XEP-0238: XMPP Protocol Flows for Inter-Domain Federation

Peter Saint-Andre
mailto:stpeter@stpeter.im
xmpp:stpeter@jabber.org
https://stpeter.im/

2008-03-31
Version 0.1

Status	Type	Short Name
Deferred	Informational	N/A

This specification provides detailed protocol flows for the establishment of communication between domains that provide XMPP services, including permutations for a wide variety of possible federation policies.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction **1**

2 Terminology **2**

3 Connection Success **3**

4 Connections from Type 1 Services **4**
4.1 Type 1 to Type 1 4
4.2 Type 1 to Type 2 6
4.3 Type 1 to Type 3 9
4.4 Type 1 to Type 4 11
4.5 Type 1 to Type 5 13
4.6 Type 1 to Type 6 15

5 Connections from Type 2 Services **17**
5.1 Type 2 to Type 1 17
5.2 Type 2 to Type 2 20
5.3 Type 2 to Type 3 22
5.4 Type 2 to Type 4 25
5.5 Type 2 to Type 5 29
5.6 Type 2 to Type 6 32

6 Connections from Type 3 Services **34**
6.1 Type 3 to Type 1 34
6.2 Type 3 to Type 2 37
6.3 Type 3 to Type 3 39
6.4 Type 3 to Type 4 42
6.5 Type 3 to Type 5 46
6.6 Type 3 to Type 6 49

7 Connections from Type 4 Services **52**
7.1 Type 4 to Type 1 52
7.2 Type 4 to Type 2 54
7.3 Type 4 to Type 3 57
7.4 Type 4 to Type 4 61
7.5 Type 4 to Type 5 64
7.6 Type 4 to Type 6 68

8 Connections from Type 5 Services **70**
8.1 Type 5 to Type 1 70
8.2 Type 5 to Type 2 71
8.3 Type 5 to Type 3 75
8.4 Type 5 to Type 4 78

8.5 Type 5 to Type 5 81
8.6 Type 5 to Type 6 84

9 Connections from Type 6 Services 88
9.1 Type 6 to Type 1 88
9.2 Type 6 to Type 2 89
9.3 Type 6 to Type 3 91
9.4 Type 6 to Type 4 94
9.5 Type 6 to Type 5 95
9.6 Type 6 to Type 6 99

10 Security Considerations 102

11 IANA Considerations 102

12 XMPP Registrar Considerations 102

13 Acknowledgements 102

1 Introduction

XMPP Core [1] describes the client-server architecture upon which Jabber/XMPP communication is based. One aspect of such communication is "federation", i.e., the ability for two XMPP servers in different domains to exchange XML stanzas. There are at least four levels of federation:

1. Permissive Federation -- a server accepts a connection from any other peer on the network, even without verifiying the identity of the peer based on DNS lookups. The lack of peer verification or authentication means that domains can be spoofed. Permissive federation was effectively outlawed on the Jabber network in October 2000 with the release of the jabberd 1.2 server, which included support for the newly-developed Server Dialback (XEP-0220) [2] protocol.

2. Verified Federation -- a server accepts a connection from a peer only after the identity of the peer has been weakly verified via Server Dialback, based on information obtained via the Domain Name System (DNS) and verification keys exchanged in-band over XMPP. However, the connection is not encrypted. The use of identity verification effectively prevents domain spoofing, but federation requires proper DNS setup and is still subject to DNS poisoning attacks. Verified federation has been the default service policy followed by servers on the open XMPP network from October 2000 until now.

3. Encrypted Federation -- a server accepts a connection from a peer only if the peer supports Transport Layer Security (TLS) as defined for XMPP in XMPP Core [3] and the peer presents a digital certificate. However, the certificate may be self-signed, in which case mutual authentication is typically not possible. Therefore, after STARTTLS negotiation the parties proceed to weakly verify identity using Server Dialback. This combination results in an encrypted connection with weak identity verification.

4. Trusted Federation -- a server accepts a connection from a peer only if the peer supports Transport Layer Security (TLS) and the peer presents a digital certificate issued by a trusted root certification authority (CA). The list of trusted root CAs is determined by local service policy, as is the level of trust accorded to various types of certificates (i.e., Class 1, Class 2, or Class 3). The use of trusted domain certificates effectively prevents DNS poisoning attacks but makes federation more difficult since typically such certificates are not easy to obtain.

[1] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[2] XEP-0220: Server Dialback <https://xmpp.org/extensions/xep-0220.html>.
[3] RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

The remainder of this document describes in more detail the protocol flows that make it possible to deploy verified federation, encrypted federation, and trusted federation. Protocol flows are shown for federation attempts between various combinations to illustrate the interaction between different federation policies.

2 Terminology

To simplify the text, this document uses the following terminology. For each service type, the domain ”example.lit” is used to illustrate connections to that same service type.

Service Type	Federation Policy	Certificate	Protocols Supported	Example Domain	Example User
Type 1	Verified Only	None	XMPP 0.9 ”XMPP 0.9” is the core XML streaming protocol used in the Jabber community before the formalization of XMPP 1.0 by the IETF in RFC 3920 RFC 3920: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc3920>., including STARTTLS and SASL. and Server Dialback	type1.lit	citizen@type1.lit

Service Type	Federation Policy	Certificate	Protocols Supported	Example Domain	Example User
Type 2	Verified Acceptable	Self-signed	XMPP 1.0 ”XMPP 1.0” is defined in RFC 3920 and includes STARTTLS and SASL negotiation. and Server Dialback	type2.lit	juliet@type2.lit
Type 3	Verified Acceptable	CA-issued	XMPP 1.0 and Server Dialback	type3.lit	romeo@type3.lit
Type 4	Encrypted Required	Self-signed	XMPP 1.0 and Server Dialback	type4.lit	hamlet@type4.lit
Type 5	Encrypted Required	CA-issued	XMPP 1.0 and Server Dialback	type5.lit	bill@type5.lit
Type 6	Trusted Required	CA-issued	XMPP 1.0	type6.lit	chris@type6.lit

3 Connection Success

The following table summarizes the results of connection attempts between the various services, where ”U” stands for ”Unsuccessful”, ”V” stands for ”Verified”, ”E” stands for ”Encrypted”, and ”T” stands for ”Trusted”. The rows indicate the initiating service and the columns indicate the receiving service.

	Type 1	Type 2	Type 3	Type 4	Type 5	Type 6
Type 1	V	V	V	U	U	U
Type 2	V	V	E	E	U	U
Type 3	V	V	E	E	E	T
Type 4	U	E	E	E	E	U
Type 5	U	E	T	E	T	T
Type 6	U	U	T	U	T	T

4 Connections from Type 1 Services

4.1 Type 1 to Type 1

In this scenario, an XMPP user citizen@type1.lit attempts to send an XML stanza to user@example.lit.

Listing 1: Test Stanza

```
<iq from='citizen@type1.lit/foo'
    id='t1_t1'
    to='user@example.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type1.lit service attempts to initiate a server-to-server connection with example.lit (both of which support verified connections only and neither of which has a certificate). First, the type1.lit service sends an initial stream header to example.lit.

Listing 2: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        to='example.lit'>
```

Next the example.lit service sends a response stream header to type1.lit.

Listing 3: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt1_t1o'
        to='type1.lit'>
```

Because neither service supports XMPP 1.0, the type1.lit service attempts to complete a server dialback negotiation with the example.lit service. Therefore it sends a dialback key to example.lit over the existing connection.

Listing 4: Dialback Key

```
<db:result
```

```
    from='type1.lit'
    to='example.lit'>
  some-long-dialback-key
</db:result>
```

The example.lit service then performs a DNS lookup on the type1.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type1.lit service.

Listing 5: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        to='type1.lit'>
```

The authoritative server for the type1.lit service then returns a response stream header.

Listing 6: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        id='idt1_t1r'
        to='example.lit'>
```

The example.lit service then sends a dialback verification request to the authoritative server for the type1.lit domain.

Listing 7: Verification Request

```
<db:verify
    from='example.lit'
    id='idt1_t1o'
    to='type1.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type1.lit domain notifies the example.lit service that the key is valid.

Listing 8: Key is Valid

```
<db:verify
```

```
    from='type1.lit'
    id='idt1_t1o'
    to='example.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The example.lit service then returns a positive server dialback result to the originating server.

Listing 9: Server Dialback Result

```
<db:result
    from='example.lit'
    to='type1.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type1.lit service routes the XML stanza from citizen@type1.lit to the example.lit service.

4.2 Type 1 to Type 2

In this scenario, an XMPP user citizen@type1.lit attempts to send an XML stanza to juliet@type2.lit:

Listing 10: Test Stanza

```
<iq from='citizen@type1.lit/foo'
    id='t1_t2'
    to='juliet@type2.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type1.lit service (which supports verified connections only and does not have a certificate) attempts to initiate a server-to-server connection with the type2.lit service (which accepts verified connections and has a self-signed certificate).
First, the type1.lit service sends an initial stream header to type2.lit.

Listing 11: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
```

```
        to='type2.lit'>
```

Next the type2.lit service sends a response stream header to type1.lit.

Listing 12: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt1_t2o'
        to='type1.lit'
        version='1.0'>
```

The type2.lit service also sends stream features.

Listing 13: Stream Features

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

Because the type1.lit service does not support XMPP 1.0, it ignores the stream features and attempts to complete a server dialback negotiation with the type2.lit service. Therefore it sends a dialback key to type2.lit over the existing connection.

Listing 14: Dialback Key

```
<db:result
    from='type1.lit'
    to='type2.lit'>
  some-long-dialback-key
</db:result>
```

The type2.lit service then performs a DNS lookup on the type1.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type1.lit service.

Listing 15: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type1.lit'
        version='1.0'>
```

The authoritative server for the type1.lit service then returns a response stream header.

Listing 16: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        id='idt1_t2r'
        to='type2.lit'>
```

The type2.lit service then sends a dialback verification request to the authoritative server for the type1.lit domain.

Listing 17: Verification Request

```
<db:verify
    from='type2.lit'
    id='idt1_t2o'
    to='type1.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type1.lit domain notifies the type2.lit service that the key is valid.

Listing 18: Key is Valid

```
<db:verify
    from='type1.lit'
    id='idt1_t2o'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type2.lit service then returns a positive server dialback result to the originating server.

Listing 19: Server Dialback Result

```
<db:result
    from='type2.lit'
    to='type1.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type1.lit service routes the XML stanza from citizen@type1.lit to the type2.lit service.

4.3 Type 1 to Type 3

In this scenario, an XMPP user citizen@type1.lit attempts to send an XML stanza to romeo@type3.lit.

Listing 20: Test Stanza

```
<iq from='citizen@type1.lit/foo'
    id='t1_t3'
    to='romeo@type3.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type1.lit service (which supports verified connections only and does not have a certificate) attempts to initiate a server-to-server connection with the type3.lit service (which accepts verified connections and has a CA-issued certificate).
First, the type1.lit service sends an initial stream header to type3.lit.

Listing 21: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        to='type3.lit'>
```

Next the type3.lit service sends a response stream header to type1.lit.

Listing 22: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt1_t3o'
        to='type1.lit'
        version='1.0'>
```

The type3.lit service also sends stream features.

Listing 23: Stream Features

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

Because the type1.lit service does not support XMPP 1.0, it ignores the stream features and attempts to complete a server dialback negotiation with the type3.lit service. Therefore it sends a dialback key to the authoritative server for the type3.lit service.

Listing 24: Dialback Key

```
<db:result
    from='type1.lit'
    to='type3.lit'>
  some-long-dialback-key
</db:result>
```

The type3.lit service then performs a DNS lookup on the type1.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server.

Listing 25: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type1.lit'
        version='1.0'>
```

The authoritative server for the type1.lit service then returns a response stream header.

Listing 26: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        id='idt1_t3r'
        to='type3.lit'>
```

The type3.lit service then sends a dialback verification request to the authoritative server for the type1.lit domain.

Listing 27: Verification Request

```
<db:verify
    from='type3.lit'
    id='idt1_t3o'
    to='type1.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type1.lit domain notifies the type3.lit service that the key is valid.

Listing 28: Key is Valid

```
<db:verify
    from='type1.lit'
    id='idt1_t3o'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type3.lit service then returns a positive server dialback result to the originating server.

Listing 29: Server Dialback Result

```
<db:result
    from='type3.lit'
    to='type1.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type1.lit service routes the XML stanza from citizen@type1.lit to the type3.lit service.

4.4 Type 1 to Type 4

In this scenario, an XMPP user citizen@type1.lit attempts to send an XML stanza to hamlet@type4.lit.

Listing 30: Test Stanza

```
<iq from='citizen@type1.lit/foo'
    id='t1_t4'
    to='hamlet@type4.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
```

```
</iq>
```

Therefore the type1.lit service (which supports verified connections only and does not have a certificate) attempts to initiate a server-to-server connection with type4.lit (which does not accept verified connections and has a self-signed certificate).
First, the type1.lit service sends an initial stream header to type4.lit.

Listing 31: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        to='type4.lit'>
```

Next the type4.lit service sends a response stream header to type1.lit.

Listing 32: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt1_t4o'
        to='type1.lit'
        version='1.0'>
```

The type4.lit service also sends stream features. Because the type4.lit service does not accept verified connections, it returns stream features with a notation that STARTTLS is required.

Listing 33: Stream Features

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

Because the type1.lit service does not support XMPP 1.0, it ignores the stream features and attempts to complete a server dialback negotiation with the type4.lit service. Therefore it sends a dialback key to the authoritative server for the type4.lit service.

Listing 34: Dialback Key

```
<db:result
    from='type1.lit'
```

```
    to='type4.lit'>
  some-long-dialback-key
</db:result>
```

The type4.lit service understands the server dialback protocol but since it requires STARTTLS at this point in the stream negotiation it returns a stream error to the type1.lit service, which should be <not-authorized/>.

Listing 35: Stream Error

```
<stream:error>
  <not-authorized
      xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
</stream:error>
</stream:stream>
```

Because the connection is unsuccessful, the type1.lit service returns a stanza error to citizen@type1.lit, which should be <remote-server-timeout/>.

Listing 36: Error Stanza

```
<iq from='romeo@type4.lit'
    id='t1_t4'
    to='citizen@type1.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

4.5 Type 1 to Type 5

In this scenario, an XMPP user citizen@type1.lit attempts to send an XML stanza to bill@type5.lit.

Listing 37: Test Stanza

```
<iq from='citizen@type1.lit/foo'
    id='t1_t5'
    to='bill@type5.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type1.lit service (which supports verified connections only and does not have a certificate) attempts to initiate a server-to-server connection with type5.lit (which does not

accept verified connections and has a CA-issued signed certificate).
First, the type1.lit service sends an initial stream header to type5.lit.

Listing 38: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        to='type5.lit'>
```

Next the type5.lit service sends a response stream header to type1.lit.

Listing 39: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt1_t5o'
        to='type1.lit'
        version='1.0'>
```

The type5.lit service also sends stream features. Because the type5.lit service does not accept verified connections, it returns stream features with a notation that STARTTLS is required.

Listing 40: Stream Features

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

Because the type1.lit service does not support XMPP 1.0, it ignores the stream features and attempts to complete a server dialback negotiation with the type5.lit service. Therefore it sends a dialback key to the authoritative server for the type5.lit service.

Listing 41: Dialback Key

```
<db:result
    from='type1.lit'
    to='type5.lit'>
  some-long-dialback-key
</db:result>
```

The type5.lit service understands the server dialback protocol but since it requires STARTTLS at this point in the stream negotiation it returns a stream error to the type1.lit service, which should be <not-authorized/>.

Listing 42: Stream Error

```
<stream:error>
  <not-authorized
      xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
</stream:error>
</stream:stream>
```

Because the connection is unsuccessful, the type1.lit service returns a stanza error to citizen@type1.lit, which should be <remote-server-timeout/>.

Listing 43: Error Stanza

```
<iq from='bill@type5.lit'
    id='t1_t5'
    to='citizen@type1.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

4.6 Type 1 to Type 6

In this scenario, an XMPP user citizen@type1.lit attempts to send an XML stanza to chris@type6.lit.

Listing 44: Test Stanza

```
<iq from='citizen@type1.lit/foo'
    id='t1_t6'
    to='chris@type6.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type1.lit service (which supports verified connections only and does not have a certificate) attempts to initiate a server-to-server connection with the type6.lit service (which accepts only trusted connections, has a CA-issued certificate, and does not support Server Dialback).
First, the type1.lit service sends an initial stream header to type6.lit.

Listing 45: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        to='type6.lit'>
```

Next the type6.lit service sends a response stream header to type1.lit. Notice that the response stream header does not include the dialback namespace, since the type6.lit service does not support Server Dialback.

Listing 46: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt1_t6o'
        to='type1.lit'
        version='1.0'>
```

The type6.lit service also sends stream features. Because the type6.lit service does not accept untrusted connections, it returns stream features with a notation that STARTTLS is required.

Listing 47: Stream Features

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

The type1.lit service does not detect support for server dialback by the type6.lit service but in any case attempts to complete server dialback.

Listing 48: Dialback Key

```
<db:result
    from='type1.lit'
    to='type6.lit'>
  some-long-dialback-key
</db:result>
```

The type6.lit service does not accept dialback negotiations so it returns a <not-authorized/> stream error and closes the stream.

Listing 49: Stream Error

```
<stream:error>
  <not-authorized
      xmlns='urn:ietf:params:xml:ns:xmpp-streams'/>
</stream:error>
</stream:stream>
```

The type1.lit service closes the stream as well.

Listing 50: Stream Close

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

Because the connection is unsuccessful, the type1.lit service returns a stanza error to citizen@type1.lit, which should be <remote-server-timeout/>.

Listing 51: Error Stanza

```
<iq from='chris@type6.lit'
    id='t1_t6'
    to='citizen@type1.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

5 Connections from Type 2 Services

5.1 Type 2 to Type 1

In this scenario, an XMPP user juliet@type2.lit attempts to send an XML stanza to citizen@type1.lit:

Listing 52: Test Stanza

```
<iq from='juliet@type2.lit/foo'
    id='t2_t1'
    to='citizen@type1.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type2.lit service (which accepts verified connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type1.lit service (which

supports verified connections only and does not have a certificate).
First, the type2.lit service sends an initial stream header to type1.lit.

Listing 53: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type1.lit'
        version='1.0'>
```

Next the type1.lit service sends a response stream header to type2.lit.

Listing 54: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        id='idt1_t2o'
        to='type2.lit'>
```

Because the type1.lit service does not support XMPP 1.0, it does not send stream features. Because the type2.lit service accepts verified connections, it attempts to verify the identity of type1.lit using server dialback. Therefore it sends a dialback key to type1.lit over the existing connection.

Listing 55: Dialback Key

```
<db:result
    from='type2.lit'
    to='type1.lit'>
  some-long-dialback-key
</db:result>
```

The type1.lit service then performs a DNS lookup on the type2.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type2.lit service.

Listing 56: Initial Stream Header

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

```
        from='type1.lit'
        to='type2.lit'>
```

The authoritative server for the type2.lit service then returns a response stream header.

Listing 57: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt1_t2r'
        to='type1.lit'
        version='1.0'>
```

The type1.lit service then sends a dialback verification request to the authoritative server for the type2.lit domain.

Listing 58: Verification Request

```
<db:verify
    from='type1.lit'
    id='idt2_t1o'
    to='type2.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type2.lit domain notifies the type1.lit service that the key is valid.

Listing 59: Key is Valid

```
<db:verify
    from='type2.lit'
    id='idt1_t1o'
    to='type1.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type1.lit service then returns a positive server dialback result to the originating server (i.e., type2.lit).

Listing 60: Server Dialback Result

```
<db:result
    from='type1.lit'
```

```
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type2.lit service routes the XML stanza from juliet@type2.lit to the type1.lit service.

5.2 Type 2 to Type 2

In this scenario, an XMPP user juliet@type2.lit attempts to send an XML stanza to user@example.lit:

Listing 61: Test Stanza

```
<iq from='juliet@type2.lit/foo'
    id='t2_t2'
    to='user@example.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type2.lit service (which accepts verified connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the example.lit service (which also supports verified connections and has a self-signed certificate).
First, the type2.lit service sends an initial stream header to example.lit.

Listing 62: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type2.lit.

Listing 63: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt2_t2o'
```

```
        to='type2.lit'>
```

Because the example.lit service supports XMPP 1.0, it also sends stream features.

Listing 64: Stream Features

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

We assume that type2.lit does not attempt STARTTLS negotiation but instead attempts server dialback for weak identity verification.

Listing 65: Dialback Key

```
<db:result
    from='type2.lit'
    to='example.lit'>
  some-long-dialback-key
</db:result>
```

The example.lit service then performs a DNS lookup on the type2.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type2.lit service.

Listing 66: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        to='type2.lit'>
```

The authoritative server for the type2.lit service then returns a response stream header.

Listing 67: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt2_t2r'
        to='example.lit'
        version='1.0'>
```

The example.lit service then sends a dialback verification request to the authoritative server for the type2.lit domain.

Listing 68: Verification Request

```
<db:verify
    from='example.lit'
    id='idt2_t2o'
    to='type2.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type2.lit domain notifies the example.lit service that the key is valid.

Listing 69: Key is Valid

```
<db:verify
    from='type2.lit'
    id='idt2_t2o'
    to='example.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The example.lit service then returns a positive server dialback result to the originating server (i.e., type2.lit).

Listing 70: Server Dialback Result

```
<db:result
    from='example.lit'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type2.lit service routes the XML stanza from juliet@type2.lit to the example.lit service.

5.3 Type 2 to Type 3

In this scenario, an XMPP user juliet@type2.lit attempts to send an XML stanza to romeo@type3.lit:

Listing 71: Test Stanza

```
<iq from='juliet@type2.lit/foo'
    id='t2_t3'
    to='romeo@type3.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type2.lit service (which accepts verified connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type3.lit service (which also supports verified connections and has a CA-issued certificate).
First, the type2.lit service sends an initial stream header to type3.lit.

Listing 72: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type2.lit.

Listing 73: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt2_t3o'
        to='type2.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features.

Listing 74: Stream Features

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

We assume that type2.lit does not attempt STARTTLS negotiation but instead attempts server dialback for weak identity verification.

Listing 75: Dialback Key

```
<db:result
    from='type2.lit'
    to='type3.lit'>
  some-long-dialback-key
</db:result>
```

The type3.lit service then performs a DNS lookup on the type2.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type2.lit service.

Listing 76: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type2.lit'>
```

The authoritative server for the type2.lit service then returns a response stream header.

Listing 77: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt2_t3r'
        to='type3.lit'
        version='1.0'>
```

The type3.lit service then sends a dialback verification request to the authoritative server for the type2.lit domain.

Listing 78: Verification Request

```
<db:verify
    from='type3.lit'
    id='idt2_t3o'
    to='type2.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type2.lit domain notifies the type3.lit service that the key is valid.

Listing 79: Key is Valid

```
<db:verify
    from='type2.lit'
    id='idt2_t3o'
    to='type3.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type3.lit service then returns a positive server dialback result to the originating server (i.e., type2.lit).

Listing 80: Server Dialback Result

```
<db:result
    from='type3.lit'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type2.lit service routes the XML stanza from juliet@type2.lit to the type3.lit service.

5.4 Type 2 to Type 4

In this scenario, an XMPP user juliet@type2.lit attempts to send an XML stanza to hamlet@type4.lit:

Listing 81: Test Stanza

```
<iq from='juliet@type2.lit/foo'
    id='t2_t4'
    to='hamlet@type4.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type2.lit service (which accepts verified connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type4.lit service (which also supports verified connections and has a CA-issued certificate).
First, the type2.lit service sends an initial stream header to type4.lit.

Listing 82: Initial Stream Header

```
<stream:stream
```

```
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type4.lit'
        version='1.0'>
```

Next the type4.lit service sends a response stream header to type2.lit.

Listing 83: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt2_t4o'
        to='type2.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 84: Stream Features

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

Because type4.lit requires encryption, type2.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 85: STARTTLS Request

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

Listing 86: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type2.lit service then opens a new stream over the encrypted connection.

Listing 87: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
```

```
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type4.lit'
        version='1.0'>
```

Next the type4.lit service sends a response stream header to type2.lit.

Listing 88: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt2_t4o2'
        to='type2.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 89: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
  </dialback>
</stream:features>
```

Notice that type4.lit requires dialback here (perhaps because of some local service policy). Therefore type2.lit sends a dialback key to type4.lit.

Listing 90: Dialback Key

```
<db:result
    from='type2.lit'
    to='type4.lit'>
  some-long-dialback-key
</db:result>
```

The type4.lit service then performs a DNS lookup on the type2.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type2.lit service.

Listing 91: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
```

```
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type2.lit'>
```

The authoritative server for the type2.lit service then returns a response stream header.

Listing 92: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt2_t4r'
        to='type4.lit'
        version='1.0'>
```

The type4.lit service then sends a dialback verification request to the authoritative server for the type2.lit domain.

Listing 93: Verification Request

```
<db:verify
    from='type4.lit'
    id='idt2_t4o'
    to='type2.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type2.lit domain notifies the type4.lit service that the key is valid.

Listing 94: Key is Valid

```
<db:verify
    from='type2.lit'
    id='idt2_t4o'
    to='type4.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type4.lit service then returns a positive server dialback result to the originating server (i.e., type2.lit).

Listing 95: Server Dialback Result

```
<db:result
```

```
    from='type4.lit'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type2.lit service routes the XML stanza from juliet@type2.lit to the type4.lit service.

5.5 Type 2 to Type 5

In this scenario, an XMPP user juliet@type2.lit attempts to send an XML stanza to bill@type5.lit:

Listing 96: Test Stanza

```
<iq from='juliet@type2.lit/foo'
    id='t2_t5'
    to='bill@type5.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type2.lit service (which accepts verified connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type5.lit service (which also supports verified connections and has a CA-issued certificate).
First, the type2.lit service sends an initial stream header to type5.lit.

Listing 97: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type2.lit.

Listing 98: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
```

```
        id='idt2_t5o'
        to='type2.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 99: Stream Features

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

Because type5.lit requires encryption, type2.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 100: STARTTLS Request

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

Listing 101: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type2.lit service then opens a new stream over the encrypted connection.

Listing 102: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type2.lit.

Listing 103: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt2_t5o2'
        to='type2.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 104: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
  </dialback>
</stream:features>
```

Notice that type5.lit requires dialback here (perhaps because of some local service policy). Therefore type2.lit sends a dialback key to type5.lit.

Listing 105: Dialback Key

```
<db:result
    from='type2.lit'
    to='type5.lit'>
  some-long-dialback-key
</db:result>
```

The type5.lit service then performs a DNS lookup on the type2.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type2.lit service.

Listing 106: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type2.lit'>
```

The authoritative server for the type2.lit service then returns a response stream header.

Listing 107: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt2_t5r'
        to='type5.lit'
        version='1.0'>
```

The type5.lit service then sends a dialback verification request to the authoritative server for the type2.lit domain.

Listing 108: Verification Request

```
<db:verify
    from='type5.lit'
    id='idt2_t5o'
    to='type2.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type2.lit domain notifies the type5.lit service that the key is valid.

Listing 109: Key is Valid

```
<db:verify
    from='type2.lit'
    id='idt2_t5o'
    to='type5.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type5.lit service then returns a positive server dialback result to the originating server (i.e., type2.lit).

Listing 110: Server Dialback Result

```
<db:result
    from='type5.lit'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type2.lit service routes the XML stanza from juliet@type2.lit to the type5.lit service.

5.6 Type 2 to Type 6

In this scenario, an XMPP user juliet@type2.lit attempts to send an XML stanza to chris@type6.lit.

Listing 111: Test Stanza

```
<iq from='juliet@type2.lit/foo'
    id='t2_t6'
    to='chris@type6.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type2.lit service (which supports verified connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type6.lit service (which accepts only trusted connections, has a CA-issued certificate, and does not support Server Dialback).
First, the type2.lit service sends an initial stream header to type6.lit.

Listing 112: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type6.lit'>
```

Next the type6.lit service sends a response stream header to type2.lit. Notice that the response stream header does not include the dialback namespace, since the type6.lit service does not support Server Dialback.

Listing 113: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt2_t6o'
        to='type2.lit'
        version='1.0'>
```

The type6.lit service also sends stream features. Because the type6.lit service does not accept untrusted connections, it returns stream features with a notation that STARTTLS is required.

Listing 114: Stream Features

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

Because type6.lit requires encryption, type2.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 115: STARTTLS Request

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

Listing 116: STARTTLS Response

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

The servers then attempt negotiate TLS. We assume the negotiation fails because type2.lit presents a self-signed certificate but type6.lit requires trusted federation relying on a common root CA.
Because the connection is unsuccessful, the type2.lit service returns a stanza error to juliet@type2.lit, which should be <remote-server-timeout/>.

Listing 117: Error Stanza

```
<iq from='chris@type6.lit'
    id='t2_t6'
    to='juliet@type2.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

6 Connections from Type 3 Services

6.1 Type 3 to Type 1

In this scenario, an XMPP user romeo@type3.lit attempts to send an XML stanza to citizen@type1.lit:

Listing 118: Test Stanza

```
<iq from='romeo@type3.lit/foo'
    id='t3_t1'
    to='citizen@type1.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type3.lit service (which accepts verified connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type1.lit service (which supports verified connections only and does not have a certificate).
First, the type3.lit service sends an initial stream header to type1.lit.

Listing 119: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type1.lit'
        version='1.0'>
```

Next the type1.lit service sends a response stream header to type3.lit.

Listing 120: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        id='idt3_t1o'
        to='type3.lit'>
```

Because the type1.lit service does not support XMPP 1.0, it does not send stream features. Therefore the type3.lit attempts to complete server dialback verification.

Listing 121: Dialback Key

```
<db:result
    from='type3.lit'
    to='type1.lit'>
  some-long-dialback-key
</db:result>
```

The type1.lit service then performs a DNS lookup on the type3.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type3.lit service.

Listing 122: Initial Stream Header

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

```
        from='type1.lit'
        to='type3.lit'>
```

The authoritative server for the type3.lit service then returns a response stream header.

Listing 123: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt3_t1r'
        to='type1.lit'
        version='1.0'>
```

The type1.lit service then sends a dialback verification request to the authoritative server for the type3.lit domain.

Listing 124: Verification Request

```
<db:verify
    from='type1.lit'
    id='idt3_t1o'
    to='type3.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type3.lit domain notifies the type1.lit service that the key is valid.

Listing 125: Key is Valid

```
<db:verify
    from='type3.lit'
    id='idt3_t1o'
    to='type1.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type1.lit service then returns a positive server dialback result to the originating server (i.e., type3.lit).

Listing 126: Server Dialback Result

```
<db:result
    from='type1.lit'
```

```
    to='type3.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type3.lit service routes the XML stanza from romeo@type3.lit to the type1.lit service.

6.2 Type 3 to Type 2

In this scenario, an XMPP user romeo@type3.lit attempts to send an XML stanza to juliet@type2.lit:

Listing 127: Test Stanza

```
<iq from='romeo@type3.lit/foo'
    id='t3_t2'
    to='juliet@type2.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type3.lit service (which accepts verified connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type2.lit service (which supports verified connections and has a self-signed certificate).
First, the type3.lit service sends an initial stream header to type2.lit.

Listing 128: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type2.lit'
        version='1.0'>
```

Next the type2.lit service sends a response stream header to type3.lit.

Listing 129: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt3_t2o'
```

```
        to='type3.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features.

Listing 130: Stream Features

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

We assume that type2.lit does not attempt STARTTLS negotiation but instead attempts server dialback for weak identity verification.

Listing 131: Dialback Key

```
<db:result
    from='type3.lit'
    to='type2.lit'>
  some-long-dialback-key
</db:result>
```

The type2.lit service then performs a DNS lookup on the type3.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type3.lit service.

Listing 132: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type3.lit'>
```

The authoritative server for the type3.lit service then returns a response stream header.

Listing 133: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt3_t2r'
        to='type2.lit'
        version='1.0'>
```

The type2.lit service then sends a dialback verification request to the authoritative server for the type3.lit domain.

Listing 134: Verification Request

```
<db:verify
    from='type2.lit'
    id='idt3_t2o'
    to='type3.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type3.lit domain notifies the type2.lit service that the key is valid.

Listing 135: Key is Valid

```
<db:verify
    from='type3.lit'
    id='idt3_t1o'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type2.lit service then returns a positive server dialback result to the originating server (i.e., type3.lit).

Listing 136: Server Dialback Result

```
<db:result
    from='type2.lit'
    to='type3.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type3.lit service routes the XML stanza from romeo@type3.lit to the type2.lit service.

6.3 Type 3 to Type 3

In this scenario, an XMPP user romeo@type3.lit attempts to send an XML stanza to user@example.lit:

Listing 137: Test Stanza

```
<iq from='romeo@type3.lit/foo'
    id='t3_t3'
    to='user@example.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type3.lit service (which accepts verified connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the example.lit service (which also supports verified connections and has a CA-issued certificate).
First, the type3.lit service sends an initial stream header to example.lit.

Listing 138: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type3.lit.

Listing 139: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt3_t3o'
        to='type3.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features.

Listing 140: Stream Features

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

We assume that example.lit does not attempt STARTTLS negotiation but instead attempts server dialback for weak identity verification.

Listing 141: Dialback Key

```
<db:result
    from='type3.lit'
    to='example.lit'>
  some-long-dialback-key
</db:result>
```

The example.lit service then performs a DNS lookup on the type3.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type3.lit service.

Listing 142: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        to='type3.lit'>
```

The authoritative server for the type3.lit service then returns a response stream header.

Listing 143: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt3_t3r'
        to='example.lit'
        version='1.0'>
```

The example.lit service then sends a dialback verification request to the authoritative server for the type3.lit domain.

Listing 144: Verification Request

```
<db:verify
    from='example.lit'
    id='idt3_t3o'
    to='type3.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type3.lit domain notifies the example.lit service that the key is valid.

Listing 145: Key is Valid

```
<db:verify
    from='type3.lit'
    id='idt3_t1o'
    to='example.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The example.lit service then returns a positive server dialback result to the originating server (i.e., type3.lit).

Listing 146: Server Dialback Result

```
<db:result
    from='example.lit'
    to='type3.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type3.lit service routes the XML stanza from romeo@type3.lit to the example.lit service.

6.4 Type 3 to Type 4

In this scenario, an XMPP user romeo@type3.lit attempts to send an XML stanza to hamlet@type4.lit:

Listing 147: Test Stanza

```
<iq from='romeo@type3.lit/foo'
    id='t3_t4'
    to='hamlet@type4.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type3.lit service (which accepts verified connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type4.lit service (which also supports verified connections and has a CA-issued certificate).
First, the type3.lit service sends an initial stream header to type4.lit.

Listing 148: Initial Stream Header

```
<stream:stream
```

```
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type4.lit'
        version='1.0'>
```

Next the type4.lit service sends a response stream header to type3.lit.

Listing 149: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt3_t4o'
        to='type3.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 150: Stream Features

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

Because type4.lit requires encryption, type3.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 151: STARTTLS Request

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

Listing 152: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type3.lit service then opens a new stream over the encrypted connection.

Listing 153: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
```

```
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type4.lit'
        version='1.0'>
```

Next the type4.lit service sends a response stream header to type3.lit.

Listing 154: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt3_t4o2'
        to='type3.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 155: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
  </dialback>
</stream:features>
```

Notice that type4.lit requires dialback here (perhaps because of some local service policy). Therefore type3.lit sends a dialback key to type4.lit.

Listing 156: Dialback Key

```
<db:result
    from='type3.lit'
    to='type4.lit'>
  some-long-dialback-key
</db:result>
```

The type4.lit service then performs a DNS lookup on the type3.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type3.lit service.

Listing 157: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
```

```
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type3.lit'>
```

The authoritative server for the type3.lit service then returns a response stream header.

Listing 158: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt3_t4r'
        to='type4.lit'
        version='1.0'>
```

The type4.lit service then sends a dialback verification request to the authoritative server for the type3.lit domain.

Listing 159: Verification Request

```
<db:verify
    from='type4.lit'
    id='idt3_t4o'
    to='type3.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type3.lit domain notifies the type4.lit service that the key is valid.

Listing 160: Key is Valid

```
<db:verify
    from='type3.lit'
    id='idt3_t4o'
    to='type4.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type4.lit service then returns a positive server dialback result to the originating server (i.e., type3.lit).

Listing 161: Server Dialback Result

```
<db:result
```

```
    from='type4.lit'
    to='type3.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type3.lit service routes the XML stanza from romeo@type3.lit to the type4.lit service.

6.5 Type 3 to Type 5

In this scenario, an XMPP user romeo@type3.lit attempts to send an XML stanza to bill@type5.lit:

Listing 162: Test Stanza

```
<iq from='romeo@type3.lit/foo'
    id='t3_t5'
    to='bill@type5.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type3.lit service (which accepts verified connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type5.lit service (which also supports encrypted connections and has a CA-issued certificate).
First, the type3.lit service sends an initial stream header to type5.lit.

Listing 163: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type3.lit.

Listing 164: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
```

```
        id='idt3_t5o'
        to='type3.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 165: Stream Features

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

Because type5.lit requires encryption, type3.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 166: STARTTLS Request

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

Listing 167: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type3.lit service then opens a new stream over the encrypted connection.

Listing 168: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type3.lit.

Listing 169: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt3_t5o2'
        to='type3.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 170: Stream Features

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

Notice that type5.lit requires use of SASL EXTERNAL here (because the certificate presented by type3.lit was issued by a common root CA). Therefore type3.lit attempts to complete SASL negotiation.

Listing 171: SASL Mechanism Selection

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='EXTERNAL'/>dHlwZTMubGl0</auth>
```

The type5.lit service determines that the authorization identity provided by type3.lit matches the information in the presented certificate and therefore returns success.

Listing 172: SASL Success

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

The type3.lit service then opens a new stream over the encrypted connection.

Listing 173: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type3.lit.

Listing 174: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
```

```
        id='idt3_t5o3'
        to='type3.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features (which in this case are empty).

Listing 175: Stream Features

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

Because the connection is successful, the type3.lit service routes the XML stanza from romeo@type3.lit to the type5.lit service.

6.6 Type 3 to Type 6

In this scenario, an XMPP user romeo@type3.lit attempts to send an XML stanza to chris@type6.lit:

Listing 176: Test Stanza

```
<iq from='romeo@type3.lit/foo'
    id='t3_t6'
    to='chris@type6.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type3.lit service (which accepts verified connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type6.lit service (which requires trusted communications and has a CA-issued certificate).
First, the type3.lit service sends an initial stream header to type6.lit.

Listing 177: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type6.lit'
        version='1.0'>
```

Next the type6.lit service sends a response stream header to type3.lit.

Listing 178: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt3_t6o'
        to='type3.lit'>
```

Because the type6.lit service supports XMPP 1.0, it also sends stream features.

Listing 179: Stream Features

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

Because type6.lit requires encryption, type3.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 180: STARTTLS Request

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

Listing 181: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type3.lit service then opens a new stream over the encrypted connection.

Listing 182: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type6.lit'
        version='1.0'>
```

Next the type6.lit service sends a response stream header to type3.lit.

Listing 183: Response Stream Header

```
<stream:stream
```

```
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt3_t6o2'
        to='type3.lit'>
```

Because the type6.lit service supports XMPP 1.0, it also sends stream features.

Listing 184: Stream Features

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

Notice that type6.lit requires use of SASL EXTERNAL here (because the certificate presented by type3.lit was issued by a common root CA). Therefore type3.lit attempts to complete SASL negotiation.

Listing 185: SASL Mechanism Selection

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='EXTERNAL'/>dHlwZTMubGl0</auth>
```

The type6.lit service determines that the authorization identity provided by type3.lit matches the information in the presented certificate and therefore returns success.

Listing 186: SASL Success

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

The type3.lit service then opens a new stream over the encrypted connection.

Listing 187: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type6.lit'
        version='1.0'>
```

Next the type6.lit service sends a response stream header to type3.lit.

Listing 188: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt3_t6o3'
        to='type3.lit'>
```

Because the type6.lit service supports XMPP 1.0, it also sends stream features (which in this case are empty).

Listing 189: Stream Features

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

Because the connection is successful, the type3.lit service routes the XML stanza from romeo@type3.lit to the type6.lit service.

7 Connections from Type 4 Services

7.1 Type 4 to Type 1

In this scenario, an XMPP user hamlet@type4.lit attempts to send an XML stanza to citizen@type1.lit:

Listing 190: Test Stanza

```
<iq from='hamlet@type4.lit/foo'
    id='t4_t1'
    to='citizen@type1.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type4.lit service (which requires encrypted connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type1.lit service (which supports verified connections only and does not have a certificate).
First, the type4.lit service sends an initial stream header to type1.lit.

Listing 191: Initial Stream Header

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

```
        from='type4.lit'
        to='type1.lit'
        version='1.0'>
```

Next the type1.lit service sends a response stream header to type4.lit.

Listing 192: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        id='idt4_t1o'
        to='type4.lit'>
```

Because the type1.lit service does not support XMPP 1.0, it does not send stream features. Because the type4.lit service requires encryption via TLS, it cannot proceed further with the stream negotiation and closes the stream.

Listing 193: Stream Close

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

The type1.lit service closes the stream as well.

Listing 194: Stream Close

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

Because the connection is unsuccessful, the type4.lit service returns a stanza error to hamlet@type4.lit, which should be <remote-server-timeout/>.

Listing 195: Error Stanza

```
<iq from='citizen@type1.lit'
    id='t4_t1'
    to='hamlet@type4.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

7.2 Type 4 to Type 2

In this scenario, an XMPP user hamlet@type4.lit attempts to send an XML stanza to juliet@type2.lit:

Listing 196: Test Stanza

```
<iq from='hamlet@type4.lit/foo'
    id='t4_t2'
    to='juliet@type2.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type4.lit service (which requires encrypted connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type2.lit service (which supports verified connections and has a self-signed certificate).
First, the type4.lit service sends an initial stream header to type2.lit.

Listing 197: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type2.lit'
        version='1.0'>
```

Next the type2.lit service sends a response stream header to type4.lit.

Listing 198: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt4_t2o'
        to='type4.lit'>
```

Because the type2.lit service supports XMPP 1.0, it also sends stream features.

Listing 199: Stream Features

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

Because the type4.lit service requires encryption, it attempts STARTTLS negotiation.

Listing 200: STARTTLS Request

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

Listing 201: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type4.lit service then opens a new stream over the encrypted connection.

Listing 202: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type2.lit'
        version='1.0'>
```

Next the type2.lit service sends a response stream header to type4.lit.

Listing 203: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt4_t2o2'
        to='type4.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 204: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
  </dialback>
</stream:features>
```

Notice that type2.lit requires dialback here (perhaps because of some local service policy). Therefore type4.lit sends a dialback key to type2.lit.

Listing 205: Dialback Key

```
<db:result
    from='type4.lit'
    to='type2.lit'>
  some-long-dialback-key
</db:result>
```

The type2.lit service then performs a DNS lookup on the type4.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type4.lit service.

Listing 206: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type4.lit'>
```

The authoritative server for the type4.lit service then returns a response stream header.

Listing 207: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt4_t2r'
        to='type2.lit'
        version='1.0'>
```

The type2.lit service then sends a dialback verification request to the authoritative server for the type4.lit domain.

Listing 208: Verification Request

```
<db:verify
    from='type2.lit'
    id='idt4_t2o'
    to='type4.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type4.lit domain notifies the type2.lit service that the key is valid.

Listing 209: Key is Valid

```
<db:verify
    from='type4.lit'
    id='idt4_t2o'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type2.lit service then returns a positive server dialback result to the originating server (i.e., type4.lit).

Listing 210: Server Dialback Result

```
<db:result
    from='type2.lit'
    to='type4.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type4.lit service routes the XML stanza from hamlet@type4.lit to the type2.lit service.

7.3 Type 4 to Type 3

In this scenario, an XMPP user hamlet@type4.lit attempts to send an XML stanza to romeo@type3.lit:

Listing 211: Test Stanza

```
<iq from='hamlet@type4.lit/foo'
    id='t4_t3'
    to='romeo@type3.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type4.lit service (which requires encrypted connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type3.lit service (which supports verified connections and has a CA-issued certificate).
First, the type4.lit service sends an initial stream header to type3.lit.

Listing 212: Initial Stream Header

```
<stream:stream
```

```
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type4.lit.

Listing 213: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt4_t3o'
        to='type4.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features.

Listing 214: Stream Features

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

Because the type4.lit service requires encryption, it attempts STARTTLS negotiation.

Listing 215: STARTTLS Request

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

Listing 216: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type4.lit service then opens a new stream over the encrypted connection.

Listing 217: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type4.lit.

Listing 218: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt4_t3o2'
        to='type4.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 219: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
  </dialback>
</stream:features>
```

Notice that type3.lit requires dialback here (perhaps because of some local service policy). Therefore type4.lit sends a dialback key to type3.lit.

Listing 220: Dialback Key

```
<db:result
    from='type4.lit'
    to='type3.lit'>
  some-long-dialback-key
</db:result>
```

The type3.lit service then performs a DNS lookup on the type4.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type4.lit service.

Listing 221: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        to='type4.lit'>
```

The authoritative server for the type4.lit service then returns a response stream header.

Listing 222: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt4_t3r'
        to='type3.lit'
        version='1.0'>
```

The type3.lit service then sends a dialback verification request to the authoritative server for the type4.lit domain.

Listing 223: Verification Request

```
<db:verify
    from='type3.lit'
    id='idt4_t3o'
    to='type4.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type4.lit domain notifies the type3.lit service that the key is valid.

Listing 224: Key is Valid

```
<db:verify
    from='type4.lit'
    id='idt4_t3o'
    to='type3.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type3.lit service then returns a positive server dialback result to the originating server (i.e., type4.lit).

Listing 225: Server Dialback Result

```
<db:result
    from='type3.lit'
    to='type4.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type4.lit service routes the XML stanza from hamlet@type4.lit to the type3.lit service.

7.4 Type 4 to Type 4

In this scenario, an XMPP user hamlet@type4.lit attempts to send an XML stanza to user@example.lit:

Listing 226: Test Stanza

```
<iq from='hamlet@type4.lit/foo'
    id='t4_t4'
    to='user@example.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type4.lit service (which requires encrypted connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the example.lit service (which also requires encrypted connections and has a self-signed certificate).
First, the type4.lit service sends an initial stream header to example.lit.

Listing 227: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type4.lit.

Listing 228: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt4_t4o'
        to='type4.lit'>
```

Because the example.lit service supports XMPP 1.0, it also sends stream features.

Listing 229: Stream Features

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

Because the type4.lit service requires encryption, it attempts STARTTLS negotiation.

Listing 230: STARTTLS Request

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

Listing 231: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type4.lit service then opens a new stream over the encrypted connection.

Listing 232: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type4.lit.

Listing 233: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt4_t4o2'
        to='type4.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 234: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
```

```
  </dialback>
</stream:features>
```

Notice that example.lit requires dialback here (perhaps because of some local service policy). Therefore type4.lit sends a dialback key to example.lit.

Listing 235: Dialback Key

```
<db:result
    from='type4.lit'
    to='example.lit'>
  some-long-dialback-key
</db:result>
```

The example.lit service then performs a DNS lookup on the type4.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type4.lit service.

Listing 236: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        to='type4.lit'>
```

The authoritative server for the type4.lit service then returns a response stream header.

Listing 237: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt4_t4r'
        to='example.lit'
        version='1.0'>
```

The example.lit service then sends a dialback verification request to the authoritative server for the type4.lit domain.

Listing 238: Verification Request

```
<db:verify
    from='example.lit'
    id='idt4_t4o'
```

```
    to='type4.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type4.lit domain notifies the example.lit service that the key is valid.

Listing 239: Key is Valid

```
<db:verify
    from='type4.lit'
    id='idt4_t4o'
    to='example.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The example.lit service then returns a positive server dialback result to the originating server (i.e., type4.lit).

Listing 240: Server Dialback Result

```
<db:result
    from='example.lit'
    to='type4.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type4.lit service routes the XML stanza from hamlet@type4.lit to the example.lit service.

7.5 Type 4 to Type 5

In this scenario, an XMPP user hamlet@type4.lit attempts to send an XML stanza to bill@type5.lit:

Listing 241: Test Stanza

```
<iq from='hamlet@type4.lit/foo'
    id='t4_t5'
    to='user@type4.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type4.lit service (which requires encrypted connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type3.lit service (which also requires encrypted connections and has a CA-issued certificate).
First, the type4.lit service sends an initial stream header to type5.lit.

Listing 242: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type4.lit.

Listing 243: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt4_t5o'
        to='type4.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 244: Stream Features

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

Because the type4.lit service requires encryption, it attempts STARTTLS negotiation.

Listing 245: STARTTLS Request

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

Listing 246: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type4.lit service then opens a new stream over the encrypted connection.

Listing 247: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type4.lit.

Listing 248: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt4_t5o2'
        to='type4.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 249: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
  </dialback>
</stream:features>
```

Notice that type5.lit requires dialback here (perhaps because of some local service policy). Therefore type4.lit sends a dialback key to type5.lit.

Listing 250: Dialback Key

```
<db:result
    from='type4.lit'
    to='type5.lit'>
  some-long-dialback-key
</db:result>
```

The type5.lit service then performs a DNS lookup on the type4.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type4.lit service.

Listing 251: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type4.lit'>
```

The authoritative server for the type4.lit service then returns a response stream header.

Listing 252: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt4_t5r'
        to='type5.lit'
        version='1.0'>
```

The type5.lit service then sends a dialback verification request to the authoritative server for the type4.lit domain.

Listing 253: Verification Request

```
<db:verify
    from='type5.lit'
    id='idt4_t5o'
    to='type4.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type4.lit domain notifies the type5.lit service that the key is valid.

Listing 254: Key is Valid

```
<db:verify
    from='type4.lit'
    id='idt4_t5o'
    to='type5.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type5.lit service then returns a positive server dialback result to the originating server (i.e., type4.lit).

Listing 255: Server Dialback Result

```
<db:result
    from='type5.lit'
    to='type4.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type4.lit service routes the XML stanza from hamlet@type4.lit to the type5.lit service.

7.6 Type 4 to Type 6

In this scenario, an XMPP user hamlet@type4.lit attempts to send an XML stanza to chris@type6.lit.

Listing 256: Test Stanza

```
<iq from='hamlet@type4.lit/foo'
    id='t4_t6'
    to='chris@type6.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type4.lit service (which requires encrypted connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type6.lit service (which accepts only trusted connections, has a CA-issued certificate, and does not support Server Dialback).
First, the type4.lit service sends an initial stream header to type6.lit.

Listing 257: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type6.lit'>
```

Next the type6.lit service sends a response stream header to type4.lit. Notice that the response stream header does not include the dialback namespace, since the type6.lit service does not support Server Dialback.

Listing 258: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt4_t6o'
        to='type4.lit'
        version='1.0'>
```

The type6.lit service also sends stream features. Because the type6.lit service does not accept untrusted connections, it returns stream features with a notation that STARTTLS is required.

Listing 259: Stream Features

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

Because type6.lit requires encryption, type4.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 260: STARTTLS Request

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

Listing 261: STARTTLS Response

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

The servers then attempt negotiate TLS. We assume the negotiation fails because type4.lit presents a self-signed certificate but type6.lit requires trusted federation relying on a common root CA.
Because the connection is unsuccessful, the type4.lit service returns a stanza error to hamlet@type4.lit, which should be <remote-server-timeout/>.

Listing 262: Error Stanza

```
<iq from='chris@type6.lit'
    id='t4_t6'
    to='hamlet@type4.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8 Connections from Type 5 Services

8.1 Type 5 to Type 1

In this scenario, an XMPP user bill@type5.lit attempts to send an XML stanza to citizen@type1.lit:

Listing 263: Test Stanza

```
<iq from='bill@type5.lit/foo'
    id='t5_t1'
    to='citizen@type1.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type5.lit service (which requires encrypted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type1.lit service (which supports verified connections only and does not have a certificate).
First, the type5.lit service sends an initial stream header to type1.lit.

Listing 264: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type1.lit'
        version='1.0'>
```

Next the type1.lit service sends a response stream header to type5.lit.

Listing 265: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        id='idt5_t1o'
        to='type5.lit'>
```

Because the type1.lit service does not support XMPP 1.0, it does not send stream features. Because the type5.lit service requires encryption via TLS, it cannot proceed further with the stream negotiation and closes the stream.

Listing 266: Stream Close

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

The type1.lit service closes the stream as well.

Listing 267: Stream Close

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

Because the connection is unsuccessful, the type5.lit service returns a stanza error to hamlet@type5.lit, which should be <remote-server-timeout/>.

Listing 268: Error Stanza

```
<iq from='citizen@type1.lit'
    id='t5_t1'
    to='bill@type5.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

8.2 Type 5 to Type 2

In this scenario, an XMPP user bill@type5.lit attempts to send an XML stanza to juliet@type2.lit:

Listing 269: Test Stanza

```
<iq from='bill@type5.lit/foo'
    id='t5_t2'
    to='juliet@type2.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type5.lit service (which requires encrypted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type2.lit service (which supports verified connections and has a self-signed certificate).
First, the type5.lit service sends an initial stream header to type2.lit.

Listing 270: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
```

```
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type2.lit'
        version='1.0'>
```

Next the type2.lit service sends a response stream header to type5.lit.

Listing 271: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt5_t2o'
        to='type5.lit'>
```

Because the type2.lit service supports XMPP 1.0, it also sends stream features.

Listing 272: Stream Features

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

Because the type5.lit service requires encryption, it attempts STARTTLS negotiation.

Listing 273: STARTTLS Request

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

Listing 274: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type5.lit service then opens a new stream over the encrypted connection.

Listing 275: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type2.lit'
        version='1.0'>
```

Next the type2.lit service sends a response stream header to type5.lit.

Listing 276: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt5_t2o2'
        to='type5.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 277: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
  </dialback>
</stream:features>
```

Notice that type2.lit requires dialback here (perhaps because of some local service policy). Therefore type5.lit sends a dialback key to type2.lit.

Listing 278: Dialback Key

```
<db:result
    from='type5.lit'
    to='type2.lit'>
  some-long-dialback-key
</db:result>
```

The type2.lit service then performs a DNS lookup on the type5.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type5.lit service.

Listing 279: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        to='type5.lit'>
```

The authoritative server for the type5.lit service then returns a response stream header.

Listing 280: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt5_t2r'
        to='type2.lit'
        version='1.0'>
```

The type2.lit service then sends a dialback verification request to the authoritative server for the type5.lit domain.

Listing 281: Verification Request

```
<db:verify
    from='type2.lit'
    id='idt5_t2o'
    to='type5.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type5.lit domain notifies the type2.lit service that the key is valid.

Listing 282: Key is Valid

```
<db:verify
    from='type5.lit'
    id='idt5_t2o'
    to='type2.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type2.lit service then returns a positive server dialback result to the originating server (i.e., type5.lit).

Listing 283: Server Dialback Result

```
<db:result
    from='type2.lit'
    to='type5.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type5.lit service routes the XML stanza from hamlet@type4.lit to the type2.lit service.

8.3 Type 5 to Type 3

In this scenario, an XMPP user bill@type5.lit attempts to send an XML stanza to romeo@type3.lit:

Listing 284: Test Stanza

```
<iq from='bill@type5.lit/foo'
    id='t5_t3'
    to='romeo@type3.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type5.lit service (which requires encrypted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type3.lit service (which accepts verified connections and has a CA-issued certificate).
First, the type5.lit service sends an initial stream header to type3.lit.

Listing 285: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type5.lit.

Listing 286: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt5_t3o'
        to='type5.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features.

Listing 287: Stream Features

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

Because type3.lit advertises encryption and type5.lit requires encryption, type5.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 288: STARTTLS Request

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

Listing 289: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type5.lit service then opens a new stream over the encrypted connection.

Listing 290: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type5.lit.

Listing 291: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt5_t3o2'
        to='type5.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features.

Listing 292: Stream Features

```
<stream:features>
  <mechanisms xmlns='urn:ietf:params:xml:ns:xmpp-sasl'>
```

```
    <mechanism>EXTERNAL</mechanism>
    <required/>
  </mechanisms>
</stream:features>
```

Notice that type3.lit requires use of SASL EXTERNAL here (because the certificate presented by type5.lit was issued by a common root CA). Therefore type5.lit attempts to complete SASL negotiation.

Listing 293: SASL Mechanism Selection

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='EXTERNAL'/>dHlwZTMubGl0</auth>
```

The type3.lit service determines that the authorization identity provided by type5.lit matches the information in the presented certificate and therefore returns success.

Listing 294: SASL Success

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

The type5.lit service then opens a new stream over the encrypted connection.

Listing 295: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type5.lit.

Listing 296: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt5_t3o3'
        to='type5.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features (which in this case are empty).

Listing 297: Stream Features

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

Because the connection is successful, the type5.lit service routes the XML stanza from bill@type5.lit to the type3.lit service.

8.4 Type 5 to Type 4

In this scenario, an XMPP user bill@type5.lit attempts to send an XML stanza to hamlet@type4.lit:

Listing 298: Test Stanza

```
<iq from='bill@type4.lit/foo'
    id='t5_t4'
    to='hamlet@type4.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type5.lit service (which requires encrypted connections and has a self-signed certificate) attempts to initiate a server-to-server connection with the type4.lit service (which also requires encrypted connections and has a self-signed certificate).
First, the type5.lit service sends an initial stream header to type4.lit.

Listing 299: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type4.lit'
        version='1.0'>
```

Next the type4.lit service sends a response stream header to type5.lit.

Listing 300: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt5_t4o'
        to='type5.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 301: Stream Features

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

Because the type5.lit service requires encryption, it attempts STARTTLS negotiation.

Listing 302: STARTTLS Request

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

Listing 303: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type5.lit service then opens a new stream over the encrypted connection.

Listing 304: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type4.lit'
        version='1.0'>
```

Next the type4.lit service sends a response stream header to type5.lit.

Listing 305: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt5_t4o2'
        to='type5.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 306: Stream Features

```
<stream:features>
  <dialback xmlns='urn:xmpp:features:dialback'>
    <required/>
  </dialback>
</stream:features>
```

Notice that type4.lit requires dialback here (perhaps because of some local service policy). Therefore type5.lit sends a dialback key to type4.lit.

Listing 307: Dialback Key

```
<db:result
    from='type5.lit'
    to='type4.lit'>
  some-long-dialback-key
</db:result>
```

The type4.lit service then performs a DNS lookup on the type5.lit domain, opens a TCP connection at the discovered IP address and port, and establishes a stream with the authoritative server for the type5.lit service.

Listing 308: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        to='type5.lit'>
```

The authoritative server for the type5.lit service then returns a response stream header.

Listing 309: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt5_t4r'
        to='type4.lit'
        version='1.0'>
```

The type4.lit service then sends a dialback verification request to the authoritative server for the type5.lit domain.

Listing 310: Verification Request

```
<db:verify
    from='type4.lit'
    id='idt5_t4o'
    to='type5.lit'>
  some-long-dialback-key
</db:verify>
```

Here we assume that the authoritative server for the type5.lit domain notifies the type4.lit service that the key is valid.

Listing 311: Key is Valid

```
<db:verify
    from='type5.lit'
    id='idt5_t4o'
    to='type4.lit'
    type='valid'>
  some-long-dialback-key
</db:verify>
```

The type4.lit service then returns a positive server dialback result to the originating server (i.e., type5.lit).

Listing 312: Server Dialback Result

```
<db:result
    from='type4.lit'
    to='type5.lit'
    type='valid'>
  some-long-dialback-key
</db:result>
```

Because the connection is successful, the type5.lit service routes the XML stanza from hamlet@type4.lit to the type4.lit service.

8.5 Type 5 to Type 5

In this scenario, an XMPP user bill@type5.lit attempts to send an XML stanza to user@example.lit:

Listing 313: Test Stanza

```
<iq from='bill@type5.lit/foo'
    id='t5_t5'
    to='user@example.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
```

```
</iq>
```

Therefore the type5.lit service (which requires encrypted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the example.lit service (which also requires encrypted connections and has a CA-issued certificate).
First, the type5.lit service sends an initial stream header to example.lit.

Listing 314: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type5.lit.

Listing 315: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt5_t5o'
        to='type5.lit'>
```

Because the example.lit service supports XMPP 1.0, it also sends stream features.

Listing 316: Stream Features

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

Because both example.lit requires encryption and type5.lit also requires encryption, type5.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 317: STARTTLS Request

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

Listing 318: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type5.lit service then opens a new stream over the encrypted connection.

Listing 319: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type5.lit.

Listing 320: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt5_t5o2'
        to='type5.lit'>
```

Because the example.lit service supports XMPP 1.0, it also sends stream features.

Listing 321: Stream Features

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

Notice that example.lit requires use of SASL EXTERNAL here (because the certificate presented by type5.lit was issued by a common root CA). Therefore type5.lit attempts to complete SASL negotiation.

Listing 322: SASL Mechanism Selection

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='EXTERNAL'/>dHlwZTMubGl0</auth>
```

The example.lit service determines that the authorization identity provided by type5.lit matches the information in the presented certificate and therefore returns success.

Listing 323: SASL Success

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

The type5.lit service then opens a new stream over the encrypted connection.

Listing 324: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type5.lit.

Listing 325: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt5_t5o3'
        to='type5.lit'>
```

Because the example.lit service supports XMPP 1.0, it also sends stream features (which in this case are empty).

Listing 326: Stream Features

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

Because the connection is successful, the type5.lit service routes the XML stanza from bill@type5.lit to the example.lit service.

8.6 Type 5 to Type 6

In this scenario, an XMPP user bill@type5.lit attempts to send an XML stanza to chris@type6.lit:

Listing 327: Test Stanza

```
<iq from='bill@type5.lit/foo'
    id='t5_t6'
    to='chris@type6.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type5.lit service (which requires encrypted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type6.lit service (which requires trusted communications and has a CA-issued certificate).
First, the type5.lit service sends an initial stream header to type6.lit.

Listing 328: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type6.lit'
        version='1.0'>
```

Next the type6.lit service sends a response stream header to type5.lit.

Listing 329: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt5_t6o'
        to='type5.lit'>
```

Because the type6.lit service supports XMPP 1.0, it also sends stream features.

Listing 330: Stream Features

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

Because type6.lit requires encryption, type5.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 331: STARTTLS Request

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

Listing 332: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type5.lit service then opens a new stream over the encrypted connection.

Listing 333: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type6.lit'
        version='1.0'>
```

Next the type6.lit service sends a response stream header to type5.lit.

Listing 334: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt5_t6o2'
        to='type5.lit'>
```

Because the type6.lit service supports XMPP 1.0, it also sends stream features.

Listing 335: Stream Features

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

Notice that type6.lit requires use of SASL EXTERNAL here (because the certificate presented by type5.lit was issued by a common root CA). Therefore type5.lit attempts to complete SASL negotiation.

Listing 336: SASL Mechanism Selection

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='EXTERNAL'/>dHlwZTMubGl0</auth>
```

The type6.lit service determines that the authorization identity provided by type5.lit matches the information in the presented certificate and therefore returns success.

Listing 337: SASL Success

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

The type5.lit service then opens a new stream over the encrypted connection.

Listing 338: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        to='type6.lit'
        version='1.0'>
```

Next the type6.lit service sends a response stream header to type5.lit.

Listing 339: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        id='idt5_t6o3'
        to='type5.lit'>
```

Because the type6.lit service supports XMPP 1.0, it also sends stream features (which in this case are empty).

Listing 340: Stream Features

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

Because the connection is successful, the type5.lit service routes the XML stanza from bill@type5.lit to the type6.lit service.

9 Connections from Type 6 Services

9.1 Type 6 to Type 1

In this scenario, an XMPP user chris@type6.lit attempts to send an XML stanza to citizen@type1.lit:

Listing 341: Test Stanza

```
<iq from='chris@type6.lit/foo'
    id='t6_t1'
    to='citizen@type1.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type6.lit service (which requires trusted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type1.lit service (which supports verified connections only and does not have a certificate).
First, the type6.lit service sends an initial stream header to type1.lit.

Listing 342: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type1.lit'
        version='1.0'>
```

Next the type1.lit service sends a response stream header to type6.lit.

Listing 343: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type1.lit'
        id='idt6_t1o'
        to='type6.lit'>
```

Because the type1.lit service does not support XMPP 1.0, it does not send stream features. Because the type6.lit service requires encryption via TLS, it cannot proceed further with the stream negotiation and closes the stream.

Listing 344: Stream Close

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

The type1.lit service closes the stream as well.

Listing 345: Stream Close

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

Because the connection is unsuccessful, the type6.lit service returns a stanza error to chris@type6.lit, which should be <remote-server-timeout/>.

Listing 346: Error Stanza

```
<iq from='citizen@type1.lit'
    id='t6_t1'
    to='bill@type5.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

9.2 Type 6 to Type 2

In this scenario, an XMPP user chris@type6.lit attempts to send an XML stanza to juliet@type2.lit:

Listing 347: Test Stanza

```
<iq from='chris@type6.lit/foo'
    id='t6_t2'
    to='juliet@type2.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type6.lit service (which requires encrypted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type2.lit service (which supports verified connections and has a self-signed certificate).
First, the type6.lit service sends an initial stream header to type2.lit.

Listing 348: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
```

```
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type2.lit'
        version='1.0'>
```

Next the type2.lit service sends a response stream header to type6.lit.

Listing 349: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type2.lit'
        id='idt6_t2o'
        to='type6.lit'>
```

Because the type2.lit service supports XMPP 1.0, it also sends stream features.

Listing 350: Stream Features

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

Because the type6.lit service requires encryption, it attempts STARTTLS negotiation.

Listing 351: STARTTLS Request

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

Listing 352: STARTTLS Response

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

The servers then attempt negotiate TLS. We assume the negotiation fails because type2.lit presents a self-signed certificate but type6.lit requires trusted federation relying on a common root CA.
Because the connection is unsuccessful, the type6.lit service returns a stanza error to chris@type6.lit, which should be <remote-server-timeout/>.

Listing 353: Error Stanza

```
<iq from='juliet@type2.lit'
    id='t4_t6'
    to='chris@type6.lit/foo'
```

```
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

9.3 Type 6 to Type 3

In this scenario, an XMPP user chris@type6.lit attempts to send an XML stanza to romeo@type3.lit:

Listing 354: Test Stanza

```
<iq from='chris@type6.lit/foo'
    id='t6_t3'
    to='romeo@type3.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type6.lit service (which requires trusted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type3.lit service (which accepts verified connections and has a CA-issued certificate).
First, the type6.lit service sends an initial stream header to type3.lit.

Listing 355: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type6.lit.

Listing 356: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt6_t3o'
        to='type6.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features.

Listing 357: Stream Features

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

Because type3.lit advertises encryption and type6.lit requires encryption, type6.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 358: STARTTLS Request

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

Listing 359: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type6.lit service then opens a new stream over the encrypted connection.

Listing 360: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type6.lit.

Listing 361: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt6_t3o2'
        to='type6.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features.

Listing 362: Stream Features

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

Notice that type3.lit requires use of SASL EXTERNAL here (because the certificate presented by type6.lit was issued by a common root CA). Therefore type6.lit attempts to complete SASL negotiation.

Listing 363: SASL Mechanism Selection

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='EXTERNAL'/>dHlwZTMubGl0</auth>
```

The type3.lit service determines that the authorization identity provided by type6.lit matches the information in the presented certificate and therefore returns success.

Listing 364: SASL Success

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

The type6.lit service then opens a new stream over the encrypted connection.

Listing 365: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type3.lit'
        version='1.0'>
```

Next the type3.lit service sends a response stream header to type6.lit.

Listing 366: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type3.lit'
        id='idt6_t3o3'
        to='type6.lit'>
```

Because the type3.lit service supports XMPP 1.0, it also sends stream features (which in this case are empty).

Listing 367: Stream Features

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

Because the connection is successful, the type6.lit service routes the XML stanza from chris@type6.lit to the type3.lit service.

9.4 Type 6 to Type 4

In this scenario, an XMPP user chris@type6.lit attempts to send an XML stanza to hamlet@type4.lit:

Listing 368: Test Stanza

```
<iq from='chris@type6.lit/foo'
    id='t6_t4'
    to='hamlet@type4.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type6.lit service (which requires encrypted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type4.lit service (which supports verified connections and has a self-signed certificate).
First, the type6.lit service sends an initial stream header to type4.lit.

Listing 369: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type4.lit'
        version='1.0'>
```

Next the type4.lit service sends a response stream header to type6.lit.

Listing 370: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
```

```
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type4.lit'
        id='idt6_t4o'
        to='type6.lit'>
```

Because the type4.lit service supports XMPP 1.0, it also sends stream features.

Listing 371: Stream Features

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

Because the type6.lit service requires encryption, it attempts STARTTLS negotiation.

Listing 372: STARTTLS Request

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

Listing 373: STARTTLS Response

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

The servers then attempt negotiate TLS. We assume the negotiation fails because type4.lit presents a self-signed certificate but type6.lit requires trusted federation relying on a common root CA.
Because the connection is unsuccessful, the type6.lit service returns a stanza error to chris@type6.lit, which should be <remote-server-timeout/>.

Listing 374: Error Stanza

```
<iq from='juliet@type4.lit'
    id='t6_t4'
    to='chris@type6.lit/foo'
    type='error'>
  <error type='cancel'>
    <remote-server-timeout
        xmlns='urn:ietf:params:xml:ns:xmpp-stanzas'/>
  </error>
</iq>
```

9.5 Type 6 to Type 5

In this scenario, an XMPP user chris@type6.lit attempts to send an XML stanza to bill@type5.lit:

Listing 375: Test Stanza

```
<iq from='chris@type6.lit/foo'
    id='t6_t5'
    to='bill@type5.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type6.lit service (which requires trusted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the type5.lit service (which requires encrypted connections and has a CA-issued certificate).
First, the type6.lit service sends an initial stream header to type5.lit.

Listing 376: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type6.lit.

Listing 377: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt6_t5o'
        to='type6.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 378: Stream Features

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

Because type5.lit advertises encryption and type6.lit requires encryption, type6.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 379: STARTTLS Request

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

Listing 380: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type6.lit service then opens a new stream over the encrypted connection.

Listing 381: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type6.lit.

Listing 382: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt6_t5o2'
        to='type6.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features.

Listing 383: Stream Features

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

Notice that type5.lit requires use of SASL EXTERNAL here (because the certificate presented by type6.lit was issued by a common root CA). Therefore type6.lit attempts to complete SASL negotiation.

Listing 384: SASL Mechanism Selection

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='EXTERNAL'/>dHlwZTMubGl0</auth>
```

The type5.lit service determines that the authorization identity provided by type6.lit matches the information in the presented certificate and therefore returns success.

Listing 385: SASL Success

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

The type6.lit service then opens a new stream over the encrypted connection.

Listing 386: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='type5.lit'
        version='1.0'>
```

Next the type5.lit service sends a response stream header to type6.lit.

Listing 387: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type5.lit'
        id='idt6_t5o3'
        to='type6.lit'>
```

Because the type5.lit service supports XMPP 1.0, it also sends stream features (which in this case are empty).

Listing 388: Stream Features

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

Because the connection is successful, the type6.lit service routes the XML stanza from chris@type6.lit to the type5.lit service.

9.6 Type 6 to Type 6

In this scenario, an XMPP user chris@type6.lit attempts to send an XML stanza to user@example.lit:

Listing 389: Test Stanza

```
<iq from='chris@type6.lit/foo'
    id='t6_t6'
    to='user@example.lit'
    type='get'>
  <ping xmlns='urn:xmpp:ping'/>
</iq>
```

Therefore the type6.lit service (which requires trusted connections and has a CA-issued certificate) attempts to initiate a server-to-server connection with the example.lit service (which requires encrypted connections and has a CA-issued certificate).
First, the type6.lit service sends an initial stream header to example.lit.

Listing 390: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type6.lit.

Listing 391: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt6_t6o'
        to='type6.lit'>
```

Because the example.lit service supports XMPP 1.0, it also sends stream features.

Listing 392: Stream Features

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

```
  </starttls>
  <dialback xmlns='urn:xmpp:features:dialback'/>
</stream:features>
```

Because example.lit advertises encryption and type6.lit requires encryption, type6.lit attempts to negotiate a STARTTLS upgrade to the stream.

Listing 393: STARTTLS Request

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

Listing 394: STARTTLS Response

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

The servers then negotiate TLS. We assume the negotiation is successful.
The type6.lit service then opens a new stream over the encrypted connection.

Listing 395: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type6.lit.

Listing 396: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt6_t6o2'
        to='type6.lit'>
```

Because the example.lit service supports XMPP 1.0, it also sends stream features.

Listing 397: Stream Features

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

Notice that example.lit requires use of SASL EXTERNAL here (because the certificate presented by type6.lit was issued by a common root CA). Therefore type6.lit attempts to complete SASL negotiation.

Listing 398: SASL Mechanism Selection

```
<auth xmlns='urn:ietf:params:xml:ns:xmpp-sasl'
      mechanism='EXTERNAL'/>dHlwZTMubGl0</auth>
```

The example.lit service determines that the authorization identity provided by type6.lit matches the information in the presented certificate and therefore returns success.

Listing 399: SASL Success

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

The type6.lit service then opens a new stream over the encrypted connection.

Listing 400: Initial Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='type6.lit'
        to='example.lit'
        version='1.0'>
```

Next the example.lit service sends a response stream header to type6.lit.

Listing 401: Response Stream Header

```
<stream:stream
        xmlns='jabber:server'
        xmlns:db='jabber:server:dialback'
        xmlns:stream='http://etherx.jabber.lit/streams'
        from='example.lit'
        id='idt6_t6o3'
        to='type6.lit'>
```

Because the example.lit service supports XMPP 1.0, it also sends stream features (which in this case are empty).

Listing 402: Stream Features

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

Because the connection is successful, the type6.lit service routes the XML stanza from chris@type6.lit to the example.lit service.

10 Security Considerations

As explained in RFC 3920 and XEP-0220, Server Dialback does not provide authentication.
In the absence of out-of-band key exchange, acceptance of a self-signed certificate does not result in authentication of a peer and therefore should be followed by Server Dialback to weakly verify peer identity.
Acceptance of a certificate issued by a trusted root CA results in some level of authentication and therefore should be followed by SASL negotiation using the EXTERNAL mechanism.

11 IANA Considerations

This document requires no interaction with the Internet Assigned Numbers Authority (IANA) [4].

12 XMPP Registrar Considerations

This document requires no interaction with the XMPP Registrar [5].

13 Acknowledgements

Thanks to Philipp Hancke, Norman Rasmussen, and Tomasz Sterna for their feedback.

[4] The Internet Assigned Numbers Authority (IANA) is the central coordinator for the assignment of unique parameter values for Internet protocols, such as port numbers and URI schemes. For further information, see <http://www.iana.org/>.

[5] The XMPP Registrar maintains a list of reserved protocol namespaces as well as registries of parameters used in the context of XMPP extension protocols approved by the XMPP Standards Foundation. For further information, see <https://xmpp.org/registrar/>.