Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 15, 2011

R. Barnes
BBN Technologies
J. Lindberg
Google

March 14, 2011

Domain Name Assertions
draft-ietf-xmpp-dna-01

Abstract

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

Requirements Language

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

Status of this Memo

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

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

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

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

Copyright Notice

Table of Contents

1. Introduction

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

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

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

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

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

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

2. Terminology

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

We will refer to four different types of domains in this document:

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

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

3. Protocol Overview

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

```
romeo@sender.tld -- xmpp1.originating.tld
                              .
                              .
                    xmpp1.receiving.tld -- juliet@target.tld
```

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

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

When Romeo wants to send a stanza to Juliet, he will first send it to his server, xmpp1.originating.tld. Seeing that the 'to' domain of the stanza is target.tld, the server will retrieve the SRV records for _xmpp-server._tcp.target.tld, plus any associated DNSSEC records

```
[RFC4034].
_xmpp-server._tcp.target.tld. 400 IN SRV
            20 0 5269 xmpp1.receiving.tld

_xmpp-server._tcp.target.tld. 400 IN RRSIG
            SRV 5 3 400 20030322173103 (
              20030220173103 2642 _tcp.target.tld.
              oJB1W6WNGv+ldvQ3WDG0MQkg5IEhjRip8WTr
              PYGv07h108dUKGMeDPKijVCHX3DDKdfb+v6o
              B9wfuh3DTJXUAfI/M0zmO/zz8bW0Rznl8O3t
              GNazPwQKkRN20XPXV6nwwfoXmJQbsLNrLfkG
              J5D6fwFm8nN+6pBzeDQfsS3Ap3o= )
```

If there are no DNSSEC records, or if the DNSSEC records do not validate, then there is nothing new to do; the server simply connects to the remote domain using normal XMPP procedures. If there is a valid DNSSEC signature on the SRV record, then the server knows that he can allow the remote server to authenticate as either target.tld or xmpp1.receiving.tld.

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

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

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

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

When this stream is created, it can immediately carry stanzas

directly between the two servers. In order to send messages to and from other domains, the servers have to authenticate and request permission. So to send Romeo's stanza to Juliet, xmpp1.originating.tld requests permission to send from sender.tld to target.tld.

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

At this point, the server xmpp1.originating.tld knows that xmpp1.receiving.tld is authorized to represent either xmpp1.receiving.tld (via the certificate) or target.tld (via DNSSEC). The other server, xmpp1.receiving.tld knows only that the other server repressents xmpp1.originating.tld.

Once the two servers have authenticated their own names over TLS, they can request permission to send stanzas:
I: <db:result from='sender.tld' to='target.tld' />

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

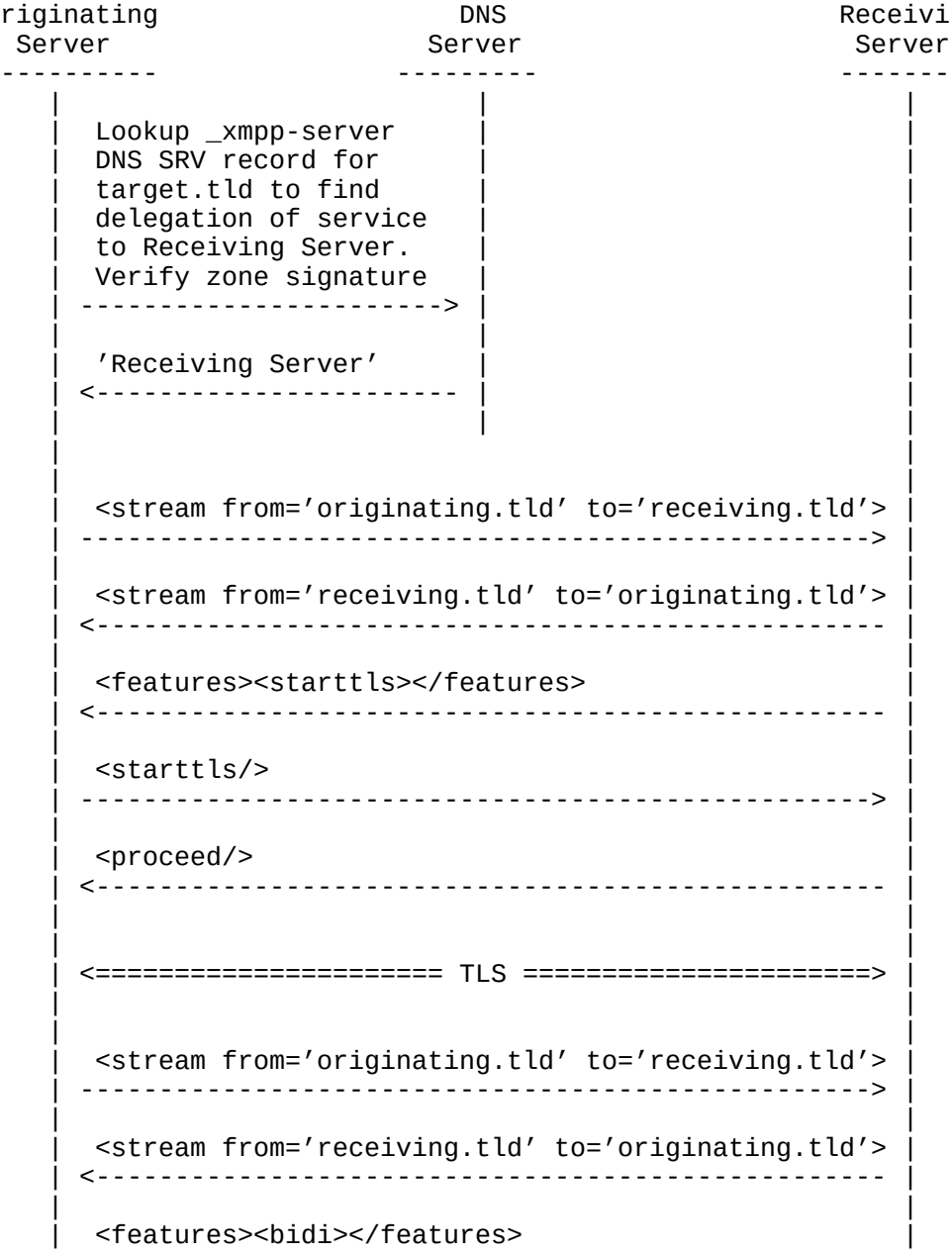
On the other hand, if the DNSSEC signature is valid, then the server can accept the request to send stanzas, and the two servers can exchange stanzas for those domains.
R: <db:result type='valid' from'sender.tld' to='target.tld' />

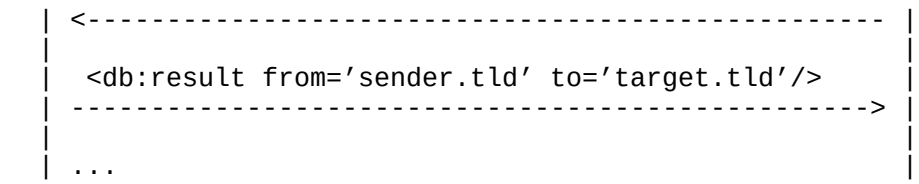
I: <!-- stanza -->

Now that the two servers have established this connection, they can re-used it for other stanzas and other domains. If either server finds another domain that is delegated to the other server, it can

send a <db:result> requesting permission to send stanzas for that domain, and the other server will grant or deny permission after checking the delegation.

The following figure summarizes the overal process:





4. Connection Model

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

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

Connection	Server Domain Names	Delegated Domain Names
XXX	xmpp1.provider.com	capulet.example
YYY	xmpp2.provider.com	capulet.example
AAA	paris.example	paris.example

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

Local	Remote	Connections
montague.example	capulet.example	XXX, YYY
laurence.example	capulet.example	AAA
laurence.example	paris.example	YYY, AAA

The binding table acts as a routing table for outgoing stanzas and a filter for incoming stanzas. When the server wishes to send a stanza, it looks in the binding table for a row that has the 'from' domain as the local domain and the 'to' domain as the remote domain. If there is such a in the binding table, then the server MUST transmit the on the first connection in the connection list. Thus, in the above example, a stanza from montague.example to capulet.example would be routed on channel XXX.

In the same way, when a server receives a stanza over a connection from a remote server, it looks up the relevant entry in the binding table, this time using the 'to' domain as the local domain and the 'from' domain as the remote domain. If the server finds a binding table entry and the connection over which the stanza arrived is listed in the entry, then it accepts the stanza. Otherwise, it MUST discard the stanza and return a stanza error <invalid-connection/>. In the above example, a stanza from capulet.example to escalus.example would be accepted on connections AAA and BBB, but no others.

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

5. Channel Establishment and Authentication

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

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

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

2. Verify that the SRV records have been signed using DNSSEC [RFC4033]. The originating server may either retrieve DNSSEC records directly or rely on a validating resolver. If the SRV records are not secured with DNSSEC, then the connection fails.

3. If there is already a connection in the connection table that has the target of any SRV record in its "server names" list, then this process terminates and the server attempts to use that connection (See Section Section 6)

4. If there is no existing connection that matches, establish a TCP connection to any of the servers listed in an SRV record and negotiate an XMPP stream with the following parameters:

   * 'from' domain: The originating server's name

   * 'to' domain: The receiving server's name from the SRV record

   * [[TODO: Add a stream feature to indicate support for this extension]]

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

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

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

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

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

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

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

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

If this process establishes a new connection, then the originating server knows that it has established a connection to a server that legitimately represents example.com. It should thus initialize a row in the connection table for this connection:

- Server names: The list of names in the server's certificate

- Delegated names: example.com

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

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

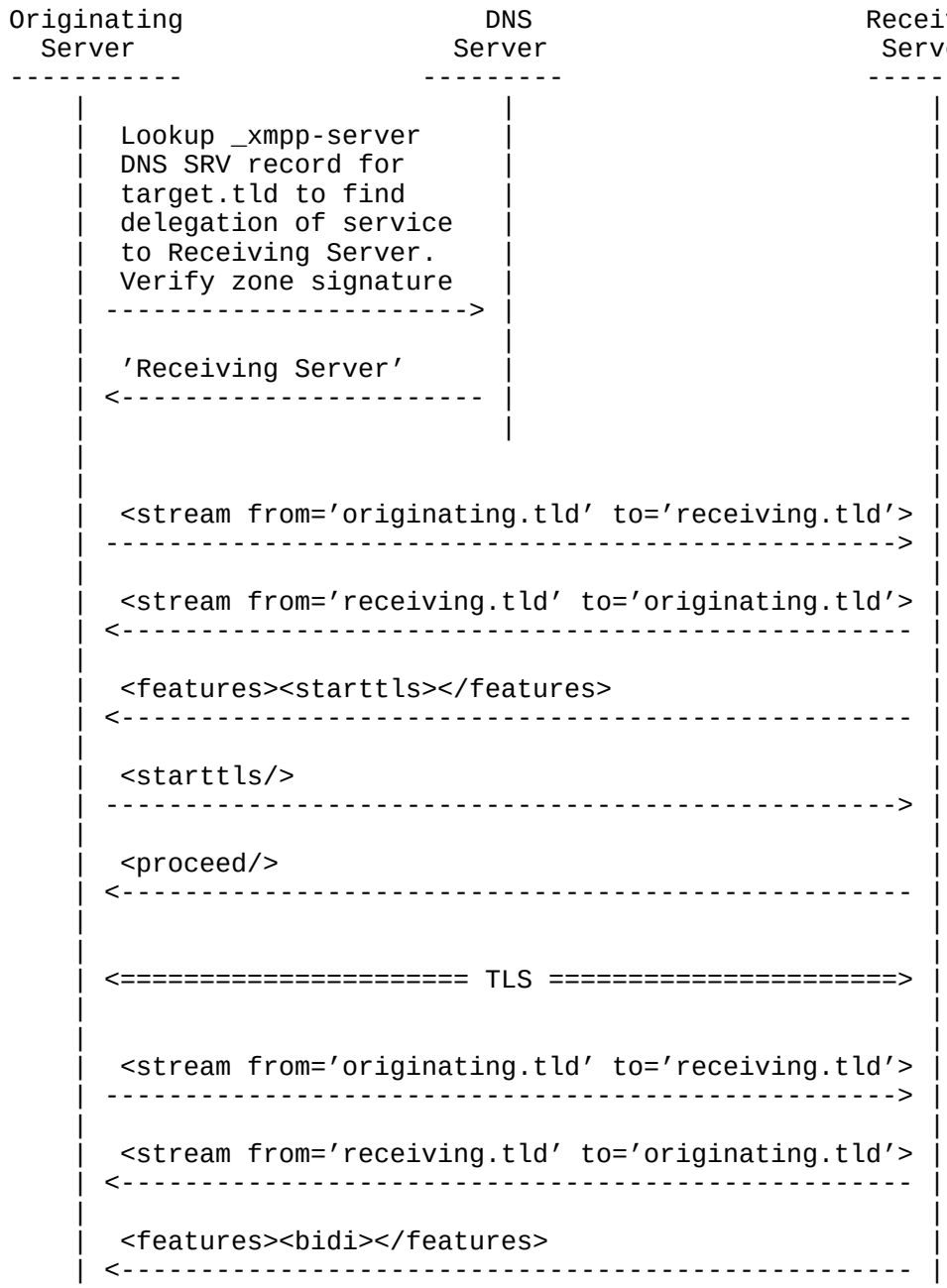
The server on the receiving end of the TLS connection MUST request a client certificate from the originating server during the TLS handshake, and the originating server MUST provide a client certificate. The receiving server can then also initialize an entry in its connection table to which delegated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

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

- Delgated names: Empty.

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

Since server-to-server connections are by default directional, it is RECOMMENDED that servers also request the <bidi> stream feature to enable bidirectional flows on this connection [XEP-0288].


Originating Server
DNS Server
Receiving Server
Lookup _xmpp-server DNS SRV record for target.tld to find delegation of service to Receiving Server. Verify zone signature
'Receiving Server'
<stream from='originating.tld' to='receiving.tld'>
<stream from='receiving.tld' to='originating.tld'>
<features><starttls></features>
<starttls/>
<proceed/>
TLS
<stream from='originating.tld' to='receiving.tld'>
<stream from='receiving.tld' to='originating.tld'>
<features><bidi></features>

6. Authorizing XMPP Stanzas

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

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

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

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

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

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

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

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

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

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

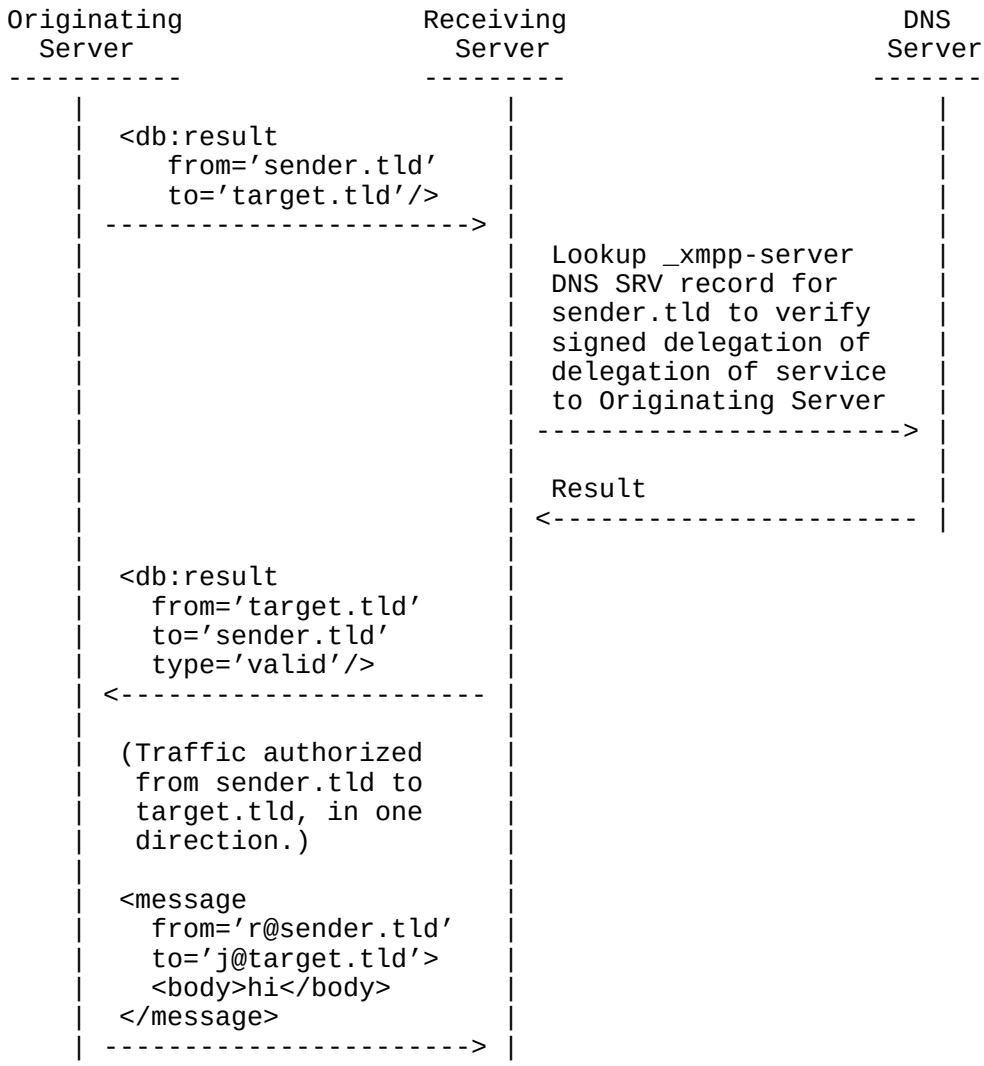
subject to local policy.

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

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

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

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



7. Backward Compatibility

Using Server Domain Names as to/from attributes in <stream> stanzas is incompatible with XMPP services that do not support this protocol, because it was previously assumed that when receiving a connection the stream to attibute will contains an XMPP domain hosted by the receiving service. It is RECOMMENDED that if the connection fails, the service tries again using the Remote Domain as stream to-attribute.

Presenting a certificate for the Server Domain Name is incompatible with XMPP services that do not support this protocol, because those will expect the Remote Domain in the certificate. It is RECOMMENDED that if the authorization fails, the service tries again presenting the certificate for the Remote Domain. A service may also choose to fall back on a weaker identification mechanism such as Server Dialback, subject to local policy.

8. Operational Considerations

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

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

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

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

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

9. IANA Considerations

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

[[Define invalid-connection error element]]

10. Security Considerations

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

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

11. Acknowledgements

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

versions of this draft.

12. Normative References

[I-D.ietf-xmpp-3920bis]
Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", draft-ietf-xmpp-3920bis-22 (work in progress), December 2010.

[I-D.saintandre-tls-server-id-check]
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)", draft-saintandre-tls-server-id-check-14 (work in progress), January 2011.

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

[RFC4033] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "DNS Security Introduction and Requirements", RFC 4033, March 2005.

[RFC4034] Arends, R., Austein, R., Larson, M., Massey, D., and S. Rose, "Resource Records for the DNS Security Extensions", RFC 4034, March 2005.

[RFC4366] Blake-Wilson, S., Nystrom, M., Hopwood, D., Mikkelsen, J., and T. Wright, "Transport Layer Security (TLS) Extensions", RFC 4366, April 2006.

[XEP-0220]
Miller, J., Saint-Andre, P., and P. Hancke, "Server Dialback", XSF XEP 0220, March 2010.

[XEP-0288]
Hancke, P. and D. Cridland, "Bidirectional Server-to-Server Connections", XSF XEP 0288, October 2010.

Authors' Addresses

Richard L. Barnes
BBN Technologies

Email: rbarnes@bbn.com

Jonas Lindberg
Google

Email: jonasl@google.com

Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: September 8, 2011

E. Rescorla
RTFM, Inc.
J. Hildebrand
Cisco Systems, Inc.

March 7, 2011

JavaScript Message Security Format
draft-rescorla-jsms-00.txt

Abstract

Many applications require the ability to send cryptographically secured messages. While the IETF has defined a number of formats for such messages (e.g. CMS) those formats use encodings which are not congenial for Web applications. This document describes a new cryptographic message format which is based on JavaScript Object Notation (JSON) and thus is easy for Web applications to generate and parse.

Status of this Memo

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

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

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

This Internet-Draft will expire on September 8, 2011.

Copyright Notice

Table of Contents

1. Introduction

Many applications require the ability to send cryptographically secured (encrypted, digitally signed, etc.) messages. While the IETF has defined a number of formats for such messages, those formats are widely viewed as being excessively complicated for the demands of Web applications, which typically only need the ability to secure simple messages. In addition, existing formats use encoding mechanisms (e.g., ASN.1 BER/DER) which are not congenial for Web applications. This presents an obstacle to the deployment of strong security by such applications.

This document describes a new cryptographic message format, JavaScript Message Security (JSMS) intended to meet the need of the Web environment. While JSMS is modeled on existing formats -- principally CMS [RFC5652] -- it uses JavaScript Object Notation (JSON) rather than ASN.1/BER/DER, making it far easier for Web applications to handle. In the interest of simplicity, JSMS also omits as many as possible of the CMS modes (multiple signatures, password-based encryption).

2. Conventions Used In This Document

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. Overview

The JSMS message format is simply a JSON [RFC4627] dictionary with an appropriate collection of fields. Each operating mode will have a separate set of fields, with a common field to distinguish between the modes.

3.1. Operational Modes

JSMS supports two operational modes:

Encrypted Data
: A block of data encrypted under a random message encryption key (MEK). The MEK is then separately encrypted for each recipient, either via symmetric or asymmetric encryption. The data is always integrity protected, either via a separate Message Authentication Code (MAC) or an Authenticated Encryption with Associated Data (AEAD) algorithm such as AES-GCM or AES-CCM.

Signed Data
: A block of data signed by a single signer using his asymmetric key and optionally carrying his certificate. Multiple signatures are not permitted in order to keep things simple.

Any other desired security functions are provided by composition of these modes. For instance, a signed and encrypted message is produced by first creating a Signed message and then encrypting that data. (See Section 4.6 for more on composition.

3.2. Conventions

In general, JSMS follows the following structural conventions:

Minimize implementation complexity
: Wherever possible, protocol choices have been made such that the time and effort required to implement the protocol in many different programming languages will be minimized. This means that optimizations for bandwidth, CPU, and memory utilization have been explicitly avoided.

Base64 as the only encoding
: Any data that does not have a straightforward string representation (binary values, large integers, etc.) is base64-encoded (see: [RFC4648]). In some cases, hexadecimal encodings might be more convenient, but consistency is even more important to reduce implementation complexity.

No canonicalization
: In many cryptographic message formats, canonical encodings are used to allow the same value to be computed at both sender and recipient (e.g., for digital signatures). This is inconvenient in JSON, which just views messages as a bundle of key/value pairs. Instead, whenever canonicalization would be required, the relevant data is serialized and base64-encoded for transport, allowing both sides to run computations over the same original set of octets.

In-memory processing
: We assume that the entire message can fit in main memory and make no effort to design a wire representation which can be handled in small chunks in a single pass. This means, for instance, that there is no need to have a message digest indicator at the beginning of the message and then the signature at the end, as is done in CMS. Fields are simply serialized in whatever order is most convenient for the JSON implementation. The examples in this document are generally shown in whatever order seems most readable and are not normative.

3.3. Certificate Processing

Experience has shown that certificate handling (path construction) is one of the trickier parts of building a cryptographic system. While JSMS supports PKIX certificates, its certificate processing is far simpler than that of CMS. When a JSMS agent provides its certificate, it must provide an ordered chain (as in TLS [RFC5246]) terminating in its own certificate, thus removing the need to construct certificate paths. The certificates MUST be ordered with the end-entity certificate first and each certificate that follows signing the certificate immediately preceding it. In addition, because many implementations will not want to do any ASN.1/BER processing at all, we will define a Web Service which applications can use for chain validation and translation to an easy-to-parse format. (See [TODO]).

3.4. Certificate Discovery

JSMS will often be used in an online messaging environment with users that have an address of the form user@domain, such as email, XMPP, or SIP. As such, protocols such as WebFinger [I-D.hammer-webfinger] or an end-to-end protocol can be used to retrieve appropriate certificates. Downstream uses of JSMS SHOULD define a discovery mechanism suitable for the intended use.

4. Message Format

All of the field definitions in this section make use of JSON Schema [I-D.zyp-json-schema]. For each of the fields that is designed to hold an enumerated value, a registry will be created allowing other values to be used in addition to the values enumerated in the schema.

4.1. Base64 Handling

As stated in section 3.1 of [RFC4648], Base64 does not require linefeeds after a specific number of characters. Since linefeeds are not valid characters in a JSON string, whenever a field is specified to be Base64-encoded in this document, it MUST NOT include any line breaks. Base64-encoded fields also MUST NOT include JSON-encoded linefeeds such as "\n". Any linebreaks in the middle of Base64-encoded sections of the examples are unintended side-effects of the production process.

Implementation Note: Much existing Base64-encoding code will generate linefeeds every 64 or 76 characters of output. Ensure that these linefeeds are removed before inserting the output into a JSON structure.

4.2. Content Object

JSMS operates by providing transformations on "Content" objects, which are just mime-typed JSON objects. These objects are then wrapped in a signed/encrypted wrapper with the following fields:

ContentType: A MIME [RFC2045] media type that MUST be included indicating the type of the "Data" field.
Type: The constant string "content", to facilitate easy determination that this is the target content. This is useful (for example) in certain operating conditions where you must continue to unwrap layers of signatures until you get to the content. This field MUST be included.
Data: The data value MUST be included as a text encoded as Base64 (See: [RFC4648]).
ID: An OPTIONAL universally unique ID that identifies this message, for use in detecting replay attacks.
Created: An OPTIONAL field describing the UTC date/time that the content was encoded into JSON, formatted according to the "date-time" production of [RFC3339].

Signing and encryption transform a "Content" object into "Signed" and "Encrypted" objects respectively. Verification and decryption transform "Signed" and "Encrypted" objects back into "Content" objects. For example:

```
{
    "ContentType":"text/plain; charset=UTF-8",
    "Type":"content",
    "Data":"SGVsbG8sIFdvcmxkCg==",
    "ID":"746a4c9f-8e84-4313-b669-81590ee2949e",
    "Created":"2011-03-07T16:17Z"
}
```

Figure 1: Content Example

4.3. Common Elements

A JSMS message is a JSON dictionary object containing a set of specific values.

The following fields MUST be present in all messages:

Version: The version number. For this specification this value MUST be set to the string "1.0". See Section 5 for details on version handling.

Type: The type of the message. MUST be either "signed" or "encrypted", to indicate a signed message (Section 4.4) or an encrypted message (Section 4.5) respectively.

4.4. Signed Data

A "signed" message contains a signed data block plus a digital signature over that data. To simplify implementation, only one signer is allowed. In addition to the required fields from Section 4.3, the fields in a signature message are:

SignedData: This field MUST consist of a Base64-encoded "Content" structure (see Section 4.2), which MUST have been encoded into octets as UTF-8 prior to Base64-encoding. The signature is computed over the UTF-8 octet stream before Base64-encoding to ensure that the sender and receiver have the exact same representation.

DigestAlgorithm: The message digest used to compute the signature. This field MUST be present for RSA-based signatures but MAY be omitted for future signatures which do not allow flexible digests. For now, this field MUST have the value "SHA-256", meaning the digest algorithm was SHA-256 [FIPS-180-3].

SignatureAlgorithm: The signature algorithm used to compute the signature. This field MUST be present. For now, this field MUST have the value "RSA-PKCS1-1.5", meaning the signature algorithm was RSASSA-PKCS1-v1_5 as specified in [RFC3447].

Signer: The signer’s identity, expressed as a URI [RFC3986]. This field MUST be present.

CertChain: The signer’s certificate chain, if any (see Section 4.4.2.1).

Signature: The Base64-encoded signature, which MUST be included (see Section 4.4.1).

```
{
    "SignedData":"ewogICAgIkNvbnRlbnRUeXBlIjoidGV4dC9wbGFpbjsgY2hhcn
                  NldD1VVEYtOCIsCiAgICAiVHlwZSI6ImNvbnRlbnQiLAogICAg
                  IkRhdGEiOiJTR1ZzYkc4c0lGZHZjbXhrQ2c9PSIsCiAgICAiSU
                  QiOiI3NDZhNGM5Zi04ZTg0LTQzMTMtYjY2OS04MTU5MGVlMjk0
                  OWUiLAogICAgIkNyZWF0ZWQiOiIyMDExLTAzLTA3VDE2OjE3Wi
                  IKfQ==",
    "DigestAlgorithm":"SHA-256",
    "SignatureAlgorithm":"RSA-PKCS1-1.5",
    "Signer":"xmpp:romeo@example.net",
    "Signature":"sNsxJltUaz4pSzAtJiPZagUMV4SwWugWexGbffK/WJRDi2uq7TxN
                 /V9SwG/kvQ7CaTABbeUuc6cKGO5YxnH5hME3bHB5L9PKPWSjxzxo
                 68RPxQyPli2YJDDHKVPbofEa86CLqYcwTF5qrcL7fQFvlRSOVxpS
                 SJfIdiAJNA+nEnk="
}
```

Figure 2: Signed Message Example

4.4.1. Signature Computation

The signature is computed over the string prior to base64 encoding. I.e., the processing order for encoding is:

1. Serialize the inner "Content" value into a UTF8-encoded octet series X.
2. Compute the signature value over X, and call the result Y. (In the case of signatures which use digests, this means feed the literal octets of the signature into the digest function.)
3. Compute the Base64 representation of X and insert it into the "SignedData" field of the message.
4. Compute the Base64 representation of Y, and insert the result into the "Signature" field.

This procedure removes dependencies on the exact serialization algorithm; variation in spacing, field order, etc. do not affect signature validity since the Base64 representation preserves them on the wire and protects them from modification by intermediaries.

Note: An alternative algorithm would be to compute the signature on the base64 representation itself, but this has two disadvantages: (1) any intermediaries which change spacing/line breaks would break the signature. (2) it is inconsistent with the algorithm for encryption (Section 4.5), which is designed to avoid multiple base64 encoding.

This procedure only specifies the input to the signature computation. The details of the computation depend on the signature algorithm itself. The mapping from code points to algorithms is found in

Section 6.

4.4.2. Signature Verification

In order to verify the signature, the steps of the previous section are reversed.

1. Process the provided "Signer" and "CertChain" fields as described in Section 4.4.2.1 in order to determine the sender's public key.
2. Base64 decode the "SignedData" field in order to recover a string X.
3. Verify the "Signature" field against X using the sender's public key and the "SignatureAlgorithm" and "DigestAlgorithm" fields. If the signature fails, return an error.
4. Deserialize X to recover the inner "Content" value.
5. Check any "ID" or "Created" fields for replay.
6. Using the value of the "ContentType" field to give MIME type context, Base64-decode the "Data" field to retrieve the intended message.

4.4.2.1. Certificate Processing

JSMS uses the "CertChain" element to carry certificate chains. For the moment, each certificate in the chain is expected to be a PKIX certificate BER-encoded then Base64-encoded. Future versions of this document will likely specify other valid certificate formats, since one of the goals of this format is to avoid . The meaning of the fields is described below:

Type: The type of the certificate chain. The only defined value is "PKIX", referring to PKIX [RFC5280] certificates.

Chain: An array of certificate values. In the case of "PKIX" certificates this is a list of base64-encoded DER/BER PKIX certificate values. PKIX certificates MUST be represented in order with each certificate certifying the next and the final certificate representing the end-entity.

```
{
    "Type":"PKIX",
    "Chain":[
        "MIICPjCCAaegAwIBAgIBETANBgkqhkiG9w0BAQUFADBDMRMwEQ
         YKCZImiZPyLGQBGRYDY29tMRcwFQYKCZImiZPyLGQBGRYHZXhh
         bXBsZTETMBEGA1UEAxMKRXhhbXBsZSBDQTAeFw0wNDA0MzAxND
         I1MzRaFw0wNTA0MzAxNDI1MzRaMEMxEzARBgoJkiaJk/IsZAEZ
         FgNjb20xFzAVBgoJkiaJk/IsZAEZFgdleGFtcGxlMRMwEQYDVQ
         QDEwpFeGFtcGxlIENBMIGfMA0GCSqGSIb3DQEBAQUAA4GNADCB
         iQKBgQDC15dtKHCqW88jLoBwOe7bb9Ut1WpPejQt+SJyR3Ad74
         DpyjCMAMSabltFtG6l5myUDfqR6UD8JZ3Ht2gZVo8RcGrX8ckR
         Tzp+P5mNbnaldF9epFVT5cdoNlPHHTsSpoX+vW6hyt81UKwI17
         m0flz+4qMs0SOEqpjAm2YYmmhH6QIDAQABo0IwQDAdBgNVHQ4E
         FgQUCGivhTPIOUp6+IKTjnBqSiCELDIwDgYDVR0PAQH/BAQDAg
         EGMA8GA1UdEwEB/wQFMAMBAf8wDQYJKoZIhvcNAQEFBQADgYEA
         bPgCdKZh4mQEplQMbHITrTxH+/ZlE6mFkDPqdqMm2fzRDhVfKL
         fvk7888+I+fLlS/BZuKarh9Hpv1X/vs5XK82aIg06hNUWEy7yb
         uMitxV5G2QsOjYDhMyvcviuSfkpDqWrvimNhs25HOL7oDaNnXf
         P6kYE8krvFXyUl63zn2KE=",
        "MIICcTCCAdqgAwIBAgIBEjANBgkqhkiG9w0BAQUFADBDMRMwEQ
         YKCZImiZPyLGQBGRYDY29tMRcwFQYKCZImiZPyLGQBGRYHZXhh
         bXBsZTETMBEGA1UEAxMKRXhhbXBsZSBDQTAeFw0wNDA5MTUxMT
         Q4MjFaFw0wNTAzMTUxMTQ4MjFaMEMxEzARBgoJkiaJk/IsZAEZ
         FgNjb20xFzAVBgoJkiaJk/IsZAEZFgdleGFtcGxlMRMwEQYDVQ
         QDEwpFbmQgRW50aXR5MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCB
         iQKBgQDhauQDMJcCPPQQ87UeTX8Ue/b10HjppIrwo3Xs7bZWln
         +ImYWa8j5od4frntGfwLQX3KuJI6QdfhYjTE+oTfUxuHyq4xpJ
         CfRLJtsnZzCCEgFK6Rq2wQxTi2z8L3pD7DM2fjKye9WqzwEUxh
         LsE/ItFHqLIVgUE0xGo5ryFpX/IwIDAQABo3UwczAhBgNVHREE
         GjAYgRZlbmQuZW50aXR5QGV4YW1wbGUuY29tMB0GA1UdDgQWBB
         QXe5Iw/0TWZuGQECJsFk/AjkHdbTAfBgNVHSMEGDAWgBQIaK+F
         M8g5Snr4gpOOcGpKIIQsMjAOBgNVHQ8BAf8EBAMCBsAwDQYJKo
         ZIhvcNAQEFBQADgYEAACAoNFtoMgG7CjYOrXHFlRrhBM+urcdi
         FKQbNjHA4gw92R7AANwQoLqFb0HLYnq3TGOBJl7SgEVeM+dwRT
         s5OyZKnDvyJjZpCHm7+5ZDd0thi6GrkWTg8zdhPBqjpMmKsr9z
         1E3kWORi6rwgdJKGDs6EYHbpc7vHhdORRepiXc0="
    ]
}
```

Figure 3: PKIX CertChain Example

The recipient MUST verify the certificate chain (in the case of PKIX certificates according to [RFC5280]). If any validation failure occurs, the implementation MUST abort processing and return an error.

Once the certificate chain is validated, the end-entity certificate must contain an identity which matches the "Signer" field. In the case of PKIX certificates, the certificate MUST contain a

subjectAltName field of type "uniformResourceIdentifier". This field MUST be equivalent to the URI in the "Signer" field. If not, an error MUST be returned.

4.5. Encrypted Data

An "encrypted" message contains an encrypted "Content" block. All "encrypted" messages contain a symmetric integrity check, either via a MAC or via an AEAD [RFC5116] algorithm such as Galois/Counter Mode (GCM: [GCM]). A message may be encrypted to an arbitrary number of recipients. Each recipient is represented by a "Recipient" block, which contains a copy of the keying material encrypted for that recipient. Both symmetric and asymmetric key establishment is supported. In order to support both integrity and encryption, what is carried in the Recipient block is a Content Master Key (CMK) which is then used with a Key Derivation Function (KDF) to generate the Content Encryption Key (CEK) used to encrypt the message and the Content Integrity Key (CIK) used with the MAC. In addition to the required fields from Section 4.3 the fields in an encrypted message are:

Recipients: The list of recipients. This is an array of Recipient objects, each of which establishes the CMK for that recipient.

KDF: Specifies the key derivation function used to generate the CEK and the CIK from the CMK. This field MAY be absent if an AEAD algorithm is used, in which case the CEK is derived by copying the CMK.

Encryption: Specifies the properties of the encryption. The Algorithm field MUST contain the encryption algorithm and the IV field specifies the initialization vector (if required for the algorithm). This field MUST be present.

Integrity: Specifies the properties of the integrity check. The Algorithm field MUST contain the MAC algorithm and the Value field MUST contain the MAC. This field MAY be absent if no integrity check is used.

Data: Contains the ciphertext.

Each Recipient object provides an encrypted copy of the CMK for a single recipient. The meaning of the fields is described below:

KEKidentifier Describes the key encrypting key (KEK) used to encrypt the CMK. Either a "RecipientName" or a "KeyIdentifier" MUST be provided. If the "RecipientName" is provided, then a "CertificateDigest" SHOULD be provided.

RecipientName: Provides the recipient's name in URI form.
CertificateDigest: For now, the SHA-1 fingerprint of the PKIX certificate associated with the recipient.
KeyIdentifier The name of a shared symmetric key known to both sender and recipient. This need not be globally unique as long as it is unique within the recipient's context.

Algorithm: The algorithm used to encrypt the CMK. For now, one of "RSA-PKCS1-1.5" (meaning RSASSA-PKCS1-v1_5 as specified in [RFC3447]) or "AES-256-CBC" (meaning [FIPS-180-3]). Note the JSMS only supports key transport and not key agreement (since key agreement can always be turned into key transport).

Value: The CMK encrypted under the specified algorithm and key.

4.5.1. Message Encryption

The message encryption process is as follows.

1. Generate a random CMK. The CMK MUST have a length at least equal to that of the larger of the required integrity or encryption keys and MUST be generated randomly. See [RFC4086] for considerations on generating random values. [[TODO - we need a section on generating randomness in browsers - it's easy to screw up]]
2. Encrypt the CMK for each recipient (see Section 4.5.4)
3. Generate a random IV (if required for the algorithm).
4. Run the key derivation algorithm (see Section 4.5.3) to generate the CEK and CIK (if not using an AEAD algorithm).
5. Serialize the content into a bitstring M.
6. Encrypt M using the CEK and IV to form the bitstring C.
7. Set the Value element equal to the base64-encoded representation of C.
8. If not using an AEAD algorithm, compute the function I = MAC(CIK, C) using the chosen integrity algorithm. Note that this is EtA encryption which is considered the best cryptographic choice (See: [krawczyk-ate]). Set the Integrity.Value element equal to the base64-encoded representation of I.

4.5.2. Message Decryption

The message decryption process is the reverse of the encryption process.

1. Identify a Recipient block which appears to reference a key known to the recipient.
2. Decrypt the CMK. If this fails and another Recipient block appears plausible, that MAY be tried.

3. Run the key derivation algorithm (see Section 4.5.3) to generate the CEK and CIK (if not using an AEAD algorithm).
4. If not using an AEAD algorithm, compute the integrity check value I' on the binary representation of the Value element using the indicated integrity check. If the Integrity.Value does not match I', then an error MUST be reported and processing MUST be aborted.
5. Decrypt the binary representation of the Value element and output the result

4.5.3. Key Derivation

The key derivation process converts the CMK into a CEK. It assumes as a primitive a Key Derivation Function (KDF) which notionally takes three arguments:
MasterKey: The master key used to compute the individual use keys
Label: The use key label, used to differentiate individual use keys
Length: The length of the desired use key
The only real KDF specified in this document is the TLS PRF, which is invoked as PRF(MasterKey, Label) with an empty seed and produces an arbitrary length output. The appropriate number of bits (Length) is simply extracted from the beginning of the output. The KDF name "P_XXX" in this document refers the the TLS [RFC5246] PRF using P_XXX as the underlying P_hash function.

To compute the CEK from the CMK, the label "Encryption" is used.

To compute the CIK from the CMK, the label "Integrity" is used.

When AEAD algorithms are used the KDF element MUST NOT be present. When they are not used, it MUST be present.

4.5.4. CMK Encryption

JSMS supports two forms of CMK encryption:

- Asymmetric encryption under the recipient's public key.
- Symmetric encryption under a shared key.

4.5.4.1. Asymmetric Encryption

In the asymmetric encryption mode, the CMK is encrypted under the recipient's public key. The only currently defined asymmetric encryption mode is RSA-PKCS1-1.5, which refers to [RFC3447] RSAES-PKCS1-v1_5.

4.5.4.2. Symmetric Encryption

In the symmetric encryption mode, the CMK is encrypted under a symmetric key shared between the sender and receiver. All such modes MUST provide integrity for the CMK. This document defines four such modes: AES-128-CBC, AES-256-CBC referring to the [RFC5649] AES key wrapping modes and AES-128-GCM, AES-256-GCM, referring to AES encryption with GCM. For GCM the random 64-bit IV is prepended to the ciphertext.

4.6. Composition

This document does not specify a combination signed and encrypted mode. However, because the contents of a message can be arbitrary, and encryption and data origin authentication can be provided by recursively encapsulating multiple JSMS messages. In general, senders SHOULD sign the message and then encrypt the result (thus encrypting the signature). This prevents attacks in which the signature is stripped, leaving just an encrypted message, as well as providing privacy for the signer.

5. Version Processing

For the moment, all version numbers in the protocol MUST be 1.0. Receivers MUST return an error for any other version number. More interesting version processing will be defined in the future.

6. IANA Considerations

[TODO]
- Register MIME types
- Registries for signature, encryption, MAC
- Well known HTTP URLs

7. Security Considerations

Much more to follow here.

8. References

8.1. Normative References

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message

Bodies", RFC 2045, November 1996.

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

[RFC3339] Klyne, G., Ed. and C. Newman, "Date and Time on the Internet: Timestamps", RFC 3339, July 2002.

[RFC3447] Jonsson, J. and B. Kaliski, "Public-Key Cryptography Standards (PKCS) #1: RSA Cryptography Specifications Version 2.1", RFC 3447, February 2003.

[RFC4086] Eastlake, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, June 2005.

[RFC4627] Crockford, D., "The application/json Media Type for JavaScript Object Notation (JSON)", RFC 4627, July 2006.

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

[RFC5116] McGrew, D., "An Interface and Algorithms for Authenticated Encryption", RFC 5116, January 2008.

[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.

[RFC5649] Housley, R. and M. Dworkin, "Advanced Encryption Standard (AES) Key Wrap with Padding Algorithm", RFC 5649, September 2009.

[I-D.zyp-json-schema]
Zyp, K. and G. Court, "A JSON Media Type for Describing the Structure and Meaning of JSON Documents", draft-zyp-json-schema-03 (work in progress), November 2010.

[FIPS-180-3]
National Institute of Standards and Technology (NIST), "Secure Hash Standard (SHS)", FIPS PUB 180-3, October 2008.

8.2. Informative References

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005.

[I-D.hammer-webfinger] Hammer-Lahav, E., Fitzpatrick, B., and B. Cook, "The WebFinger Protocol", draft-hammer-webfinger-00 (work in progress), October 2009.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[krawczyk-ate] Krawczyk, H., "The Order of Encryption and Authentication for Protecting Communications (or: How Secure Is SSL?)", Advances in cryptology--CRYPTO 2001 August 2001.

[GCM] National Institute of Standards and Technology (NIST), "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", SP 800-38D, November 2007.

Appendix A. JSON Schema

The following schemas formally define various namespaces used in this document, in conformance with [I-D.zyp-json-schema]. Because validation of JSON documents is optional, these schemas are not normative and are provided for descriptive purposes only.

A.1. Message Contents Schema

```
{
    "description":"Message Contents",
    "type":"object",
    "properties":{
        "ContentType":{
            "description":"A MIME content type",
            "type":"string",
            "required":true
        },
        "Type":{
            "description":"Dictionary type",
            "type":"string",
            "enum":["content"],
            "required":true
        },
        "Data":{
            "description":"The underlying data",
            "type":"string",
            "required":true
        },
        "ID":{
            "description":"(optional) unique ID for this message",
            "type":"string"
        },
        "Created":{
            "description":"(optional) time the message was created",
            "type":"string",
            "format":"date-time"
        }
    }
}
```

A.2. Common Elements Schema

```
{
    "description":"The basic schema for a JSMS message",
    "type":"object",
    "properties":{
        "Type":{
            "description":"Message type",
            "type":"string",
            "enum":["signed", "encrypted"]
        },
        "Version":{
            "description":"Version number for the message",
            "type":"string",
            "enum":["1.0"]
        }
    }
}
```

A.3. Signed Message Schema

```
{
    "description":"A signed message",
    "type":"object",
    "extends":message_schema,
    "properties":{
        "Signature":{
            "description":"The signature over the SignedData",
            "type":"object",
            "properties":{
                "SignedData":{
                    "description":"content to be signed, Base64",
                    "type":"string",
                    "required":true
                },
                "DigestAlgorithm":{
                    "description":"",
                    "type":"string",
                    "enum":["SHA-256"]
                },
                "SignatureAlgorithm":{
                    "description":"",
                    "type":"string",
                    "enum":["RSA-PKCS1-1.5"]
                },
                "Signer":{
                    "description":"",
                    "type":"string",
                    "format":"uri",
                    "required":true
                },
                "CertChain": {
                    "description":"the signer's cert chain",
                    "type":"PKIXcertchain"
                },
                "Signature":{
                    "description":"the signature",
                    "type":"string",
                    "required":true
                }
            }
        }
    }
}
```

A.4. PKIX Certificate Chain Schema

```
{
    "description":"A chain of PKIX certificates",
    "id":"PKIXcertchain",
    "properties":{
        "Type":{
            "description":"The type of certificate chain",
            "type":"string",
            "enum":["PKIX"] },
        "Chain":{
            "description":"PKIX certs ordered from root to end",
            "type":"array",
            "items":{
                "description":"A base64-encoded BER certificate",
                "type":"string"
            }
        }
    }
}
```

A.5. Encrypted Message Schema

```
{
    "description":"An encrypted object",
    "type":"object",
    "extends":message_schema,
    "properties":{
        "Recipients":{
            "description":"The list of recipient blocks",
            "type":"array",
            "required":true,
            "items":{
                "description":"A single recipient block",
                "type":"Recipient"
            }
        },
        "KDF":{
            "description":
            "The KDF used to derive the MAC and encryption keys",
            "type":"string",
            "enum":["P_SHA256"]
        },
        "Encryption":{
            "description":"Encryption control information",
            "type":"object",
            "required":true,
            "properties":{
```

```
                    "Algorithm":{
                        "description":"The algorithm used to encrypt",
                        "type":"string",
                        "enum":["AES-256-CBC"]
                    },
                    "IV":{
                        "description":"Initialization vector (base64)",
                        "type":"string"
                    }
                }
            },
            "Integrity":{
                "description":"The integrity control information",
                "type":"object",
                "properties":{
                    "Algorithm":{
                        "description":"The MAC algorithm",
                        "type":"string",
                        "enum":["HMAC-SHA-256"]
                    },
                    "Value":{
                        "description":"The MAC value (base64-encoded)",
                        "type":"string",
                        "required":true
                    }
                }
            },
            "Data":{
                "description":"The ciphertext (Base64-encoded)",
                "type":"string",
                "required":true
            }
        }
    }
```

A.6. Recipient Schema

```
{
    "description":"The recipient of an encrypted object",
    "type":"object",
    "id":"Recipient",
    "properties":{
        "KEKidentifier":{
            "type":"object",
            "description":"Identifies the key encrypting key",
            "properties":{
                "RecipientName":{
                    "type":"string",
                    "description":"The recipient's name",
                    "format":"uri"
                },
                "CertificateDigest":{
                    "type":"string",
                    "description":"Recipient's cert fingerprint"
                },
                "KeyIdentifier":{
                    "type":"string",
                    "description": "Shared symmetric key (opaque)"
                }
            }
        },
        "Algorithm":{
            "description":"The algorithm used to protect the CMK",
            "type":"string",
            "enum":["RSA-PKCS1-1.5", "AES-256-CBC"]
        },
        "Value":{
            "description": "Base64 of the encrypted CMK",
            "type":"string"
        }
    }
}
```

Appendix B. Acknowledgments

[TODO]

Authors' Addresses

Eric Rescorla
RTFM, Inc.
2064 Edgewood Drive
Palo Alto, CA 94303
USA

Email: ekr@rtfm.com

Joe Hildebrand
Cisco Systems, Inc.
1899 Wyknoop Street, Suite 600
Denver, CO 80202
USA

Email: jhildebr@cisco.com

Network Working Group
Internet-Draft
Intended status: Informational
Expires: September 15, 2011

P. Saint-Andre
Cisco

March 14, 2011

Internationalized Addresses in XMPP
draft-saintandre-xmpp-i18n-03

Abstract

The Extensible Messaging and Presence Protocol (XMPP) as defined in RFC 3920 used stringprep in the preparation and comparison of non-ASCII characters within XMPP addresses. This document explores a post-stringprep approach to XMPP addresses.

Status of this Memo

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

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

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

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

Table of Contents

1. Introduction

The Extensible Messaging and Presence Protocol [RFC6120] is a widely-deployed technology for real-time communication, commonly used for instant messaging (IM) among human users but also for communication among automated systems. XMPP addresses (also called "JabberIDs" or JIDs) are of the form <localpart@domainpart/resourcepart>, where the localpart and resourcepart are formally optional but quite common because they are used to identify clients and other entities on the network. In some sense, XMPP addresses have always been internationalized, because the developers of the original Jabber open-source project intended that all data sent over the wire would consist of UTF-8 encoded Unicode code points. However, at that time (1999) the Jabber developers were quite unsophisticated about internationalization, nor could they simply re-use a reliable internationalization technology that had been developed by the wider Internet community (as they could, for example, by re-using Secure Sockets Layer and Transport Layer Security for channel encryption); this lack of sophistication is evident in the community's first attempt at formally defining the format for JabberIDs in early 2002 [XEP-0029].

When the first instantiation of the IETF's XMPP WG was formed in late 2002, IDNA2003 [RFC3490] had not yet been published and stringprep [RFC3454] was a new technology. During its work on [RFC3920], the XMPP WG absorbed as best it could the advice of internationalization experts regarding appropriate methods for preparing and comparing XMPP addresses; however, the participants in the XMPP WG were ignorant of internationalization and therefore did not necessarily make fully-informed decisions. As a result of this early work, in [RFC3920] the XMPP WG decided to re-use IDNA2003 [RFC3490] and Nameprep [RFC3491] for the domainpart of a JID and to define two additional stringprep profiles: Nodeprep for the localpart and Resourceprep for the resourecepart.

Since the publication of [RFC3920] in 2004, the Internet community has gained more experience with internationalization. In particular, IDNA2003, which is based on stringprep, has been superseded by IDNA2008 ([RFC5890], [RFC5891], [RFC5892], [RFC5893], [RFC5894]), which does not use stringprep. This migration away from stringprep for internationalized domain names has prompted other "customers" of stringprep to consider new approaches to the preparation and comparison of internationalized addresses. As a result, the IETF has formed the PRECIS WG as a common forum for seeking solutions to the problem statement outlined in [PROBLEM].

This document has two purposes: (1) provide input to the PRECIS WG and (2) help inform the decisions of the XMPP WG regarding

internationalization of XMPP addresses, eventually leading to replacement of [RFC6122]. Note well that so far this document present only the author's opinions, and that it does not reflect the consensus of the XMPP WG or the PRECIS WG.

2. Proposed PRECIS String Classes

Both [PROBLEM] and [FRAMEWORK] propose that it might be valuable to think of internationalized addresses in terms of broad "string classes". Application technologies like XMPP could either borrow such a string class unchanged or "profile" such a string class with modifications.

This document does not yet make recommendations about borrowing or adapting more general string classes, in part because those classes are not yet clearly defined. However, as input to further discussion, this document explores four string classes that are used in XMPP:

- Domain names. These are defined in IDNA specification and re-used in XMPP and other applications. However, additional guidelines might be helpful for applications (or at least for XMPP) to fill the gap between what was provided in IDNA2003 (such as normalization and various mapping steps) and what is now provided in IDNA2008. For consistency with the next three string classes we call these "domaineythings".

- Username-like things. Such a "nameything" is a word or set of words that is used to identify or address a network entity such as a user, an account, a venue (e.g., a chatroom), an information source (e.g., a feed), or a collection of data (e.g., a file). An XMPP localpart is a kind of nameything, but might profile a base definition of nameythings developed by the PRECIS WG.

- Password-like things. Such a "wordything" is a sequence of letters, numbers, and symbols that is used as a secret for access to some resource on a network (e.g., an account or a venue). In XMPP, wordythings are often used by clients to authenticate with servers, as provided in various SASL mechanisms.

- Free-form things. Such a "stringything" is a sequence of letters, numbers, symbols, spaces, and other code points that is used for more expressive purposes in an application protocol. An XMPP resourcepart is a kind of stringything, but might profile a base definition of stringythings developed by the PRECIS WG.

The following subsections discuss these string classes in more

detail, with reference to the properties described in Section 3 of [PROBLEM] (input restrictions, normalization, case mapping, and bidirectionality).

2.1. Domaineythings

The IDNA2008 protocol is defined in [RFC5890], [RFC5891], [RFC5892], [RFC5893], and [RFC5894]. However, IDNA2008 covers a smaller range of topics than IDNA2003 [RFC3490]. In particular, normalization and mappings are out of scope for IDNA2008 (although one possible approach is described informationally in [RFC5895]). The XMPP WG, or even the PRECIS WG, might want to choose a normalization form and a set of mappings that would be recommended or required for use on the wire, despite the fact that these matters were not specified in a normative way for IDNA2008. This is especially important in modern application protocols that communicate using UTF-8-encoded Unicode code points instead of 8-bit or 7-bit ASCII (as in older application protocols such as [RFC5322]).

2.2. Nameythings

Most application technologies need a special class of strings that can be used to include or communicate things like usernames, chatroom names, file names, and data feed names. We group such things into a bucket called "nameythings". Ideally, the PRECIS WG would define a "nameything" class that could be profiled by various application technologies. We suggest that the base class would have the following features:

- Control characters (e.g., U+0000 through U+001F) would be disallowed.
- Space characters (U+0020, along with any code point having a GeneralCategory of Zs) would be disallowed.
- All other 7-bit ASCII characters (i.e., U+0021 through U+007E) would be protocol-valid, even if their Unicode GeneralCategory is disallowed by the rules specified below.
- As with IDNA2008, any character that has a compatibility equivalent would be disallowed.
- Uppercase and titlecase code points would be mapped to their lowercase equivalents.
- The normalization form would be NFD (see below).
- Profiles of the base class would be able to exclude specific code points that are included in the base.
- Profiles of the base class would be able to exclude character classes with other properties (e.g., math symbols) that are included in the base.

OPEN ISSUE: Should symbol characters outside the 7-bit ASCII range be

disallowed?

OPEN ISSUE: How to handle right-to-left code points? It might be reasonable to simply use the "Bidi Rule" from [RFC5893], however "." is allowed in nameythings and the Bidi Rule is probably too complex for our purposes because domaineythings have internal structure (based around the "." character) whereas nameythings do not.

2.3. Wordythings

Many application technologies need a special class of strings that can be used to communicate secrets that are typically used as passwords or passphrases. We group such things into a bucket called "wordythings". Ideally, the PRECIS WG would define a "wordything" class that could be profiled by various application technologies. We suggest that the base class would have the following features:

- Control characters (e.g., U+0000 through U+001F) would be disallowed.
- Space characters (U+0020, along with any code point having a GeneralCategory of Zs) would be disallowed.
- All other 7-bit ASCII characters (i.e., U+0021 through U+007E) would be protocol-valid, even if their Unicode GeneralCategory is disallowed by the rules specified below.
- Any character that has a compatibility equivalent would be disallowed.
- In order to maximize the entropy of passwords and passphrases, uppercase and titlecase code points would be protocol-valid and would not be mapped to their lowercase equivalents.
- The normalization form would be NFD (see below).
- Profiles of the base class would be able to exclude specific code points that are included in the base.
- Profiles of the base class would be able to exclude character classes with other properties (e.g., math symbols) that are included in the base.

Although some application protocols use passwords and passphrases directly, others re-use technologies that themselves use passwords in some deployments (e.g., this is true of XMPP, which re-uses Simple Authentication and Security Layer or SASL [RFC4422]).

2.4. Stringythings

Some application technologies need a special class of strings that can be used in a free-form way. We group such things into a bucket called "stringythings". Ideally, the PRECIS WG would define a "stringything" class that could be profiled by various application technologies. We suggest that the base class would have the

following features:

- Control characters (e.g., U+0000 through U+001F) would be disallowed.
- Space characters (U+0020, along with any code point having a GeneralCategory of Zs) would be protocol-valid.
- All other 7-bit ASCII characters (i.e., U+0021 through U+007E) would be protocol-valid, even if their Unicode GeneralCategory is disallowed by the rules specified below.
- Characters with compatibility equivalents would be protocol-valid.
- Uppercase and titlecase code points would protocol-valid and would not be mapped to their lowercase equivalents.
- The normalization form would be NFD (see below).
- Profiles of the base class would be able to exclude specific code points that are included in the base.
- Profiles of the base class would be able to exclude character classes with other properties (e.g., math symbols) that are included in the base.

OPEN ISSUE: How to handle right-to-left code points? It might be reasonable to simply use the "Bidi Rule" from [RFC5893], however "." is allowed in stringythings and the Bidi Rule is probably too complex for our purposes because domaineythings have internal structure (based around the "." character) whereas stringythings do not.

3. Normalization

Following IDNA2003, existing stringprep profiles all use Unicode Normalization Form KC (NFKC), which performs canonical decomposition and compatibility decomposition, followed by canonical and compatibility recomposition (regarding normalization forms, see [UAX15]). This choice made sense in IDNA2003 because the DNS packet format has fixed-length labels, and NFKC in effect compresses a sequence of characters into the smallest number of bytes possible by performing recomposition. However, experience with some of the application protocols that are currently using NFKC has shown that recomposition is an expensive operation to perform in application servers. In addition, the application protocols that use stringprep all use TCP with security-layer or application-layer compression, so fixing the length of strings is much less important.

What matters most in application protocols is ensuring that network entities (such as clients and servers) all communicate a consistent string representation over the wire. For this purpose, Normalization Form D (NFD), which simply performs canonical decomposition, provides the most efficient approach. As noted above, we can disallow any characters that would require compatibility decomposition, thus

removing the need for compatibility decomposition and recomposition. This is what happened in IDNA2008, enabling IDNA technologies to move from NFKC to NFC. If the same basic approach is taken in the PRECIS WG, while at the same time removing the need for recomposition entirely (by making code points with compatibility equivalents), NFKC (the most complex and therefore most computationally intensive normalization form) can be replaced with NFD (the least complex and therefore least computationally intensive normalization form). Another relevant factor is that NFD(x) = NFD(NFD(x)), which means that application servers can be optimized for the case where the normalization has already occurred. In general, using NFD will likely result in significant performance improvements within application servers.

4. Subclassing

The opportunity for subclassing PRECIS string classes opens the possibility that different applications technologies will subclass a given class in different ways. For example, imagine that the XMPP community defines a detailed subclass of "nameything" that is optimized for the comparison of JabberIDs. However, the email community might do the same for email addresses. At that point, the XMPP comparison methods might diverge significantly from the mail comparison methods, leading to interoperability problems if a given deployment makes use of the same usernames for both JabberIDs and email addresses. The PRECIS WG needs to consider these matters and find a productive balance between compatibility within an application technology and interoperability across application technologies.

5. XMPP Use of PRECIS String Classes

5.1. Localpart

The localpart of an XMPP address would be redefined as a profile or subclass of the PRECIS "nameything" class. The following additional restrictions would apply:

- Space characters (U+0020, along with any code point having a GeneralCategory of Zs) would be disallowed.
- The following Unicode code points would be disallowed: U+0022 ("), U+0026 (&), U+0027 (’), U+002F (/), U+003A (:), U+003C (<), U+003E (>), U+0040 (@).

OPEN ISSUE: Should symbol characters outside the 7-bit ASCII range be disallowed?

5.2. Resourcepart

The resourcepart of an XMPP address would be redefined as a profile or subclass of the PRECIS "stringything" class, or might even simply use the identity subclass of "stringything".

6. XMPP Migration Issues

Any move away from Nameprep, Nodeprep, and Resourceprep as they are defined today will inevitably introduce the potential for migration issues, such as JIDs that were not ambiguous before the migration but that become ambiguous after the migration. These issues need to be clearly defined and well understood so that the costs and benefits of any change can be properly assessed -- especially if the change might have an impact on authentication (e.g., as described in [RFC3920]), authorization (e.g., presence subscriptions as described in [RFC6121]), access (e.g., joining a chatroom as described in [XEP-0045]), identification (e.g., in XMPP URIs or IRIs as described in [RFC5122]), and other security-related functions.

7. XMPP Protocol Slots

IDNA2008 defined the concept of a "domain name slot", i.e., "a protocol element or a function argument or a return value (and so on) explicitly designated for carrying a domain name" (Section 2.3.2.6 of [RFC5890]). Similarly, the XMPP community can define the concepts of a "JID slot", a "localpart slot", and a "resourcepart slot" (and might re-use the concepts of a "nameything slot", "wordything slot", and "stringything slot" from PRECIS specifications). The community has yet to determine the full inventory of such slots. However, the following subsections provide a start at such an inventory.

7.1. JID Slot

In XMPP systems, JabberIDs can appear in at least the following slots:

- Core [RFC6120]: the 'from' and 'to' stream attributes; the 'from' and 'to' stanza attributes.
- IM [RFC6121]: the 'jid' attribute of the roster <item/> element.
- Privacy Lists [RFC3921], [XEP-0016]: the 'value' attribute of the <item/> element when the value of the 'type' attribute is "jid".
- Data Forms [XEP-0004]: the <value/> element when the 'type' attribute is "jid-single" or "jid-multi".

- Flexible Offline Message Retrieval [XEP-0013]: the 'jid' attribute of the <x/> element.
- Service Discovery [XEP-0030]: the 'jid' attribute of the <item/> element.
- Extended Stanza Addressing [XEP-0033]: the 'jid' attribute of the <address/> element.
- Multi-User Chat [XEP-0045]: the 'actor' child of the <item/> element; the 'jid' attribute of the <item/> element; the 'from' and 'to' attributes of the <invite/> and <decline/> elements; the 'jid' attribute of the <destroy/> element.
- Bookmarks [XEP-0048]: the 'jid' attribute of the <conference/> element.
- vCards [XEP-0054]: the <JABBERID/> of the <vCard/> element.
- Jabber Search [XEP-0055]: the 'jid' attribute of the <item/> element.
- Publish-Subscribe [XEP-0060]: the 'jid' attribute of the <affiliation/>, <options/>, <subscribe>, <subscription/>, and <unsubscribe/> elements; the 'publisher' attribute of the <item/> element.
- SOCKS5 Bytestreams [XEP-0065]: the 'jid' attribute of the <streamhost/> and <streamhost-used/> elements.
- Advanced Message Processing [XEP-0079]: the 'from' and 'to' attributes of the <amp/> element.
- Jabber Component Protocol [XEP-0114]: the 'from' and 'to' attributes of the <iq/>, <message/>, and <presence/> elements.
- Message Archiving [XEP-0136]: the 'with' attribute of the <chat/>, <from/>, and <item/> elements.
- Roster Item Exchange [XEP-0144]: the 'jid' attribute of the <item/> element.
- Jingle [XEP-0166]: the 'initiator' and 'responder' attributes of the <jingle/> element.
- Delayed Delivery [XEP-0203]: the 'from' attribute of the <delay/> element.
- Simple Communications Blocking [XEP-0191]: the 'jid' attribute of the <item/> element.
- Server Dialback [RFC3921], [XEP-0220]: the 'from' and 'to' attributes of the <result/> and <verify/> elements.
- Direct MUC Invitations [XEP-0249]: the 'jid' attribute of the <x/> element.

7.2. Localpart Slot

In XMPP systems, localparts can appear in at least the following slots:

- Multi-User Chat [XEP-0045]: the <unique/> element.

- In-Band Registration [XEP-0077]: the <username/> element.

7.3. Resourcepart Slot

In XMPP systems, resourceparts can appear in at least the following slots:

- Core [RFC6120]: the <resource/> child of the <bind/> element.
- Multi-User Chat [XEP-0045]: the 'nick' attribute of the <item/> element.
- Bookmarks [XEP-0048]: the 'nick' attribute of the <conference/> element.
- Jabber Search [XEP-0055]: the 'nick' attribute of the <item/> and <query/> elements.
- Publish-Subscribe [XEP-0060]: the 'node' attribute of the <address/> element (this might actually be a "stringything slot" but typically it is handled as a resourcepart).

7.4. Wordything Slot

In XMPP systems, generic "wordythings" can appear in at least the following slots:

- Multi-User Chat [XEP-0045]: the <password/> child of the <destroy/> and <x/> elements.
- Bookmarks [XEP-0048]: the 'password' attribute of the <conference/> element.
- Direct MUC Invitations [XEP-0249]: the 'password' attribute of the <x/> element.

7.5. Stringything Slot

In XMPP systems, generic "stringythings" can appear in at least the following slots:

- Flexible Offline Message Retrieval [XEP-0013]: the 'node' attribute of the <x/> element.
- Extended Stanza Addressing [XEP-0033]: the 'node' attribute of the <address/> element.
- Publish-Subscribe [XEP-0060]: the 'node' attribute of various XML elements.

8. XMPP Error Handling

Both the core XMPP specifications and various XMPP extensions might need to define more robust error handling. Although this topic has yet to be explored in detail, it is likely that specifications can

more widely use the existing <jid-malformed/> error condition defined in [RFC6120].

9. XMPP User Interface Issues

[RFC5895] introduces the helpful concept of "the dividing line between user interface and protocol" and applies that concept to the complexs process of translating the user's (presumed) intentions into bits on the wire. IDNA2003 conflated user interface processing and machine-readable protocols, and in many ways XMPP inherited that same error. It would be desirable for XMPP technologies to define a clear dividing line between user interface and protocol. This might mean that the XMPP community will need to define recommended mappings that are applied to a string before it is considered a JID (or the localpart of resourcepart of a JID).

10. Security Considerations

The inclusion of non-ASCII characters in XMPP addresses has important security implications, such as the ability to mimic characters or entire addresses through the inclusion of "confusable characters" (see [RFC4690] and [RFC5890]). These issues are explored at some length in [RFC6122]. Other security considerations might apply and will be described in a future version of this specification.

11. IANA Considerations

This document defines no actions for the IANA.

12. Acknowledgements

Special thanks to Joe Hildebrand for extensive discussions about internationalization and XMPP. Many participants in the XMPP WG Interim Meeting in February 2011 provided valuable feedback. Thanks also to Jack Erwin, Matt Miller, and Tory Patnoe for additional discussions.

13. Informative References

[FRAMEWORK]
Blanchet, M., "Precis Framework: Handling Internationalized Strings in Protocols", draft-blanchet-precis-framework-00 (work in progress),

July 2010.

[PROBLEM] Blanchet, M. and A. Sullivan, "Stringprep Revision Problem Statement", draft-ietf-precis-problem-statement-01 (work in progress), December 2010.

[RFC3454] Hoffman, P. and M. Blanchet, "Preparation of Internationalized Strings ("stringprep")", RFC 3454, December 2002.

[RFC3490] Faltstrom, P., Hoffman, P., and A. Costello, "Internationalizing Domain Names in Applications (IDNA)", RFC 3490, March 2003.

[RFC3491] Hoffman, P. and M. Blanchet, "Nameprep: A Stringprep Profile for Internationalized Domain Names (IDN)", RFC 3491, March 2003.

[RFC3920] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Core", RFC 3920, October 2004.

[RFC3921] Saint-Andre, P., Ed., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence", RFC 3921, October 2004.

[RFC4422] Melnikov, A. and K. Zeilenga, "Simple Authentication and Security Layer (SASL)", RFC 4422, June 2006.

[RFC4690] Klensin, J., Faltstrom, P., Karp, C., and IAB, "Review and Recommendations for Internationalized Domain Names (IDNs)", RFC 4690, September 2006.

[RFC5122] Saint-Andre, P., "Internationalized Resource Identifiers (IRIs) and Uniform Resource Identifiers (URIs) for the Extensible Messaging and Presence Protocol (XMPP)", RFC 5122, February 2008.

[RFC5322] Resnick, P., Ed., "Internet Message Format", RFC 5322, October 2008.

[RFC5890] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Definitions and Document Framework", RFC 5890, August 2010.

[RFC5891] Klensin, J., "Internationalized Domain Names in Applications (IDNA): Protocol", RFC 5891, August 2010.

[RFC5892] Faltstrom, P., "The Unicode Code Points and

Internationalized Domain Names for Applications (IDNA)", RFC 5892, August 2010.

[RFC5893] Alvestrand, H. and C. Karp, "Right-to-Left Scripts for Internationalized Domain Names for Applications (IDNA)", RFC 5893, August 2010.

[RFC5894] Klensin, J., "Internationalized Domain Names for Applications (IDNA): Background, Explanation, and Rationale", RFC 5894, August 2010.

[RFC5895] Resnick, P. and P. Hoffman, "Mapping Characters for Internationalized Domain Names in Applications (IDNA) 2008", RFC 5895, September 2010.

[RFC6120] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Core", draft-ietf-xmpp-3920bis-22 (work in progress), December 2010.

[RFC6121] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Instant Messaging and Presence", draft-ietf-xmpp-3921bis-20 (work in progress), January 2011.

[RFC6122] Saint-Andre, P., "Extensible Messaging and Presence Protocol (XMPP): Address Format", draft-ietf-xmpp-address-09 (work in progress), January 2011.

[UAX15] The Unicode Consortium, "Unicode Standard Annex #15: Unicode Normalization Forms", September 2010.

[XEP-0004] Eatmon, R., Hildebrand, J., Miller, J., Muldowney, T., and P. Saint-Andre, "Data Forms", XSF XEP 0004, August 2007.

[XEP-0013] Saint-Andre, P. and C. Kaes, "Flexible Offline Message Retrieval", XSF XEP 0013, July 2005.

[XEP-0016] Millard, P. and P. Saint-Andre, "Privacy Lists", XSF XEP 0016, February 2007.

[XEP-0029] Kaes, C., "Definition of Jabber Identifiers (JIDs)", XSF XEP 0029, October 2003.

[XEP-0030]
Hildebrand, J., Millard, P., Eatmon, R., and P. Saint-Andre, "Service Discovery", XSF XEP 0030, June 2008.

[XEP-0033]
Hildebrand, J. and P. Saint-Andre, "Extended Stanza Addressing", XSF XEP 0033, September 2004.

[XEP-0045]
Saint-Andre, P., "Multi-User Chat", XSF XEP 0045, July 2008.

[XEP-0048]
Blackman, R., Millard, P., and P. Saint-Andre, "Bookmarks", XSF XEP 0048, November 2007.

[XEP-0054]
Saint-Andre, P., "vcard-temp", XSF XEP 0054, July 2008.

[XEP-0055]
Saint-Andre, P., "Jabber Search", XSF XEP 0055, September 2009.

[XEP-0060]
Millard, P., Saint-Andre, P., and R. Meijer, "Publish-Subscribe", XSF XEP 0060, July 2010.

[XEP-0065]
Smith, D., Miller, M., Saint-Andre, P., and J. Karneges, "SOCKS5 Bytestreams", XSF XEP 0065, April in progress, last updated 2010.

[XEP-0077]
Saint-Andre, P., "In-Band Registration", XSF XEP 0077, September 2009.

[XEP-0079]
Miller, M. and P. Saint-Andre, "Advanced Message Processing", XSF XEP 0079, November 2005.

[XEP-0114]
Saint-Andre, P., "Jabber Component Protocol", XSF XEP 0114, March 2005.

[XEP-0136]
Paterson, I., Perlow, J., Saint-Andre, P., Karneges, J., Tsvyashchenko, A., and Y. Leboulanger, "Message Archiving", XSF XEP 0136, June 2010.

[XEP-0144]
Saint-Andre, P., "Roster Item Exchange", XSF XEP 0144, August 2005.

[XEP-0166]
Ludwig, S., Beda, J., Saint-Andre, P., McQueen, R., Egan, S., and J. Hildebrand, "Jingle", XSF XEP 0166, December 2009.

[XEP-0191]
Saint-Andre, P., "Simple Communications Blocking", XSF XEP 0191, February 2007.

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

[XEP-0220]
Miller, J., Saint-Andre, P., and P. Hancke, "Server Dialback", XSF XEP 0220, March 2010.

[XEP-0249]
Saint-Andre, P., "Direct MUC Invitations", XSF XEP 0249, December 2009.

Author's Address

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

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