilkins, "Known Issues and Best Practices for the Use of Long Polling and Streaming in Bidirectional HTTP", RFC 6202, April 2011.
- [RFC6455] Fette, I. and A. Melnikov, "The WebSocket Protocol", RFC 6455, December 2011.
- [W3C.REC-xml-20081126]
Sperberg-McQueen, C., Yergeau, F., Paoli, J., Bray, T.,
and E. Maler, "Extensible Markup Language (XML) 1.0 (Fifth
Edition)", World Wide Web Consortium Recommendation REC-
xml-20081126, November 2008,
<<http://www.w3.org/TR/2008/REC-xml-20081126>>.
- [XEP-0124] Paterson, I., Smith, D., Saint-Andre, P., and J. Moffitt,
"Bidirectional-streams Over Synchronous HTTP (BOSH)", XSF
XEP 0124, July 2010.
- [XEP-0156] Hildebrand, J. and P. Saint-Andre, "Discovering
Alternative XMPP Connection Methods", XSF XEP 0156, June
2007.
- [XEP-0198] Karneges, J., Saint-Andre, P., Hildebrand, J., Forno, F.,
Cridland, D., and M. Wild, "Stream Management", XSF XEP
0198, June 2011.
- [XEP-0199] Saint-Andre, P., "XMPP Ping", XSF XEP 0199, June 2009.
- [XEP-0206] Paterson, I. and P. Saint-Andre, "XMPP Over BOSH", XSF XEP
0206, July 2010.

Authors' Addresses

Lance Stout (editor)
&yet

Email: lance@andyet.net

Jack Moffitt
Mozilla

Email: jack@metajack.im

Eric Cestari
cstar industries

Email: eric@cestari.info

XMPP
Internet-Draft
Intended status: Standards Track
Expires: December 15, 2013

M. Miller
Cisco Systems, Inc.
June 13, 2013

End-to-End Object Encryption and Signatures for the Extensible Messaging
and Presence Protocol (XMPP)
draft-miller-xmpp-e2e-06

Abstract

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

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 December 15, 2013.

Copyright Notice

Copyright (c) 2013 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
2. Terminology	3
3. Encryption	3
3.1. Determining Support	4
3.2. Encrypting XMPP Stanzas	4
3.2.1. Prerequisites	4
3.2.2. Process	5
3.3. Decrypting XMPP Stanzas	7
3.3.1. Protocol Not Understood	7
3.3.2. Process	7
3.3.3. Insufficient Information	8
3.3.4. Failed Decryption	9
3.3.5. Timestamp Not Acceptable	10
3.3.6. Successful Decryption	11
3.4. Example - Securing a Message	11
4. Signatures	14
4.1. Determining Support	14
4.2. Signing XMPP Stanzas	15
4.2.1. Process	15
4.3. Verifying Signed XMPP Stanzas	17
4.3.1. Protocol Not Understood	17
4.3.2. Process	17
4.3.3. Insufficient Information	18
4.3.4. Failed Verification	18
4.3.5. Timestamp Not Acceptable	19
4.3.6. Successful Verification	20
4.4. Example - Signing a Message	20
5. Requesting Session Keys	23
5.1. Request Process	23
5.2. Accept Process	24
5.3. Error Conditions	25
5.4. Example of Successful Key Request	26
6. Multiple Operations	30
7. Inclusion and Checking of Timestamps	30
8. Interaction with Stanza Semantics	31
9. Interaction with Offline Storage	31
10. Mandatory-to-Implement Cryptographic Algorithms	32
11. Security Considerations	32
11.1. Storage of Encrypted Stanzas	32
11.2. Re-use of Session Master Keys	32

12. IANA Considerations	32
12.1. XML Namespaces Name for e2e Data in XMPP	32
13. References	33
13.1. Normative References	33
13.2. Informative References	34
Appendix A. Schema for urn:ietf:params:xml:ns:xmpp-e2e:6	35
Appendix B. Acknowledgements	38
Author's Address	38

1. Introduction

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

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

Comments are solicited and should be addressed to

2. Terminology

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

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

3. Encryption

3.1. Determining Support

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

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

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

3.2. Encrypting XMPP Stanzas

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

3.2.1. Prerequisites

First, the sending agent prepares and retains the following:

- o The JID of the sender (i.e. its own JID). This SHOULD be the bare JID (localpart@domainpart).
- o The JID of the recipient. This SHOULD be the bare JID (localpart@domainpart).
- o A Session Master Key (SMK). The SMK MUST have a length at least equal to that required by the key wrapping algorithm in use and MUST be generated randomly. See [RFC4086] for considerations on generating random values.

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

3.2.2. Process

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

1. Ensures the plaintext stanza is fully qualified, including the proper namespace declarations (e.g., contains the attribute 'xmlns' set to the value "jabber:client" for 'jabber:client' stanzas defined in [RFC6120]).
2. Notes the current UTC date and time (N) when this stanza is constructed, formatted as described under Section 7.
3. Constructs a forwarding envelope (M) using a <forwarded/> element qualified by the "urn:xmpp:forward:0" namespace (as defined in [XEP-0297]) as follows:
 - * The child element <delay/> qualified by the "urn:xmpp:delay" namespace (as defined in [XEP-0203]) with the attribute 'stamp' set to the UTC date and time value N
 - * The plaintext stanza S
4. Converts the forwarding envelope (M) to a UTF-8 encoded string (M'), optionallly removing line breaks and other insignificant whitespace between elements and attributes, i.e. M' = UTF8-encode(M). We call M' a "stanza-string" because for purposes of encryption and decryption it is treated not as XML but as an opaque string (this avoids the need for complex canonicalization of the XML input).
5. Generates a Content Master Key (CMK). The CMK MUST have a length at least equal to that required by the content encryption algorithm in use and MUST be generated randomly. See [RFC4086] for considerations on generating random values.

6. Generates any additional unprotected block cipher factors (IV); e.g., initialization vector/nonce. A sending agent MUST ensure that no two sets of factors are used with the same CMK, and SHOULD NOT reuse such factors for other stanzas.
7. Performs the message encryption steps from [JOSE-JWE] to generate the JWE Header (H), JWE Encrypted Key (E), JWE Ciphertext (C), and JWE Integrity Value (I); using the following inputs:
 - * The 'alg' property is set to an appropriate key wrapping algorithm (e.g., "A256KW" or "A128KW"); recipients use the key request process in Section 5 to obtain the SMK.
 - * The 'enc' property is set to the intended content encryption algorithm.
 - * SMK as the key for CMK Encryption.
 - * CMK as the JWE Content Master Key.
 - * IV as the JWE Initialization Vector.
 - * M' as the plaintext content to encrypt.
8. Constructs an <e2e/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace as follows:
 - * The attribute 'type' set to the value "enc".
 - * The attribute 'id' set to the identifier value SID.
 - * The child element <encheader/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as H, encoded base64url as per [RFC4648].

- * The child element <cmk/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character as E, encoded base64url as per [RFC4648].
 - * The child element <iv/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character as IV, encoded base64url as per [RFC4648].
 - * The child element <data/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as C, encoded base64url as per [RFC4648].
 - * The child element <mac/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as I, encoded base64url as per [RFC4648].
9. Sends the <e2e/> element as the payload of a stanza that SHOULD match the stanza from step 1 in kind (e.g., <message/>), type (e.g., "chat"), and addressing (e.g., to="romeo@montague.net" from="juliet@capulet.net/balcony"). If the original stanza (S) has a value for the 'id' attribute, this stanza MUST NOT use the same value for its 'id' attribute.

3.3. Decrypting XMPP Stanzas

3.3.1. Protocol Not Understood

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

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

3.3.2. Process

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

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

the sending agent JID specified by the 'from' attribute of the wrapping stanza. If the receiving agent does not already have the SMK, it requests it according to Section 5.

2. Performs the message decryption steps from [JOSE-JWE] to generate the plaintext forwarding envelope string M', using the following inputs:
 - * The JWE Header (H) from the <encheader/> element's character data content.
 - * The JWE Encrypted Key (E) from the <cmk/> element's character data content.
 - * The JWE Initialization Vector/Nonce (I) from the <iv/> element's character data content.
 - * The JWE Ciphertext (C) from the <data/> element's character data content.
 - * The JWE Integrity Value (I) from the <mac/> element's character data content.
3. Converts the forwarding envelope UTF-8 encoded string M' into XML element (M).
4. Obtains the UTC date and time (N) from the <delay/> child element, and verifies it is within the accepted range, as specified in Section 7.
5. Obtains the plaintext stanza (S), which is a child element node of M; the stanza MUST be fully qualified with proper namespace declarations for XMPP stanzas, to help distinguish it from other content within M.

.

3.3.3. Insufficient Information

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

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

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

3.3.4. Failed Decryption

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

```
<message xmlns='jabber:client'
  from='juliet@capulet.lit/balcony'
  id='fJZd9WFIIwNjFctT'
  to='romeo@montegue.lit/garden'
  type='chat'>
```

```
<e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
      id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
  <encheader>[XML character data]</encheader>
  <cmk>[XML character data]</cmk>
  <iv>[XML character data]</iv>
  <data>[XML character data]</data>
  <mac>[XML character data]</mac>
</e2e>
<error type='modify'>
  <bad-request xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
  <decryption-failed xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6' />
</error>
</message>
```

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

3.3.5. Timestamp Not Acceptable

At step 4, if the stanza is successfully decrypted but the timestamp fails the checks outlined in Section 7, the receiving agent MAY return a <not-acceptable/> error to the sender (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <bad-timestamp/> (previously defined in [RFC3923]):


```

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

```

3.3.6. Successful Decryption

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

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

3.4. Example - Securing a Message

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

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

```

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

```

```
    The more I have, for both are infinite.
  </body>
</message>
```

and the following prerequisites:

- o Sender JID as "juliet@capulet.lit/balcony"
- o Recipient JID as "romeo@montague.lit"
- o Session Master Key (SMK) as (base64 encoded)
"xWtdjhYsH4Va_9SfYSeFsJfZu03m5RrbXo_UavxxeU8"
- o SMK identifier (SID) as "835c92a8-94cd-4e96-b3f3-b2e75a438f92"

The sending agent performs steps 1, 2, and 3 from Section 3.2.2 to generate the envelope:

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

Then the sending agent performs steps 4 through 7 (with Content Master Key as "LViSXX0Jx-I3v1zY1-KcGeivmWKuq0QE_7lywQGU6OhlM2NoQo1zHi77zI3ieIUh7Wb1S3kXmNily0_FZoIG7A", base64url encoded) to generate the [JOSE-JWE] outputs:

JWE Header

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

JWE Encrypted Key

```
2tsmGH-WQdBxxJES3d6LB2ovK6e1_9ClogizJ9c6OvLmC6IeilHZ2Mimq2AElgI
ploz0VQv5LOH9ST93WvvhVzMHSfx0Cwl0
```

JWE Initialization Vector

```
ncOH4MsHT9HlJxnirx4qwg
```

JWE Ciphertext

```
FkFc4xGTVkjin7ojtS0SUY8IWfqsQKEIAlvLaBKieqVXlPA1qlZjPp4TZC2I2eh7
0lLef3iRuNZdlnlgP2aREyHYCpE3FAelUoVG90B1FrJMnDUKAka7eb6GImamWPf
9onV-m5-GcUpejO9f1oPi-rwHzp475UPdAeKq5Z4zds8yXhQP-XyJbCPTtM-UQC
2-_q-3EKBHC4jM3qWDxVJ0JbIif3fCVRowzJh4AOB84YrfvkgUjMitqQPg2H6QB
NqGUspLI634lM8R-mhGciDZX2Jh_nKoXLAf5GCnvL9PlI7OdFqocPBIIPpjNrgX
_Z4PFjeq7ILx98GhVkryLYU9HVOFPCYci-lF9nfwlgeliLfkoj5QZyi4J2SotYa
O_zPmQvCXaUREqPf5UDAlgvc50a4ByYnNbkWSbhZ5Z388s8ELzPSE9XypdgP-lc
SyRke7V8iGe4eHNsm0lTgWILYOFK4mYAM52OTitJxmQtmRp6izY5ZFdh9f_WdoB
lRXmGEZydvL-estcJx5ghsV3gktdIl0HA4R_M_N5TFIwv7hiisyRLi2aQtyFbE
7pZ6Oz-cYsLc4qFfXbb13U9a2-Byu18hm_E2b3m4GMhmsCiROm-uh9Ek4h9BIx
FhDKPr-hTOXc93-uQNZlAQfkITAKlJfQ
```

JWE Integrity Value

```
Aj8lKdPMDE4U82UAhDJBaRrl3USmuzS2hfFOe_OBEv8
```

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

```
<message xmlns='jabber:client'
  from='juliet@capulet.lit/balcony'
  id='fJZd9WFIiwNjFctT'
  to='romeo@montague.lit'
  type='chat'>
```

```

<e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
      type='enc'
      id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
  <encheader>
    eyJhbGciOiJBbmJlLCJlbmMiOiJBbmJlU2Q0JDK0hTNTEyIiwia2lkI
    joiODMlYzkyYTgtOTRjZC00ZTk2LWIzZjMtYjJlNzVhNDM4ZjkyIn0
  </encheader>
  <cmk>
    2tsmGH-WQdBxxJES3d6LB2ovK6e1_9ClogizJ9c6OvLmC6IeilHZ2Mimq
    2AElgIploz0VQv5LOH9ST93WvvhVzMHSfx0Cwl0
  </cmk>
  <iv>
    ncOH4MsHT9HlJxnirx4qwg
  </iv>
  <data>
    FkFc4xGTVkjin7ojtS0SUY8IWfqsQKEIAlvLaBKieqVXlPAIqlZjPp4TZC
    2I2eh70lLef3iRuNZdlnlgP2aREyHYCpE3FAelUoVG90B1FrJMnDUKAKa
    7eb6GImamWPf9onV-m5-GcUpejO9floPi-rwHzp475UPdAeKq5Z4zds8y
    XhQP-XyJbCPTtM-UQC2-_q-3EKBHC4jM3qWDxVJ0JbIif3fCVRowzJh4A
    OB84YrfvkgUjMItqQPg2H6QBNqGUspLI634lM8R-mhGciDZX2Jh_nKoXL
    Af5GCnvL9PlI70dFqocPBIIPpjNrgX_Z4PFjeq7ILx98GhVkryLYU9HVO
    FPCYci-lF9nfwlgeliLfk0j5QZyi4J2SotYaO_zPmQvCXaUREqPf5UDAl
    gvc50a4ByYnNbkWSbhZ5Z388s8ELzPSE9XypdgP-lcSyRke7V8iGe4eHN
    sm0lTgWILYOFK4mYAM52OTitJxmQtmRp6izY5ZFdH9f_WdoBlRXmGEZyd
    vL-estcjsx5ghsV3gktdIl0HA4R_M_N5TFIwv7hiisyRLi2aQtyFbE7pZ
    60z-cYsLc4qFfXbb13U9a2-Byu18hm_E2b3m4GMhmsCiROm-uht9Ek4h9
    BixFhDKPr-htOXc93-uQNZlAQfkITAKlJfQ
  </data>
  <mac>
    Aj8lKdPMDE4U82UAhDJBaRrl3USmuzS2hffOe_OBEv8
  </mac>
</e2e>
</message>

```

4. Signatures

4.1. Determining Support

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

```

<iq xmlns='jabber:client'
  id='disco1'
  to='romeo@montegue.lit/garden'
  type='result'>

```

```
<query xmlns='http://jabber.org/protocol/disco#info'>
  ...
  <feature xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6:signatures' />
  ...
</query>
</iq>
```

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

4.2. Signing XMPP Stanzas

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

4.2.1. Process

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

1. Ensures the plaintext stanza is fully qualified, including the proper namespace declarations (e.g., contains the attribute 'xmlns' set to the value "jabber:client" for 'jabber:client' stanzas defined in [RFC6120]).
2. Notes the current UTC date and time (N) when this stanza is constructed, formatted as described under Section 7.
3. Constructs a forwarding envelope (M) using a <forwarded/> element qualified by the "urn:xmpp:forward:0" namespace (as defined in [XEP-0297]) as follows:
 - * The child element <delay/> qualified by the "urn:xmpp:delay" namespace (as defined in [XEP-0203]) with the attribute 'stamp' set to the UTC date and time value N
 - * The plaintext stanza S
4. Converts the forwarding envelope (M) to a UTF-8 encoded string (M'), optionallly removing line breaks and other insignificant whitespace between elements and attributes, i.e. M' =

UTF8-encode(M). We call M' a "stanza-string" because for purposes of encryption and decryption it is treated not as XML but as an opaque string (this avoids the need for complex canonicalization of the XML input).

5. Chooses a private asymmetric key (PK) for which the sending agent has published the corresponding public key to the intended recipients.
6. Performs the message signatures steps from [JOSE-JWS] to generate the JWS Header (H) and JWS Signature (I); using the following inputs:
 - * The 'alg' property is set to an appropriate signature algorithm for PK (e.g., "R256").
 - * M' as the JWS Payload.
7. Constructs an <e2e/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace as follows:
 - * The attribute 'type' set to the value "sig"
 - * The child element <sigheader/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as H, encoded base64url as per [RFC4648].
 - * The child element <data/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as M', encoded base64url as per [RFC4648].
 - * The child element <sig/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as I, encoded base64url as per [RFC4648].
8. Sends the <e2e/> element as the payload of a stanza that SHOULD match the stanza from step 1 in kind (e.g., <message/>), type (e.g., "chat"), and addressing (e.g., to="romeo@montegue.lit" from="juliet@capulet.lit/balcony"). If the original stanza (S) has a value for the 'id' attribute, this stanza SHOULD NOT use the same value for its "id" attribute.

4.3. Verifying Signed XMPP Stanzas

4.3.1. Protocol Not Understood

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

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

4.3.2. Process

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

1. Ensures it has appropriate materials to verify the signature, which generally means ensuring that it possesses one or more public keys for the sending agent (if one is not provided as part of the JWS Header).
2. Performs the message validation steps from [JOSE-JWS], with the following inputs:
 - * The JWS Header H from the <sigheader/> element's character data content.
 - * The JWS Payload M' from the <data/> element's character data content.
 - * The JWS Signature from the <sig/> element's character data content.
3. Converts the forwarding envelope UTF-encoded string M' into XML element M.
4. Obtains the UTC date and time N from the <delay/> child element, and verifies it is within the accepted range, as specified in Section 7.

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

4.3.3. Insufficient Information

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

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

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

4.3.4. Failed Verification

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


```

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

```

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

4.3.5. Timestamp Not Acceptable

At step 4, if the stanza is successfully verified but the timestamp fails the checks outlined in Section 7, the receiving agent MAY return a <not-acceptable/> error to the sender (as described in [RFC6120]), optionally supplemented by an application-specific error condition element of <bad-timestamp/> (previously defined in [RFC3923]):

```

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

```

```
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas' />
      <bad-timestamp
        xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6' />
    </error>
</message>
```

4.3.6. Successful Verification

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

Otherwise, the receiving agent SHOULD send the <iq/> response signed as per Section 4.2.1, with the 'type' attribute set to the value "result", even if the response to the signed <iq/> stanza is of type "error". The error applies to the signed stanza, not the wrapping stanza.

4.4. Example - Signing a Message

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

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

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

Then the sending agent performs steps 1, 2, and 3 from Section 4.2.1 generate the envelope M:

```
<forwarded xmlns='urn:xmpp:forward:0'>
  <delay xmlns='urn:xmpp:delay'>
```

```

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

```

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

JWS Header (before base64url encoding)

```

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

```

JWS Payload

```

PGZvcndhcmRlZCB4bWxucuz0idXJuOnhtcHA6Zm9yd2FyZDowIj48ZGVsYXkgeG1
sbnM9InVybjp4bXBwOmRlbGF5IiBzdGFtcD0iMTQ5Mi0wNS0xM1QyMDowNzozNy
4wMTJaIi8-PG1lc3NhZ2UgeG1sbnM9ImphYmJlcjpbGllbnQiIGZyb209Imp1b
GlldEBjYXB1bGV0LmxdpC9iYWxjb255IiB0bz0icm9tZW9AbW9udGVndWUubGl0
IiB0eXB1PSJjaGF0Ij48dGhyZWFKPjM1NzQwYmU1LWIlYTQtNGM0ZS05NjJhLWE
wM2IxNGVktOTJmNDwvdGhyZWFKPjxib2R5PkJldCB0byBiZSBmcmFuaywgYW5kIG
dpdmUgaXQgdGhlZSBhZ2Fpbi4gQW5kIHlldCBJIHdpc2ggYnV0IGZvciB0aGUgd
GhpbmcmGSSBoYXZlLiBNeSBib3VudHkgaXMgYXMGYm91bmRsZXNzIGFzIHRoZSBz
ZWEsIE15IGxvdmUgYXMGZGVlcDsgdGhlIGlvcmUgSSBnaXZlIHRvIHRoZWUsIFR
oZSBtb3JlIEkgaGF2ZSwgZm9yIGJvdGggYXJlIGluZmluaXRlLjwvYm9keT48L2
1lc3NhZ2U-PC9mb3J3YXJkZWQ-

```

JWS Signature

YPfGouD50j0C_C-RneawG0jxXWDXgBkN3FJz6eaBFIPCh3hopiwtwKir7Yamvgt
OrqhXx2pcu-70caGi6mKKLWvpdwdJ3nEnhdjPod3CmLdaK_PBAMtIt8d3155hdl
qNxSMsJN7PxmNLNwJhbkAsI-2TcCQsuxdIPXh6hcqBm44BpVio6AoRPqWf06XZ
MMBOMnEfCV6Ht20wCK1BEGGmN3KYPbwKeTctG8HKPAh25_K66aEXT661I19uW
j1fGFJ79QCHUhc5y9pSKmpK7HKruPMRyrvpzBSfUhc62nLXhM-LzY5taaDECzi
fCi-IxySBtJtPCqYAYW_IbrRfG

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

```
<message xmlns='jabber:client'
  from='juliet@capulet.lit/balcony'
  id='6aAWpciGV98qaegk'
  to='romeo@montegue.lit'
  type='cat'>
<e2e xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
  type='sig'>
  <sigheader>
    eyJhbGciOiJSUzUxMiIsImtpZCI6ImplbGlldEBjYXB1bGV0LmxpdCJ9
  </sigheader>
  <data>
    PGZvcndhcmRlZCB4bWxucuz0idxJuOnhtcHA6Zm9yd2FyZDowIj48ZGVsY
    XkgeGlsbnM9InVybjp4bXBwOmRlbGF5IiBzdGFtcD0iMTQ5Mi0wNS0xMl
    QyMDowNzozNy4wMTJaIi8-PG1lc3NhZ2UgeG1sbnM9ImpPhYmJlcjppbjB
    lbnQiIGZyb209ImplbGlldEBjYXB1bGV0LmxpdC9iYWxjb255IiB0bz0i
    cm9tZW9AbW9udGVndWUubGl0IiB0eXB1PSJjaGF0Ij48dGhyZWZkPjMlN
    zQwYmU1LWl1YTQtNGM0ZS05NjJhLWEwM2IxNGVkJOTJmNDwvdGhyZWZkPj
    xib2R5PkJldCB0byBiZSBmcmFuaywgYW5kIGdpdmUgaXQgdGhlZSBhZ2F
    pbi4gQW5kIHlldCBJIHdpc2ggYnV0IGZvciB0aGUgdGhpbmcmcgSSBoYXZl
    LiBNeSBib3VudHkgaXMGYXMGYm91bmRsZXNzIGFzIHRoZSBzZWESIE15I
    GxvdmUgYXMGZGVlcDsgdGhlIGlvcmluUgSSBnaXZlIHRvIHRoZWUuSFRoZS
    Btb3JlIEkgaGF2ZSwgZm9yIGJvdGggYXJlIGluZmluaXRlLjpwYm9keT48
    8L21lc3NhZ2U-PC9mb3J3YXJkZWQ-
  </data>
  <sig>
    YPfgGouD50j0C_C-RneawG0jxXWDXgBkN3FJz6eaBFIPCh3hopiwtwKir7
    YamvgTOrqhXx2pcu-70caGi6mKKLWvpdwdJ3nEnhdpOd3CmLdaK_PBAM
    tIt8d3155hdlqNxSMsJN7PxmNLNwJhbksAsI-2TcCQsuxdIPXh6hcqBm4
    4BpVio6AoRPqwF06XZMMBMOMnEFcV6Ht20wCK1BEGGomN3KYPbwKeTctG
    8HKPAh25_k66aEXT66l1i9uWj1fGFJ79QQHUhc5y9pSKmpK7HKruPMRyr
    vpzBSfUhcb62nLxhM-LzY5taaDECzifCi-IxySBtJtPCqYAYW_IbrRFg
  </sig>
</e2e>
</message>
```

5. Requesting Session Keys

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

5.1. Request Process

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

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

1. Constructs a [JOSE-JWK] JWK Set (KS), containing information about each public key the requesting agent wishes to use. Each key **SHOULD** include a value for the property 'kid' which uniquely identifies it within the context of all provided keys. Each key **MUST** include a value for the property 'kid' if any two keys use the same algorithm.
2. Constructs a <keyreq/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace as follows:
 - * The attribute 'id' set to the SMK identifier value SID.
 - * The child element <pkey/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as KS, encoded base64url as per [RFC4648].
3. Sends the <keyreq/> element as the payload of an <iq/> stanza with the attribute 'type' set to "get", the attribute 'to' set to the full JID of the original encrypted stanza's sender, and the attribute 'id' set to an opaque string value the receiving agent uses to track the <iq/> response.

5.2. Accept Process

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

1. Generate a JSON Web Key (JWK) representing the symmetric SMK (according to [JOSE-JWK]):
 - * The "kty" parameter MUST be "oct".
 - * The "kid" parameter MUST be the SID.
 - * The "k" parameter MUST be the SMK, encoded as base64url.
 - * The "use" parameter, if present, MUST be set to the algorithm in use for encrypting messages from Section 3.2.
 - * The "use" parameter, if present, MUST be set to "enc".
2. Chooses a key (PK) from the keys provided via KS, and notes its identifier value 'kid'.
3. Protects the SMK using the process outlined in [JOSE-KEYPROTECT] to generate the JWE Header (H), JWE Encrypted Key (E), JWE Initialization Vector (IV), JWE Ciphertext (C), and JWE Integrity Value (I); using the following inputs:
 - * The 'alg' property is set to an algorithm appropriate for the chosen PK (e.g., "RSA-OAEP" for a "RSA" key).
 - * The 'enc' property is set to the intended content encryption algorithm.
 - * A randomly generated CMK. See [RFC4086] for considerations on generating random values.

- * A randomly generated initialization vector. See [RFC4086] for considerations on generating random values.
 - * SMK, formatted as a JWK as above.
4. Constructs a <keyreq/> element qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace as follows:
- * The attribute 'id' set to the SMK Identifier (SID).
 - * The child element <encheader/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as H, encoded base64url as per [RFC4648].
 - * The child element <cmk/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as E, encoded base64url as per [RFC4648].
 - * The child element <iv/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as IV, encoded base64url as per [RFC4648].
 - * The child element <data/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as C, encoded base64url as per [RFC4648].
 - * The child element <mac/> qualified by the "urn:ietf:params:xml:ns:xmpp-e2e:6" namespace and with XML character data as I, encoded base64url as per [RFC4648].
5. Sends the <keyreq/> element as the payload of an <iq/> stanza with the attribute 'type' set to "result", the attribute 'to' set to the full JID from the request <iq/>'s 'from' attribute, and the attribute 'id' set to the value of the request <iq/>'s 'id' attribute.

5.3. Error Conditions

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

- o <forbidden/>: the key request is made by an entity that is not authorized to decrypt stanzas from the sending agent and/or for the indicated SID.
- o <item-not-found/>: the requested SID is no longer valid.
- o <not-acceptable/>: the key request did not contain any keys the sending agent understands.

5.4. Example of Successful Key Request

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

To begin a key request, the receiving agent performs step 1 from Section 5.1 to generate the [JOSE-JWK]:

```
{
  "keys": [{
    "kty": "RSA",
    "kid": "romeo@montegue.lit/garden",
    "n": "vtqejkMF0lh8oKEaHfHEY00C2jM7eISbbSvNs0SNItYWO6GbjpJf
N4ldXw2vpVRdysnwU3zk6o2_SD0YCHlWgeuI0QK1knMTDdNSXx52e1c4BTw
h1A8iHuutTWmpBqesn1GNZmqB3jYsJ0kVBYwCJtkB9APaBvk0itlRtizjCf
1HHnau7nGStyshgu8-srxid8rC5TTLsB_zTli6fP8fwDloemXOtC0U65by
5P-1ZHxaf_bD8fpjps6gwSgdkZKMJAi0bOWZWuMpp2ntqa0wLB7Ndx2Ijr
eog_s5ssAoSiXDVdoswSbp36ZP-1lnCk2j-vZ4qbhaFg5bZtgt-gwQ",
    "e": "AQAB"
  }]
}
```

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

```
<keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
  id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
  <pkey>
    eyJrZXlziJpbeyJrdHkiOiJSU0EiLCJraWQiOiJyb2llb0Btb250ZWdlZS55
    saXQvZ2FyZGVuIiwibI6InZ0cWVqa0lGMDFoOG9LRWFIZkhFWU8wQzJqTT
    dlSVNiYlN2TnMwU05JdFlXTzZHYmpwSmZONGxkWHcydnBWUmR5c253VTN6a
```



```

zZvMl9TRDBZQ0gxV2dldUkwUUsxa25NVERkTlNYeDUyZTFjNEJUd2hsQThp
SHVldFRXbXBCCcWVzbjFHTlptcUIza1lzSk9rVkZd0NKdGtCOUFQYUJ2azB
pdGxSdGl6akNmMUhIbmF1N25HU3R5c2hndTgtc3J4aV9kOHJDNRUTFNCX3
pUMWk2ZlA4ZndEbG9lbVhPdEMwVTY1Ynk1UC0xWkh4YWZfYkQ4ZnBqcHM2Z
3dTZ2RrWktNSkFJMGJpV1pXdU1wcDJudHFhMHdMQjdOZHhiMklqcmVvZl9z
NXNzQW9TaVhEVmRvc3dTynAzNlpQLTFsbkNrMmotdlo0cWJoYUZnNWJadGd
0LWd3USIsImUiOiJBuUFCInldfQ
</pkey>
</keyreq>

```

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

```

<iq xmlns='jabber:client'
  from='romeo@montegue.lit/garden'
  id='xdJbWMA+'
  to='juliet@capulet.lit/balcony'
  type='get'>
  <keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
    id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
    <pkey>
      eyJrZXlziJpbeyJrdHkiOiJSU0EiLCJraWQiOiJyb2llb0Btb250ZWdlZ
      S5saXQvZ2FyZGVuIiwibIi6InZ0cWVqa0lGMDFoOG9LRWFIZkhFWU8wQz
      JqTTdlSVNiYlN2TnMwU05JdFlXTzZHYmpwSmZONGxkWHcydnBWUmr5c25
      3VTN6azZvMl9TRDBZQ0gxV2dldUkwUUsxa25NVERkTlNYeDUyZTFjNEJU
      d2hsQThpSHVldFRXbXBCCcWVzbjFHTlptcUIza1lzSk9rVkZd0NKdGtCO
      UUFQYUJ2azBpdGxSdGl6akNmMUhIbmF1N25HU3R5c2hndTgtc3J4aV9kOH
      JDNRUTFNCX3pUMWk2ZlA4ZndEbG9lbVhPdEMwVTY1Ynk1UC0xWkh4YWZ
      fYkQ4ZnBqcHM2Z3dTZ2RrWktNSkFJMGJpV1pXdU1wcDJudHFhMHdMQjdO
      ZHhiMklqcmVvZl9zNXNzQW9TaVhEVmRvc3dTynAzNlpQLTFsbkNrMmotd
      lo0cWJoYUZnNWJadGd0LWd3USIsImUiOiJBuUFCInldfQ
    </pkey>
  </keyreq>
</iq>

```

If the sending agent accepts this key request, it performs step 1 from Section 5.2 to generate JWK representation of the SMK:

```

{
  "kty": "oct",
  "kid": "835c92a8-94cd-4e96-b3f3-b2e75a438f92",
  "k": "xWtdjYhSH4Va_9SfYSeFsJfZu03m5RrbXo_UavxxeU8"
}

```

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

JWE Header (before base64url encoding)

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

JWE Encrypted Key

```
hKUOpAif76c-hmRwEphVB9wXjloLpwu75x98MSWyCBtfUgmopk93ttUXoZ4AAIk
rZJOtrPUqPZwYHjay3ggfgjVljj_KGhgqI5cScIzaAQs0Pxep6FnrsnUrw09Sjv
2VRXOay4guMQnbQo0ibpifBxeuL9MJ_vdeb_BdSE8YZ4iTfMb7GT35gZC9NgweX
3fiTEo2LjY8hEV3DHud5LlNZzYp9kLmAUZNIwGu7LtYyI4F7NnOv9oLx1HtmfE3
_skkYtQoKMvMewLkIO88h325qCpWFdrLwPp63betCmewDJPaBdrp91rLchkXVo-
d2ueKkb59TxWjMx7esBdaxCACDQ
```

JWE Initialization Vector

```
Ggiego8UiSsj7GgY94qOng
```

JWE Ciphertext

```
4vIGDz9Hm6X4lSo9JoA6ZzS0KitztLGaiMUs3RTviFO09choPhxJNlOj8KX8QIL
u4zZ-ytCnG-yzNx5SsT8KEQJhIf6_9yWplxpXl73k6ZJV-sXGd4Mj9u7N0IqWQL
K5DMytv7XopsZsR9QFCDNGew
```

JWE Integrity Value

```
3GuaasWV0XGTBbRtNP6OQl4_cHL-ZJClnaDtU6EIecw
```

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

```
<keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
  id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
  <encheader>
    eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InJvbWVvQG1vbnRlZ3VlLmxpdc9
    nYXJkZW4iLCJlbmMiOiJBMjU2Q0JDK0hTNTFyIiwiaWF0IjY3R5IjoieYXBwbGljYX
    Rpb24vandrk2pz24ifQ
  </encheader>
  <cmk>
```

```

hKUOpAif76c-hmRwEphVB9wXjloLpwu75x98MSWyCBtfUgmopk93ttUXoZ4
AAIkrZJOtrPUqPZwYHjay3ggfgjVljj_KGhgqI5cScIzaAQs0Pxep6Fnrsn
Urw09Sjv2VRX0ay4guMQnbQo0ibpifBxeuL9MJ_vdeb_BdSE8YZ4iTfMb7G
T35gZC9NgweX3fiTEo2LjY8hEV3DHud5LlNZzYp9kLmAUZNIwGu7LtYyI4F
7NnOv9oLx1HtmfE3_skkytQoKMvMewLkIO88h325qCpWFdrLwPp63betCme
wDJPaBdrp9lrLchkXVo-d2ueKkb59TxWjMx7esBdaxCACDQ
</cmk>
<iv>
  Ggiego8UiSsj7GgY94qOng
</iv>
<data>
  4vIGDz9Hm6X4lSo9JoA6ZzS0KitztLGaiMUS3RTviFO09choPhxJNlOj8KX
  8QILu4zz-ytCnG-yzNx5SsT8KEQJhIf6_9yWplxpXl73k6ZJV-sXGd4Mj9u
  7N0IqWQLK5DMytv7XopsZsR9QFCDNGew
</data>
<mac>
  3GuaasWV0XGTBbRtNP6OQl4_cHL-ZJClnaDtU6EIecw
</mac>
</keyreq>

```

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

```

<iq xmlns='jabber:client'
  from='juliet@capulet.lit/balcony'
  id='xdJbWMA+'
  to='romeo@montegue.lit/garden'
  type='result'>
<keyreq xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
  id='835c92a8-94cd-4e96-b3f3-b2e75a438f92'>
  <encheader>
    eyJhbGciOiJSU0EtT0FFUCIsImtpZCI6InJvbWVvQG1vbnRlZ3VlLmxpdc9
    nYXJkZW4iLCJlbmMiOiJBMjU2Q0JDK0hTNTFyIiwiaWF0IjY3R5IjoieYXBwbGljYX
    Rpb24vandrk2pz24ifQ
  </encheader>
  <cmk>
    hKUOpAif76c-hmRwEphVB9wXjloLpwu75x98MSWyCBtfUgmopk93ttUXoZ4
    AAIkrZJOtrPUqPZwYHjay3ggfgjVljj_KGhgqI5cScIzaAQs0Pxep6Fnrsn
    Urw09Sjv2VRX0ay4guMQnbQo0ibpifBxeuL9MJ_vdeb_BdSE8YZ4iTfMb7G
    T35gZC9NgweX3fiTEo2LjY8hEV3DHud5LlNZzYp9kLmAUZNIwGu7LtYyI4F
    7NnOv9oLx1HtmfE3_skkytQoKMvMewLkIO88h325qCpWFdrLwPp63betCme
    wDJPaBdrp9lrLchkXVo-d2ueKkb59TxWjMx7esBdaxCACDQ
  </cmk>
  <iv>
    Ggiego8UiSsj7GgY94qOng
  </iv>
  <data>
    4vIGDz9Hm6X4lSo9JoA6ZzS0KitztLGaiMUS3RTviFO09choPhxJNlOj8KX

```

```
      8QILu4zz-ytCnG-yzNx5SsT8KEQJhIf6_9yWplxpXl73k6ZJV-sXGd4Mj9u
      7N0IqWQLK5DMytv7XopsZsR9QFCDNGew
    </data>
    <mac>
      3GuaasWV0XGTBbRtNP6OQl4_cHL-ZJClnaDtU6EIecw
    </mac>
  </keyreq>
</iq>
```

6. Multiple Operations

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

7. Inclusion and Checking of Timestamps

Timestamps are included to help prevent replay attacks. All timestamps **MUST** conform to [XEP-0082] and be presented as UTC with no offset, and **SHOULD** include the seconds and fractions of a second to three digits. Absent a local adjustment to the sending agent's perceived time or the underlying clock time, the sending agent **MUST** ensure that the timestamps it sends to the receiver increase monotonically (if necessary by incrementing the seconds fraction in the timestamp if the clock returns the same time for multiple requests). The following rules apply to the receiving agent:

- o It **MUST** verify that the timestamp received is within an acceptable range of the current time. It is **RECOMMENDED** that implementations use an acceptable range of five minutes, although implementations **MAY** use a smaller acceptable range.
- o It **SHOULD** verify that the timestamp received is greater than any timestamp received in the last 10 minutes which passed the previous check.
- o If any of the foregoing checks fails, the timestamp **SHOULD** be presented to the receiving entity (human or application) marked as "old timestamp", "future timestamp", or "decreasing timestamp", and the receiving entity **MAY** return a stanza error to the sender.

Note the foregoing assumes the stanza is received while the receiving agent is online; see Section 9 for offline storage considerations.

8. Interaction with Stanza Semantics

The following limitations and caveats apply:

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

9. Interaction with Offline Storage

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

- o If the sending agent is not also online when the message is delivered to the receiving agent from offline storage, then the decryption process fails for insufficient information as described in Section 3.3.3.
- o When performing the timestamp checks in Section 7, if the server includes delayed delivery data as specified in [XEP-0203] for when the server received the message, then the receiving agent SHOULD use the delayed delivery timestamp rather than the current time.

10. Mandatory-to-Implement Cryptographic Algorithms

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

- o MUST implement the "RSA1_5" JWE algorithm.
- o MUST implement the "RS256" JWS algorithm.

11. Security Considerations

11.1. 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.

11.2. Re-use of Session Master Keys

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

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

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

12. IANA Considerations

12.1. XML Namespaces Name for e2e Data in XMPP

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

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

Specification: RFC XXXX

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

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

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

Specification: RFC XXXX

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

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

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

Specification: RFC XXXX

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

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

13. References

13.1. Normative References

- [E2E-REQ] Saint-Andre, P., "Requirements for End-to-End Encryption in the Extensible Messaging and Presence Protocol (XMPP)", draft-saintandre-xmpp-e2e-requirements-01 (work in progress), March 2010.
- [JOSE-JWA] Jones, M., "JSON Web Algorithms (JWA)", draft-ietf-jose-json-web-algorithms-11 (work in progress), May 2013.
- [JOSE-JWE] Jones, M., Rescola, E., and J. Hildebrand, "JSON Web Encryption (JWE)", draft-ietf-jose-json-web-encryption-11 (work in progress), May 2013.
- [JOSE-JWK] Jones, M., "JSON Web Key (JWK)", draft-ietf-jose-json-web-key-11 (work in progress), December 2012.

[JOSE-JWS]

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

[JOSE-KEYPROTECT]

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

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

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

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

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

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

[XEP-0030]

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

[XEP-0082]

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

[XEP-0115]

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

[XEP-0203]

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

[XEP-0297]

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

13.2. Informative References

- [RFC3610] Whiting, D., Housley, R., and N. Ferguson, "Counter with CBC-MAC (CCM)", RFC 3923, September 2003.
- [RFC3923] Saint-Andre, P., "End-to-End Signing and Object Encryption for the Extensible Messaging and Presence Protocol (XMPP)", RFC 3923, October 2004.
- [RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", RFC 4086, June 2005.
- [XEP-0160] Saint-Andre, P., "Best Practices for Handling Offline Messages", XSF XEP 0160, January 2006.
- [XEP-0201] Saint-Andre, P., Paterson, I., and K. Smith, "Best Practices for Message Threads", XSF XEP 0203, November 2010.

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

The following XML schema is descriptive, not normative.

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

<xs:schema
  xmlns:xs='http://www.w3.org/2001/XMLSchema'
  targetNamespace='urn:ietf:params:xml:ns:xmpp-e2e:6'
  xmlns='urn:ietf:params:xml:ns:xmpp-e2e:6'
  elementFormDefault='qualified'>

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

```
    </xs:sequence>
  </xs:complexType>
</xs:element>

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

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

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

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

<xs:element name='encheader'>
  <xs:complexType>
    <xs:simpleType>
```

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

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

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

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

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

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

</xs:schema>
```

Appendix B. Acknowledgements

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

Author's Address

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

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