XEP-0388: Extensible SASL Profile

Dave Cridland
mailto:dave@hellopando.com
xmpp:dwd@dave.cridland.net

Thilo Molitor
mailto:thilo+xmpp@eightysoft.de
xmpp:thilo.molitor@juforum.de

Matthew Wild
mailto:mwild1@gmail.com
xmpp:me@matthewwild.co.uk

2024-08-06
Version 1.0.2

Status	Type	Short Name
Draft	Standards Track	sasl2

This document describes a replacement for the SASL profile documented in RFC 6120 which allows for greater extensibility.

Contents

1 Introduction

While SASL provides an excellent framework that has served us well over the past 18 years, a number of shortcomings in the profile - the syntax binding to XMPP - that is in use.
This specification addresses a number of shortfalls:

- Number of round trips
- Extensibility
- Support for second factor
- Support for mandatory password changes

The new SASL profile documented herein is primarily a syntactic change to allow extensibility, combined with removal of the (largely) redundant stream restart, and additional results beyond total success or abject failure.

1.1 Terminology

Although initiating entities, in general, use SASL, and receiving entities offer it, the SASL specification and common parlance both use "Client " and "Server"; this specification uses Client and Server and assumes C2S links. This is not intended to preclude use of this SASL profile on S2S links. The term "SASL2" is used to mean the new SASL profile specified in this document; however the same RFC 4422 definition of SASL (and SASL profiles) applies.
Examples often use hypothetical SASL mechanisms and sub-extensions; this specification does not intend to make a position on any particular SASL mechanism, and the Mandatory To Implement mechanisms are unaffected.

2 Overview

2.1 SASL mechanism negotiation

Servers capable of SASL2 offer a stream feature of <authentication/>, qualified by the "urn:xmpp:sasl:2" namespace. This in turn contains one or more <mechanism/> elements in the same namespace, and potentially other elements (for example, the <hostname/> element defined within Use of Domain-Based Service Names in XMPP SASL Negotiation (XEP-0233) [1]). The mechanism elements MUST only offer mechanisms the server can offer for the JID provided by the client in the "from" attribute of the stream-header. If that attribute is omitted or does not contain a recognised bare JID, the server SHOULD only offer mechanisms it can provide independently of the JID. Note that varying responses based on whether an account

[1] XEP-0233: Use of Domain-Based Service Names in XMPP SASL Negotiation <https://xmpp.org/extensions/xep-0233.html>.

exists or not may leak the existence of accounts to unauthorized parties. See the security considerations section for more information. If the ”from” attribute is not empty and the domain used in this attribute does not match the domain used in the ”to” attribute the server SHOULD respond with an invalid-from error as defined in §4.9.3.9 of RFC 6120 [2].
The feature so advertised, and its child content, SHOULD be stable for the given stream to and from attributes and encryption state, and therefore MAY be cached by clients for later connections.
Note that SASL2 is impossible for clients to initiate without at least one mechanism being available, and therefore MUST NOT be offered if not at least one mechanism can be provided.
The Service Name used by XMPP is unchanged from RFC 6120 [3].
All servers and clients supporting channel-binding MUST implement SASL Channel-Binding Type Capability (XEP-0440) [4].
Additional stream features that can be negotiated as part of a successful SASL authentication can be included in the <inline/> element which is an immediate child of <authentication/>.

Listing 1: Client sending stream header

```
<?xml version='1.0'?>
<stream:stream
  from='user@example.org'
  to='example.org'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
```

Listing 2: Server responding with stream header and features

```
<?xml version='1.0'?>
<stream:stream
  from='example.org'
  id='++TR84Sm6A3hnt3Q065SnAbbk3Y='
  to='user@example.org'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>
<stream:features>
  <authentication xmlns='urn:xmpp:sasl:2'>
    <mechanism>SCRAM-SHA-1</mechanism>
    <mechanism>SCRAM-SHA-1-PLUS</mechanism>
    <inline>
      <!-{}- Server indicates that XEP-0198 stream resumption can be
          done "inline" -{}->
```

[2]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[3]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[4]XEP-0440: SASL Channel-Binding Type Capability <https://xmpp.org/extensions/xep-0440.html>.

```
      <sm xmlns='urn:xmpp:sm:3'/>
      <!-{}- Server indicates support for XEP-0386 Bind 2 -{}->
      <bind xmlns='urn:xmpp:bind:0'/>
    </inline>
  </authentication>
  <!-{}- Channel-binding information provided by XEP-0440 -{}->
  <sasl-channel-binding xmlns='urn:xmpp:sasl-cb:0'>
    <channel-binding type='tls-server-end-point'/>
    <channel-binding type='tls-exporter'/>
  </sasl-channel-binding>
</stream:features>
```

2.2 SASL Data Encoding

In all cases, both Clients and Servers encode SASL exchanges using Base 64 encoding. This SHOULD NOT include any line wrapping or other whitespace. As the form <element/> is equivalent to <element></element>, these both indicate an empty string. Challenges and responses with no data do not occur in SASL, and so require no special handling. To indicate the absence of an initial response, or the absence of success data, the element is simply not included.

2.3 Initiation

Clients, upon observing this stream feature, initiate the authentication by the use of the <authenticate/> top-level element within the same namespace. The nature of this element is to inform the server about properties of the final stream state, as well as initiate authentication itself. To achieve this, it has a single mandatory attribute of "mechanism", with a string value of a mechanism name offered by the Server in the stream feature and an optional child element of <initial-response/>, containing a base64-encoded SASL Initial Response. If the "mechanism" attribute contains a string not previously announced by the server in in the stream feature, the server MUST fail the authentication.
If the stream's from attribute (if present) does not match the non-empty authorization string, the server MUST fail the authentication as defined in Failure.
Clients SHOULD also include a <user-agent/> element, informing the server about the connecting client. The 'id' attribute is RECOMMENDED, and if present contains a unique stable identifier for the client installation. The contents of the 'id' attribute MUST be a UUID v4. This allows the server to provide functionality such as deriving stable resource identifiers (see Bind 2.0 (XEP-0386) [5]). The child elements <software/> and <device/> MAY be supplied by clients and contain text descriptions of the client software and the device it is installed on. These allow the server to keep the user informed about what devices are connected to their account. Servers MUST NOT expose this information to other entities (such functionality is available in Software Version (XEP-0092) [6] if required).

[5]XEP-0386: Bind 2.0 <https://xmpp.org/extensions/xep-0386.html>.
[6]XEP-0092: Software Version <https://xmpp.org/extensions/xep-0092.html>.

Listing 3: An authentication request

```
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='SCRAM-SHA-1-PLUS'>
  <!-{}- Base64 of: 'p=tls-exporter,,n=user,r=12C4CD5C-E38E-4A98-8F6D
      -15C38F51CCC6' -{}->
  <initial-response>
      cD10bHMtZXhwb3J0ZXIsLG49dXNlcixyPTEyQzRDRDVDLUUzOEUtNEE5OC04RjZELTE1QzM4RjUxQ0ND
      ==</initial-response>
  <user-agent id='d4565fa7-4d72-4749-b3d3-740edbf87770'>
    <software>AwesomeXMPP</software>
    <device>Kiva's␣Phone</device>
␣␣</user-agent>
</authenticate>
```

In order to provide support for other desired stream states beyond authentication, additional child elements are used. For example, a hypothetical XEP-0198 session resumption element might be included, and/or Resource Binding requests.

Listing 4: An authentication request with a (hypothetical) bind request

```
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='BLURDYBLOOP'>
  <initial-response>
     SSBzaG91bGQgbWFrZSB0aGlzIGEgY29tcGV0aXRpb24=
  </initial-response>
  <user-agent id='d4565fa7-4d72-4749-b3d3-740edbf87770'>
    <software>AwesomeXMPP</software>
    <device>Kiva's␣Phone</device>
␣␣</user-agent>
␣␣<bind␣xmlns='urn:xmpp:bind:example'/>
</authenticate>
```

2.4 During Authentication

At any time while authentication is in progress, neither Client nor Server sends any element (including stanzas) or other data except the top-level elements defined herein. Clients MUST NOT send whitespace, and MUST send only <response/> elements as appropriate or an <abort/> element to immediately cause an error. Servers MUST disconnect Clients immediately if any other traffic is received. Servers are similarly REQUIRED to send no whitespace, and only the <response/> and completion elements from the section below.

2.5 Challenges and Responses

Server Challenges MAY then be sent. Each Challenge MUST be responded to by a Client in a Client Response. These are not extensible, and contain the corresponding base64 encoded

SASL data:

Listing 5: A challenge and response exchange (SCRAM-SHA-1-PLUS)

```
<!-{}-
  SCRAM-SHA-1-PLUS challenge issued by the server as defined in RFC
      5802.
  Base64 of: 'r=12C4CD5C-E38E-4A98-8F6D-15C38F51CCC6a09117a6-ac50-4f2f
      -93f1-93799c2bddf6,s=QSXCR+Q6sek8bf92,i=4096'
-{}->
<challenge xmlns='urn:xmpp:sasl:2'>
  cj0xMkM0Q0Q1Qy1FMzhFLTRBOTgtOEY2RC0xNUMzOEY1MUNDQzZhMDkxMTdhNi1hYzUwLTRmMmYtOTNmMS0
      ==
</challenge>

<!-{}-
  The client responds with the base64 encoded SCRAM-SHA-1-PLUS client-
      final-message (password: 'pencil').
  Base64 of: 'c=
      cD10bHMtZXhwb3J0ZXIsLMcoQvOdBDePd4OswlmAWV3dg1a1Wh1tYPTBwVid10VU
      ,r=12C4CD5C-E38E-4A98-8F6D-15C38F51CCC6a09117a6-ac50-4f2f-93f1
      -93799c2bddf6,p=UApo7xo6Pa9J+Vaejfz/dG7BomU='
  The c-attribute contains the GS2-header and channel-binding data
      blob (32 bytes) as defined in RFC 5802.
-{}->
<response xmlns='urn:xmpp:sasl:2'>
  Yz1jRDEwYkhNdFpYaHdiM0owWlhJc0xNY29Rdk9kQkRlUGQ0T3N3bG1BV1YzZGcxYTFXaDF0WVBUQndWaWQ
      ==
</response>
```

Listing 6: A challenge and response exchange (hypothetical)

```
<!-{}- A server might send: -{}->
<challenge xmlns='urn:xmpp:sasl:2'>
  U28sIG5leHQgRk9TREVNIC0gMjAxOCwgdGhhdCBpcy4uLg==
</challenge>

<!-{}- A client might respond: -{}->
<response xmlns='urn:xmpp:sasl:2'>
  Li4uSSdsbCBidXkgYSBiZWVyIGZvciB0aGUgZmlyc3QgcGVyc29uIHdoby4uLg==
</response>
```

2.6 Completing Authentication

Authentication may complete in one of three ways. It may complete successfully, in which case the client is authenticated. It may also fail, in which case the client is not authenticated and the stream and session state remain entirely unchanged.
Finally, it may have completed successfully, but further interaction is required - for example,

a password change or second-factor authentication.

2.6.1 Success

If the Client is now authenticated, the Server sends a <success/> element, which contains an <authorization-identifier/> element containing the negotiated identity - this is a bare JID, unless resource binding has occurred, in which case it is a full JID.
If the <success/> element it the response to a SASL mechanism exchange, it MAY contain an <additional-data> element, containing additional data from the SASL mechanism that has just completed.
If the <success/> element is the response to a task exchange initiated by a <continue/> element, it MAY contain any other element defined in a future protocol containing additional data from the task that has just completed
If the client requested any stream features inline as part of the SASL negotiation, they are processed by the server at this point.

Listing 7: Successful authentication with SCRAM-SHA-1-PLUS

```
<success xmlns='urn:xmpp:sasl:2'>
  <!-{}- Base64 of: 'v=msVHs/BzIOHDqXeVH7EmmDu9id8=' -{}->
  <additional-data>
    dj1tc1ZIcy9CeklPSERxWGVWSDdFbW1EdTlpZDg9
  </additional-data>
  <authorization-identifier>user@example.org</authorization-identifier
      >
</success>
```

Listing 8: Successful authentication with BLURDYBLOOP

```
<success xmlns='urn:xmpp:sasl:2'>
  <additional-data>
    ip/AeIOfZXKBV+fW2smE0GUB3I//nnrrLCYkt0Vj
  </additional-data>
  <authorization-identifier>juliet@montague.example/Balcony/
     a987dsh9a87sdh</authorization-identifier>
</success>
```

The results of activating any requested inline stream features are included in the server's <success/> response.

Listing 9: Successful re-authentication and resumption using BLURDYBLOOP

```
<success xmlns='urn:xmpp:sasl:2'>
  <additional-data>
    SGFkIHlvdSBnb2luZywgdGhlcmUsIGRpZG4ndCBJPw==
  </additional-data>
  <authorization-identifier>juliet@montague.example</authorization-
     identifier>
  <resumed xmlns='urn:xmpp:sm:3' h='345' previd='124'/>
</success>
```

Any security layer negotiated SHALL take effect after the ">" octet of the closing tag (ie, immediately after "</success>"), if it has not already taken effect at a <continue> - see Continue below.
The <success> element is immediately followed by a <features> element in the "http://etherx.jabber.org/streams" namespace containing the applicable stream features of the newly authenticated stream. Note that no stream restart occurs.

2.6.2 Failure

A <failure/> element is used by the server to terminate the authentication attempt. It MAY contain application-specific error codes, and MAY contain a textual error. It MUST contain one of the SASL error codes from RFC 6120 Section 6.5.
The server MUST NOT process any inline features requested by the client in a failed authentication request, if any.

Listing 10: Failure

```
<failure xmlns='urn:xmpp:sasl:2'>
  <aborted xmlns='urn:ietf:params:xml:ns:xmpp-sasl'/>
  <optional-application-specific xmlns='urn:something:else'/>
  <text>This is a terrible example.</text>
</failure>
```

2.6.3 Continue

A <continue/> element is used to indicate that while the SASL exchange was successful, it is insufficient to allow authentication at this time. This can be used to indicate, for example, that the Client needs to perform a Second Factor Authentication ("2FA"), or is required to change password (this list is not meant to be exhaustive in any way).
Such tasks are presented within a <tasks> element, which contains a sequence of <task> elements, each containing a name. These tasks are analogous to a SASL mechanism, but have a number of differences - they may never attempt to negotiate a new authorization identifier, nor a new security layer.
Like the <success/> element, the <continue/> element MAY contain an <additional-data/> element containing additional data from the SASL mechanism that has completed.
A client MAY choose any one of the offered tasks; if multiple are required a sequence of <continue> exchanges will occur until all mandatory tasks are complete.
The <continue element therefore always contains a <tasks/> element, as defined above. It MAY contain an <additional-data/> element, as the <success/> element does.

Finally, it MAY contain a <text/> element, which can contain human-readable data explaining the nature of the step required.

Listing 11: Continue Required

```
<continue xmlns='urn:xmpp:sasl:2'>
  <additional-data>
    SSdtIGJvcmVkIG5vdy4=
  </additional-data>
  <tasks>
    <task>HOTP-EXAMPLE</task>
    <task>TOTP-EXAMPLE</task>
  </tasks>
  <text>This account requires 2FA</text>
</continue>
```

After the final octet of the first <continue> element, any SASL security layer negotiated in the preceding exchange SHALL be immediately in effect.
Clients respond with a <next/> element, which has a single mandatory attribute of "task", containing the selected task name, and MAY contain any other element as defined in a future protocol defining this task.
The concrete elements exchanged for each task after the <next/> reside inside a <task-data/> wrapper element in the namespace "urn:xmpp:sasl:2". Each wrapper element can contain any other element as defined in a future protocol defining this concrete task. Each task MUST end either by the server sending a <failure/> element, if the task failed, a <continue/> element, if the task was completed successfully and the server requests the client to perform a new task, or a <success/> element, indicating that the task was completed successfully and no further tasks are needed.

Listing 12: Fictional TOTP task

```
<!-{}- Client starts TOTP-EXAMPLE task -{}->
<next xmlns='urn:xmpp:sasl:2' task='TOTP-EXAMPLE'>
  <totp xmlns="urn:totp:example">
    SSd2ZSBydW4gb3V0IG9mIGlkZWFzIGhlcmUu
  </totp>
</next>

<!-{}- Server provides needed data to Client -{}->
<task-data xmlns='urn:xmpp:sasl:2'>
  <totp xmlns="urn:totp:example">
    94d27acffa2e99a42ba7786162a9e73e7ab17b9d
  </totp>
</task-data>

<!-{}- Client responds with requested TOP data -{}->
<task-data xmlns='urn:xmpp:sasl:2'>
  <totp xmlns="urn:totp:example">
```

```
    OTRkMjdhY2ZmYTJlOTlhNDJiYTc3ODYxNjJhOWU3M2U3YWIxN2I5ZAo=
  </totp>
</task-data>

<!-{}- Server indicates successful completion of TOTP-EXAMPLE task
   -{}->
<success xmlns='urn:xmpp:sasl:2'>
  <totp xmlns="urn:totp:example">
    SGFkIHlvdSBnb2luZywgdGhlcmUsIGRpZG4ndCBJPw==
  </totp>
  <authorization-identifier>juliet@montague.example</authorization-
      identifier>
</success>
```

3 Security Considerations

Relative to the SASL profile documented in RFC 6120, this introduces more data unprotected by any security layer negotiated by SASL itself. While no actual exchanges are introduced that are unprotected, the nature of this exchange might allow for (for example) a resource binding extension to be introduced. SASL security layers are sparingly used in the field, however, so this is thought to be a theoretical, rather than practical, concern. TLS, if negotiated before authentication, could mitigate these risks further.
As explained in § 13.8.3 of RFC 6120 [7], servers and clients SHOULD NOT support SASL PLAIN. Usage of PLAIN allows for trivial downgrade atacks by Man-In-The-Middle attackers circumventing any possible channel-binding or mutual authentication and even allowing the attacker to gain access to the cleartext password. If possible, PLAIN should be disabled by default on servers and clients and only be enabled when configured by the admin respective client user (if at all).
This protocol makes use of the hint of its identity contained in the stream "from" attribute as defined in RFC 6120 [8] prior to authentication. This value MUST NOT be trusted to contain the real identity of the connecting client. Varying pre-authentication responses (such as supported SASL mechanisms) based on this value may allow unauthorized parties to discover whether an account exists, and even guess at what software might be used with that account (e.g. by observing which SASL mechanisms can be used on this account). Minimizing differences between the default response for unregistered JIDs and registered JIDs, as well as rate-limiting (to prevent enumeration attempts), are possible countermeasures. If the server does not know the client-identity it is RECOMMENDED to randomize the list of provided SASL mechanisms to reduce these risks even further.
SASL2 MUST only be used by Clients or offered by Servers after TLS negotiation, see § 5.2 and § 5.3.4 of RFC 6120 [9].
Clients MUST NOT send, and servers MUST NOT process, authentication requests included in

[7]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[8]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.
[9]RFC 6120: Extensible Messaging and Presence Protocol (XMPP): Core <http://tools.ietf.org/html/rfc6120>.

the TLS 0-RTT ("early data") extension. Such data is vulnerable to replay attacks, and without adequate protection this could allow an attacker to disrupt established sessions or cause other side-effects depending on the inline features negotiated. This rule may be overridden by later specifications, provided they define appropriate protocols to mitigate these issues. Examples of these attacks and mitigations are discussed in Section 8 of RFC 8446 [10].

4 SASL Profile Definition

This provides pointers and/or clarifications to the Overview in the order and manner defined in RFC 4422, section 4.

4.1 Service Name

The service name SHALL be "xmpp", as defined by RFC 6120.

4.2 Mechanism negotiation

Servers list mechanisms in the <mechanisms/> list in response to a <request/> by the client. (See Initiation).

4.3 Message Definitions

4.3.1 Initiation

Clients initiate using the <authenticate/> top level element (See Initiation.

4.3.2 Server Challenges and Client Responses

See Challenges and Responses.

4.3.3 Outcome

See Completing Authentication.

[10] RFC 8446: The Transport Layer Security (TLS) Protocol Version 1.3 <http://tools.ietf.org/html/rfc8446>.

4.4 Non-Empty Authorization Strings

If a Client specifies an authorization string which is non-empty, the identifier is normalized by treating it as a JID, and performing normalization as described in RFC 7622.
Any non-empty authorization string MUST be considered an error if the stream's from attribute (if present) does not match.

4.5 Aborting

Clients MAY abort unilaterally at any time before the authentication completed by sending an <abort/> element. This element MAY contain a <text/> element containing a textual representation of the reason as well as any other element defined in some future protocol.
Servers MAY abort unliterally by sending <failure/> with the <aborted/> error code as defined in Failure.

4.6 Security Layer Effect

Security Layers take effect after the SASL mechanism itself (ie, the first negotiation) has completed successfully, after the final octet of the server's <success> or <continue>. See Success and Continue.

4.7 Security Layer Order

Option (a) is used - any SASL Security Layer is applied first to data being sent, and TLS applied last.

4.8 Multiple Authentication

Although the <continue/> concept does use tasks analogous to multiple SASL sequences, only the first SASL mechanism used is considered an authentication, and only the first can negotiate a security layer.
In particular, once <success/> or <continue/> has been sent by the server, any further <authenticate/> element MUST result in a stream error.

5 Example Flows

This section provides some example flows. It aims to demonstrate the structure of SASL2 negotiation, but it does not definitively describe any of the additional extensions shown here.

5.1 Syntactic Equivalence

Where no additional features that SASL2 makes available are used, the flow of information is identical to the original SASL profile. This example shows the new syntax and draws the reader's attention to the differences.

Listing 13: PLAIN Authentication

```
<!-{}-
  Client sends stream header
-{}->
<stream:stream
  from='alice@example.org'
  to='example.org'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>

<!-{}-
  Server sends stream features.
  These are identical to SASL1, but within a different namespace.
-{}->
<stream:features>
  <authentication xmlns='urn:xmpp:sasl:2'>
    <mechanism>PLAIN</mechanism>
    <mechanism>SCRAM-SHA-1</mechanism>
  </authentication>
</stream:features>

<!-{}-
  Client intiates authentication.
  Beyond the element local name and namespace,
  the main distinction is that initial-response data is held within an
      element,
  so the ”=” special case no longer applies.
-{}->
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='PLAIN'>
  <initial-response>AGFsaWNlQGV4YW1wbGUub3JnCjM0NQ==</initial-response
      >
  <user-agent id='d4565fa7-4d72-4749-b3d3-740edbf87770'>
    <software>AwesomeXMPP</software>
    <device>Kiva's␣Phone</device>
␣␣</user-agent>
</authenticate>

<!-{}-
␣␣This␣completes␣in␣one␣step,␣so␣the␣Server␣sends␣a␣success.
␣␣A␣SASL2␣success␣always␣includes␣the␣authorization␣identifier:
```

```
-{}->
<success␣xmlns='urn:xmpp:sasl:2'>
␣␣<authorization-identifier>alice@example.org</authorization-
    identifier>
</success>

<!-{}-
␣␣The␣server␣immediately␣sends␣a␣new␣set␣of␣stream␣features␣at␣this␣
    point.
␣␣There␣is␣no␣stream␣restart.
-{}->
<stream:features>
␣␣<bind␣xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
␣␣␣␣<required/>
␣␣</bind>
</stream:features>

<!-{}-
␣␣The␣client␣now␣continues␣with␣resource␣binding.
-{}->
```

Use of SASL2 in this simple scenario saves one round-trip (due to the lack of stream restart).

5.2 Once More, This Time With CRAM

In this example, multiple potential round-trips are saved by negotiating resource binding and stream management inline.

Listing 14: CRAM-MD5 Authentication

```
<!-{}-
  Client sends stream header
-{}->
<stream:stream
  from='tim@example.org'
  to='example.org'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>

<!-{}-
  Server sends stream features.
-{}->
<stream:features>
  <mechanisms xmlns='urn:xmpp:sasl:2'>
    <mechanism>PLAIN</mechanism>
    <mechanism>CRAM-MD5</mechanism>
  </mechanisms>
</stream:features>

<!-{}-
  Client intiates authentication.
  Here, no initial response is needed, so none is sent.
-{}->
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='CRAM-MD5'>
  <user-agent id='d4565fa7-4d72-4749-b3d3-740edbf87770'>
    <software>AwesomeXMPP</software>
    <device>Kiva's␣Phone</device>
␣␣</user-agent>
</authenticate>

<!-{}-
␣␣CRAM-MD5␣starts␣with␣a␣server␣challenge:
-{}->
<challenge␣xmlns='urn:xmpp:sasl:2'>
␣␣PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
</challenge>

<!-{}-
␣␣And␣the␣client␣responds:
-{}->
<response␣xmlns='urn:xmpp:sasl:2'>
␣␣dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw
</response>

<!-{}-
␣␣This␣completes,␣so␣the␣Server␣sends␣a␣success.
-{}->
<success␣xmlns='urn:xmpp:sasl:2'>
␣␣<authorization-identifier>tim@example.org</authorization-identifier>
</success>

<!-{}-
␣␣The␣server␣immediately␣sends␣a␣new␣set␣of␣stream␣features␣at␣this␣
    point.
␣␣There␣is␣no␣stream␣restart.
-{}->
<stream:features>
␣␣<bind␣xmlns='urn:ietf:params:xml:ns:xmpp-bind'>
␣␣␣␣<required/>
␣␣</bind>
</stream:features>

<!-{}-
␣␣The␣client␣now␣continues␣with␣resource␣binding.
-{}->
```

5.3 Advanced Usage

This moves into the deeply hypothetical. A binding extension is posited, alongside an unrealistic 2FA mechanism which somehow mutually authenticates because why not.

Listing 15: BLURDYBLOOP and UNREALISTIC-2FA

```
<!-{}-
  Client sends stream header
-{}->
<stream:stream
  from='alice@example.org'
  to='example.org'
  version='1.0'
  xml:lang='en'
  xmlns='jabber:client'
  xmlns:stream='http://etherx.jabber.org/streams'>

<!-{}-
  Server sends stream features.
  Note that whilst megabind is advertised, the unrealistic 2FA
      mechanism is not.
-{}->
<stream:features>
  <authentication xmlns='urn:xmpp:sasl:2'>
    <mechanism>BLURDYBLOOP</mechanism>
    <inline>
      <megabind xmlns='urn:example:megabind'/>
    </inline>
  </authentication>
</stream:features>

<!-{}-
  Client intiates authentication.
-{}->
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='BLURDYBLOOP'>
  <initial-response>
    SW5pdGlhbCBSZXNwb25zZQ==
  </initial-response>
  <user-agent id='d4565fa7-4d72-4749-b3d3-740edbf87770'>
    <software>AwesomeXMPP</software>
    <device>Kiva's␣Phone</device>
␣␣</user-agent>
␣␣<megabind␣xmlns='urn:example:megabind'>
␣␣␣␣<resource>this-one-please</resource>
␣␣</megabind>
</authenticate>

<!-{}-
```

```
␣␣Maybe␣BLURDYBLOOP␣does␣this:
-{}->
<challenge␣xmlns='urn:xmpp:sasl:2'>
␣␣PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
</challenge>

<!-{}-
␣␣And␣the␣client␣responds:
-{}->
<response␣xmlns='urn:xmpp:sasl:2'>
␣␣dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw
</response>

<!-{}-
␣␣This␣completes,␣so␣the␣Server␣sends␣a␣continue,␣with␣the
␣␣success␣data␣(probably␣mutual␣auth)␣from␣BLURDYBLOOP.
␣␣The␣next␣task␣required␣is␣the␣unrealistic-2FA.
␣␣Note␣that␣the␣session␣is␣-␣probably␣-␣bound␣at␣this␣point,␣but
␣␣the␣authorization-identifier␣is␣not␣passed␣to␣the␣client␣until
␣␣the␣full␣authentication␣sequence␣is␣completed.
-{}->
<continue␣xmlns='urn:xmpp:sasl:2'>
␣␣<additional-data>
␣␣␣␣QWRkaXRpb25hbCBEYXRh
␣␣</additional-data>
␣␣<tasks>
␣␣␣␣<task>UNREALISTIC-2FA</task>
␣␣</tasks>
</continue>

<!-{}-
␣␣The␣client␣supports␣the␣unrealistic-2FA,␣so␣can␣move␣onto␣the␣next␣
    task:
-{}->
<next␣xmlns='urn:xmpp:sasl:2'␣task='UNREALISTIC-2FA'>
␣␣<parameters␣xmlns='urn:example:unrealistic2fa'>
␣␣␣␣VW5yZWFsaXN0aWMgMkZBIElS
␣␣</parameters>
</next>

<!-{}-
␣␣This␣2FA␣process␣is␣both␣unrealistic␣and␣also␣uses␣multiple␣round-
    trips.
-{}->
<task-data␣xmlns='urn:xmpp:sasl:2'>
␣␣<question␣xmlns='urn:example:unrealistic2fa'>
␣␣␣␣PDE4OTYuNjk3MTcwOTUyQHBvc3RvZmZpY2UucmVzdG9uLm1jaS5uZXQ+
␣␣</question>
</task-data>
```

```
<!-{}-
␣␣The␣client␣responds␣here.
-{}->
<task-data␣xmlns='urn:xmpp:sasl:2'>
␣␣<response␣xmlns='urn:example:unrealistic2fa'>
␣␣␣␣dGltIGI5MTNhNjAyYzdlZGE3YTQ5NWI0ZTZlNzMzNGQzODkw
␣␣</response>
</task-data>

<!-{}-
␣␣Finally,␣the␣server␣sends␣a␣success.
␣␣Here,␣we␣have␣an␣authzid␣which␣is␣a␣full␣jid,␣since␣the␣session␣is␣
    now␣bound.
␣␣The␣result␣element␣here␣contains␣the␣mutual␣authentication␣data␣for␣
    the
␣␣unrealistic␣2FA,␣and␣doesn't refer back to the BLURDYBLOOP exchange.
-{}->
<success xmlns='urn:xmpp:sasl:2'>
  <result xmlns='urn:example:unrealistic2fa'>
    VW5yZWFsaXN0aWMgMkZBIG11dHVhbCBhdXRoIGRhdGE=
  </result>
  <authorization-identifier>
    alice@example.org/this-one-please
  </authorization-identifier>
</success>
```

Although the unrealistic 2FA here uses 2 round-trips (real ones will probably use one), the embedding of resource binding as shown here means that a second RTT is saved by SASL2, and there's no net change. A more realistic example would see RTTs saved, and additional negotiations could be added to further reduce RTTs.

6 IANA Considerations

This XEP requires no interaction with the Internet Assigned Numbers Authority (IANA) [11].

7 XMPP Registrar Considerations

7.1 Protocol Namespaces

This specification defines the following XML namespace:

[11]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/>.

- urn:xmpp:sasl:2

7.2 Stream Features

The XMPP Registrar includes "urn:xmpp:sasl:2" in its registry of stream features (see <https://xmpp.org/registrar/stream-features.html>).

```
<feature>
  <ns>urn:xmpp:sasl:2</ns>
  <name>sasl2</name>
  <element>authentication</element>
  <desc>Indicate support for Extensible SASL Profile</desc>
  <doc>&xep0388;</doc>
</feature>
```

8 XML Schema

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

<xs:schema xmlns:xs='http://www.w3.org/2001/XMLSchema'
           targetNamespace="urn:xmpp:sasl:2"
           xmlns="urn:xmpp:sasl:2"
           elementFormDefault="qualified">

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0388: http://xmpp.org/extensions/xep-0388.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name="authentication">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="mechanism" type="SaslMechName" minOccurs="1"
            maxOccurs="unbounded"/>
        <xs:element name="inline" minOccurs="0" maxOccurs="1">
          <xs:complexType>
            <xs:sequence>
              <xs:any namespace='##other' maxOccurs='unbounded'
                  processContents='lax'/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
```

```
</xs:element>

<xs:element name="abort">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="text" type="xs:string" minOccurs="0"
          maxOccurs="1"/>
      <xs:any namespace='##other' minOccurs='0' maxOccurs='unbounded
          ' processContents='lax'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="authenticate">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="initial-response" type="SaslData" minOccurs=
          "0" maxOccurs="1"/>
      <xs:element name="user-agent">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="software" type="xs:string" minOccurs="
                0" maxOccurs="1"/>
            <xs:element name="device" type="xs:string" minOccurs="0"
                maxOccurs="1"/>
          </xs:sequence>
          <xs:attribute name="id" type="Uuid"/>
        </xs:complexType>
      </xs:element>
      <xs:any namespace='##other' minOccurs='0' maxOccurs='unbounded
          ' processContents='lax'/>
    </xs:sequence>
    <xs:attribute name="mechanism" type="SaslMechName"/>
  </xs:complexType>
</xs:element>

<xs:element name="challenge" type="SaslData"/>

<xs:element name="response" type="SaslData"/>

<xs:element name="success">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace='##other' minOccurs='0' maxOccurs='unbounded
          ' processContents='lax'/>
      <xs:element name="additional-data" type="SaslData" minOccurs="
          0"/>
      <xs:element name="authorization-identifier" type="Jid"/>
      <xs:any namespace='##other' minOccurs='0' maxOccurs='unbounded
          ' processContents='lax'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="failure">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="text" type="xs:string" minOccurs="0"/>
      <xs:any namespace='##other' minOccurs='0' maxOccurs='unbounded
          ' processContents='lax'/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="continue">
  <xs:complexType>
    <xs:sequence>
      <xs:element name="additional-data" type="SaslData"/>
      <xs:element name="tasks">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="task" type="SaslMechName" maxOccurs='
                unbounded'/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
      <xs:element name="text" type="xs:string" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>

<xs:element name="next">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace='##other' minOccurs='0' maxOccurs='unbounded
          ' processContents='lax'/>
    </xs:sequence>
    <xs:attribute name="task" type="SaslMechName"/>
  </xs:complexType>
</xs:element>

<xs:element name="task-data">
  <xs:complexType>
    <xs:sequence>
      <xs:any namespace='##other' minOccurs='0' maxOccurs='unbounded
          ' processContents='lax'/>
    </xs:sequence>
    </xs:complexType>
  </xs:element>

  <xs:simpleType name="Jid">
    <xs:restriction base="xs:string">
      <xs:maxLength value="3071"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="Uuid">
    <xs:restriction base="xs:string">
      <xs:pattern value="([0-9]|[a-f]|[A-F]){8}-([0-9]|[a-f]|[A-F])
          {4}-([0-9]|[a-f]|[A-F]){4}-([0-9]|[a-f]|[A-F]){4}-([0-9]|[a-
          f]|[A-F]){12}"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="SaslMechName">
    <xs:restriction base="xs:string">
      <xs:minLength value="1"/>
      <xs:maxLength value="20"/>
    </xs:restriction>
  </xs:simpleType>

  <xs:simpleType name="SaslData">
    <xs:restriction base="xs:base64Binary"/>
  </xs:simpleType>
</xs:schema>
```

9 Acknowledgements

The authors wish to share any credit with many members of the community, including Lance Stout, Ralph Meijer, Phil Roberts and Florian Schmaus.