XEP-0484: Fast Authentication Streamlining Tokens

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

2024-06-30
Version 0.2.0

Status	Type	Short Name
Experimental	Standards Track	fast

This specification defines a token-based method to streamline authentication in XMPP, allowing fully authenticated stream establishment within a single round-trip.

Legal

Copyright

Permissions

Warranty

Liability

Conformance

Contents

1 Introduction

This specification defines a protocol that allows a client that successfully authenticates using one mechanism (e.g. a password-based authentication mechanism) to exchange it for a server-generated token. Such tokens can be used on subsequent connections to quickly and efficiently authenticate to the server. They also exclude the client from interactive authentication steps, such as multi-factor authentication.
This allows clients, especially those operating in shared or less secure environments (such as web browsers), to avoid storing a password locally at all. It also enables a user to selectively revoke a client's access to their account.

1.1 Why tokens?

XMPP streams are most commonly authenticated using passwords today. Unfortunately, passwords may not be unique to the service that the user is authenticating to, may be optimized for memorability rather than security, and may contain sensitive information. Therefore, secure password authentication mechanisms (such as SCRAM and OPAQUE) necessarily involve multiple round-trips and more resource-intensive cryptography to protect the password during authentication and at rest.
Using server-issued secret tokens can improve security in many ways - such tokens can be longer, more random (unguessable) and can be rotated much more frequently than passwords. They are also useless outside the scope of the service that they were issued by and for, and easily invalidated, reducing consequences of accidental or malicious exposure.
Without the same weaknesses as passwords, it is appropriate to use simpler and faster authentication mechanisms when authenticating using tokens. That is how this protocol reduces authentication overhead while maintaining an equivalent (or higher) level of account security.

2 Requirements

- Authenticate security without introducing any extra round-trips.
- Integrate with Extensible SASL Profile (XEP-0388) [1].
- Support rotation and revocation of tokens.
- Allow channel binding for clients that can support it, while resisting downgrade attacks.
- Safe to use in TLS 1.3 0-RTT ("early data") extensions.

[1] XEP-0388: Extensible SASL Profile <https://xmpp.org/extensions/xep-0388.html>.

2.1 Differences from XEP-0397 Instant Stream Resumption

A XEP with very similar goals already exists, Instant Stream Resumption (XEP-0397) [2]. While inspired by several aspects of that protocol, FAST has a number of differences:

- This protocol does not link tokens to the lifetime of a XEP-0198 session. In fact it does not depend on XEP-0198 at all, though that can be negotiated alongside using the usual SASL2 methods for that.
- In particular, the above means that session establishment can still be accomplished in a single round-trip even if a XEP-0198 session has expired.
- Channel binding is not required, thus making this protocol suitable for clients that cannot support it, such as web clients.
- Token rotation is resilient over unreliable links.

3 Use Cases

3.1 Server advertises support for FAST

Servers that support FAST MUST advertise this as a SASL2 inline feature. The <fast/> feature element is qualified by the 'urn:xmpp:fast:0' namespace.
The <fast/> element MUST contain one or more compatible authentication mechanisms. These mechanisms MUST support authenticating with a token (instead of a password) and MUST result in success or failure within a single round-trip. There SHOULD be at least one mechanism capable of channel binding, and there SHOULD be at least one mechanism without channel binding. A set of compatible mechanisms can be found in draft-schmaus-sasl-ht-09[3].
If the server allows the client to include authentication data in a TLS 0-RTT extension payload, it MUST indicate this with a 'tls-0rtt' attribute on the element set to '1' or 'true'.

Listing 1: Server advertises support for FAST

```
<stream:features>
  <authentication xmlns='urn:xmpp:sasl:2'>
    <mechanism>SCRAM-SHA-1</mechanism>
    <mechanism>SCRAM-SHA-1-PLUS</mechanism>
    <inline>
      <fast xmlns='urn:xmpp:fast:0' tls-0rtt='true'>
        <mechanism>HT-SHA-256-ENDP</mechanism>
        <mechanism>HT-SHA-256-EXPR</mechanism>
        <mechanism>HT-SHA-256-NONE</mechanism>
      </fast>
```

[2] XEP-0397: Instant Stream Resumption <https://xmpp.org/extensions/xep-0397.html>.
[3] draft-schmaus-sasl-ht-09: The Hashed Token SASL Mechanism <https://datatracker.ietf.org/doc/draft-schmaus-kitten-sasl-ht/09/>.

```
    </inline>
  </authentication>
</stream:features>
```

3.2 Client performs initial authentication

Initially, the client won't have any FAST token to authenticate with. To obtain a token, it MUST first authenticate using another method, e.g. using a password.
To request a FAST token, a client MUST include a <request-token/> element qualified by the 'urn:xmpp:fast:0' namespace. The element MUST contain a 'mechanism' attribute, the value of which MUST be one of the FAST mechanisms advertised by the server.
In the following example, the client authenticates with SCRAM-SHA-1-PLUS using a password, but requests a token for fast reauthentication in the future, using the HT-SHA-256-ENDP mechanism.

```
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='SCRAM-SHA-1-PLUS'>
  <initial-response>[base64 encoded SASL data]</initial-response>
  <bind xmlns='urn:xmpp:bind:0'>
    <tag>AwesomeXMPP</tag>
  </bind>
  <request-token xmlns='urn:xmpp:fast:0' mechanism='HT-SHA-256-ENDP'/>
</authenticate>
```

3.3 Server provides token to client

Upon receiving a token request and successfully authenticating the client, the server generates a new unique token, valid for the requested mechanism, and includes it in the SASL2 <success/> response in a <token/> element qualified by the 'urn:xmpp:fast:0' namespace.
The server MUST NOT provide a token unless the client has been successfully and fully authenticated, including any necessary post-authentication tasks (such as multi-factor authentication).
The <token/> element MUST possess the following attributes:

'token' The secret token to be used for authentication.

'expiry' The timestamp at which the token will expire, in the DateTime profile defined by XMPP Date and Time Profiles (XEP-0082) XEP-0082: XMPP Date and Time Profiles <https://xmpp.org/extensions/xep-0082.html>..

Listing 2: Server provides a new token to the client

```
<success xmlns='urn:xmpp:sasl:2'>
  <authorization-identity>user@example.com/AwesomeXMPP.4232f4d4</
    authorization-identity>
```

```
  <bound xmlns='urn:xmpp:bind:0'>
    <metadata xmlns='urn:xmpp:mam:2'>
      <start id='YWxwaGEg' timestamp='2008-08-22T21:09:04Z' />
      <end id='b21lZ2Eg' timestamp='2020-04-20T14:34:21Z' />
    </metadata>
  </bound>
  <token xmlns='urn:xmpp:fast:0'
         expiry='2020-03-12T14:36:15Z'
         token='WXZzciBwYmFmdmZnZiBqdmd1IGp2eXFhcmZm' />
</success>
```

3.4 Client authenticates using FAST

The client authenticates normally using SASL2, using the FAST SASL mechanism it previously selected, and the token provided by the server. To indicate that it is providing a token, the client MUST include a <fast/> element qualified by the 'urn:xmpp:fast:0' namespace, within its SASL2 authentication request.
If the server indicated support for TLS 0-RTT data, the client MAY send its authentication request within the TLS 0-RTT payload of its handshake. If it does this, it MUST also include a 'count' attribute on the <fast/> element. The value of this attribute MUST be a positive integer, which is incremented by the client on every authentication attempt with this token (it SHOULD be reset to zero when the token changes).
Servers MUST reject any authentication requests received via TLS 0-RTT payloads that do not include a 'count' attribute, or where the count is less than or equal to a count that has already been processed for this token. This protects against replay attacks that 0-RTT is susceptible to.
Servers MUST bind tokens to the mechanism selected by the client in its original request, and reject attempts to use them with other mechanisms. For example, if the client selected a mechanism capable of channel binding, an attempt to use a mechanism without channel binding MUST fail even if the token would otherwise be accepted by that mechanism.

Listing 3: Client authenticates successfully using a FAST token

```
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='HT-SHA-256-ENDP'>
  <initial-response>[base64 encoded SASL data]</initial-response>
  <bind xmlns='urn:xmpp:bind:0'>
    <tag>AwesomeXMPP</tag>
  </bind>
  <fast xmlns='urn:xmpp:fast:0' count='123' />
</authenticate>

<success xmlns='urn:xmpp:sasl:2'>
  <authorization-identity>user@example.com/AwesomeXMPP.4232f4d4</
     authorization-identity>
  <bound xmlns='urn:xmpp:bind:0'>
    <metadata xmlns='urn:xmpp:mam:2'>
```

```
      <start id='YWxwaGEg' timestamp='2008-08-22T21:09:04Z' />
      <end id='b21lZ2Eg' timestamp='2020-04-20T14:34:21Z' />
    </metadata>
  </bound>
</success>
```

3.5 Server initiates token rotation

If the authentication succeeded, but the token is due for rotation (e.g. it is close to expiry), the server will generate a new token and provide it to the client in the <success/> response (even if the client did not explicitly request a token):

Listing 4: Server provides an updated token to the client

```
<success xmlns='urn:xmpp:sasl:2'>
  <authorization-identity>user@example.com/AwesomeXMPP.4232f4d4</
     authorization-identity>
  <bound xmlns='urn:xmpp:bind:0'>
    <metadata xmlns='urn:xmpp:mam:2'>
      <start id='YWxwaGEg' timestamp='2008-08-22T21:09:04Z' />
      <end id='b21lZ2Eg' timestamp='2020-04-20T14:34:21Z' />
    </metadata>
  </bound>
  <token xmlns='urn:xmpp:fast:0'
         expiry='2020-03-31T14:36:15Z'
         token='R3VyIHpiZmcgbnl2aXIgdmYgZ3VyIGp2eXFyZmcu' />
</success>
```

When the server provides a new token to the client in this way, it MUST NOT invalidate the existing token until the new token is actually used by the client. This ensures that if the client gets disconnected before receiving the newer token from the server, it can still successfully authenticate on its next connection attempt.
Upon successful use of any token, the server MUST invalidate all tokens issued to the same client with an earlier expiry than the current token (even if those tokens have not yet reached their expiry time).
Additionally, upon providing a new token to the client, the server SHOULD invalidate any tokens previously generated that have not been used.

3.6 Client requests token invalidation

A client can choose to invalidate a token before its expiry.
For example, a client might implement a ”log out” mechanism for people sharing a web browser or system. Explicitly invalidating the token with the server ensures that even if an unauthorized user managed to recover the token from the system, it would be useless.
To invalidate the token, the client MUST successfully authenticate using the token as normal,

but include an 'invalidate' attribute on the <fast/> element with a value of '1' or 'true'.
Upon successful authentication with the 'invalidate' attribute set, the server MUST immediately invalidate the token and prevent its use for future authentication attempts. The server MUST NOT include a new token in the response (even if the token was due for rotation), unless the client also included a FAST <request-token/> element in its authentication request. The client MAY close the stream after the server acknowledges successful authentication, or it MAY proceed with the session as normal.

Listing 5: Client requests token invalidation

```
<authenticate xmlns='urn:xmpp:sasl:2' mechanism='HT-SHA-256-ENDP'>
    <initial-response>[base64 encoded SASL data]</initial-response>
    <fast xmlns='urn:xmpp:fast:0' count='123' invalidate='true'/>
</authenticate>
```

4 Business Rules

4.1 Client responsibilities

- Tokens are issued on a per-client basis. Clients MUST treat tokens as sensitive information, equivalent to passwords. For example, they should not be exposed in user interfaces or included in backups and other data exports.
- Clients wishing to use FAST authentication MUST provide the authenticating JID in the secure stream's 'from' attribute. They MUST also provide the a SASL2 <user-agent> element with an 'id' attribute (both of these values are discussed in more detail in XEP-0388).
- If a client attempts authentication using a token, but the server returns a SASL <failure/>, the client SHOULD discard the token and automatically fall back to alternative authentication mechanisms.

4.2 Server responsibilities

- Tokens should be issued with a reasonable lifetime, reflective of a deployment's policy on inactive devices. Shorter lifetimes require more frequent rotation and increase the chances that a device will get "logged out" if it is offline during a token expiry. Longer lifetimes put tokens at greater risk of exposure in the event a device or its data becomes lost, stolen or compromised.
- Servers MUST NOT require interactive authentication steps (such as multi-factor authentication) when authenticating via a FAST token. If the server no longer trusts a token, it MUST instead fail the authentication (returning the SASL 'credentials-expired' error condition), and then allow the client to authenticate using other mechanisms (e.g. password based).

5 Implementation Notes

5.1 Server-side handling of multiple active tokens

As noted in the section on token rotation, this specification requires a server to sometimes accept multiple tokens for the same client. This can be achieved with the following algorithm:

- For every client using FAST, have two token slots - 'current' and 'new'.
- Whenever generating a new token, always place it into the 'new' slot.
- During authentication, first check against the token in the 'new' slot (if any). If successful, move the token from the 'new' slot to the 'current' slot (overwrite any existing token in that slot).
- If the client's provided token does not match the token in the 'new' slot, or if the 'new' slot is empty, compare against the token in the 'current' slot (if any).

This method ensures servers do not need to check against an unbounded number of active tokens, while still allowing safe rollover on unreliable connections. It also ensures that tokens are invalidated as soon as later tokens are used by the client.
Note that anywhere in this flow where the server verifies the client's token against a stored token, it needs to check not just the token itself, but also that the token has not expired, that the correct mechanism was used, and the replay counter (if applicable). Token hash comparison itself MUST be performed using constant-time comparison functions, as already available in most environments and cryptography libraries.

6 Security Considerations

FAST authentication MUST only be performed over a secure connection (e.g. using TLS with verified certificates). Due to the lack of a challenge/response step, it is generally true that any attacker able to passively observe the authentication exchange can replay the authentication and gain access to the account. Channel binding mechanisms mitigate certain attacks, and MUST be preferred by the client. However they do not mitigate all attacks, and are not available in all environments.
When the SASL payload is included in the TLS 0-RTT payload and combined with the single round-trip property of FAST SASL mechanisms, an attacker may be able to replay the same authentication multiple times, including the negotiation of features requested by the client (resumption and/or resource binding, for example). Such feature negotiations may have side-effects, such as (but not necessarily limited to) the disruption of established sessions. The per-token counter described in this document mitigates this issue.
Mechanisms that communicate using hashes (including HMACs) MUST be compared by the server using constant-time comparison functions, to prevent leaking secrets via timing attacks.

7 IANA Considerations

None.

8 XMPP Registrar Considerations

This specification defines the following XML namespace:

- urn:xmpp:fast:0

9 Acknowledgements

Many thanks to Daniel Gultsch and Thilo Molitor for their input, support, and implementations. Thanks also to Florian Schmaus for prior work on Instant Stream Resumption and the HT family of SASL mechanisms, which inspired and influenced this specification.

10 XML Schema

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

<xs:schema
    xmlns:xs='http://www.w3.org/2001/XMLSchema'
    xmlns='urn:xmpp:fast:0'
    targetNamespace='urn:xmpp:fast:0'
    elementFormDefault='qualified'>

  <xs:annotation>
    <xs:documentation>
      The protocol documented by this schema is defined in
      XEP-0484: https://xmpp.org/extensions/xep-0484.html
    </xs:documentation>
  </xs:annotation>

  <xs:element name='fast'>
    <xs:complexType>
      <xs:sequence>
        <xs:element ref='mechanism' minOccurs='0' maxOccurs='unbounded
            '/>
      </xs:sequence>
      <xs:attribute name='tls-0rtt' type='xs:boolean' use='optional'/>
      <xs:attribute name='count' type='xs:int' use='optional'/>
      <xs:attribute name='invalidate' type='xs:boolean' use='optional'
          />
```

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

  <xs:element name='mechanism' type='xs:string'/>

  <xs:element name='request-token'>
    <xs:complexType>
      <xs:attribute name='mechanism' type='xs:string'/>
    </xs:complexType>
  </xs:element>

  <xs:element name='token'>
    <xs:complexType>
      <xs:attribute name='expiry' type='xs:dateTime'/>
      <xs:attribute name='token' type='xs:string'/>
    </xs:complexType>
  </xs:element>

</xs:schema>
```