

Common Authentication Technology Next Generation
Internet-Draft
Intended status: Standards Track
Expires: 2 December 2026

M. Sabadello
Danube Tech GmbH
31 May 2026

The DID-CHALLENGE SASL Mechanism
draft-sabadello-did-challenge-sasl-01

Abstract

This specification defines "DID-CHALLENGE", a mechanism for the Simple Authentication and Security Layer (SASL) based on Decentralized Identifiers (DIDs). The mechanism follows a server-first challenge/response pattern in which the client authenticates by producing a cryptographic signature over a server-generated challenge, using the private key associated with its DID. Unlike password-based SASL mechanisms, no shared secret is transmitted or stored on the server; authentication is grounded entirely in asymmetric cryptography and the verifiable binding between a DID and its associated key material.

An optional extension adds support for Verifiable Credentials (VCs) and Verifiable Presentations (VPs), enabling attribute-based access control in addition to identity authentication.

Status of This Memo

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

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

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

This Internet-Draft will expire on 2 December 2026.

Copyright Notice

Copyright (c) 2026 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<https://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Revised BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Revised BSD License.

Table of Contents

1. Introduction	3
2. SASL mechanism name	4
3. Authentication	4
3.1. The Authentication Exchange	4
3.2. Authorization Identity String	5
3.3. DID Challenge	5
3.4. DID Response	6
3.5. Client Verification	7
3.6. Server Verification	7
4. SASL Exchange with DIDs	8
5. (Optional) Authentication with VCs/VPs	11
5.1. The Authentication Exchange (with VC/VP support)	11
5.2. VC-VP Challenge	11
5.3. VC-VP Response	11
5.4. Server Verification	12
6. (Optional) SASL Exchange with DIDs and VCs/VPs	13
7. Example Exchange	16
7.1. Step 1: Client NameCallback for DID	16
7.2. Step 2: Client JWKCallback for Private Key	16
7.3. Step 3: Server -> Client Challenge	17
7.4. Step 4: Client Signature	17
7.5. Step 5: Client -> Server Response	17
7.6. Step 6: Server NameCallback with DID	17
7.7. Step 7: Server Verification	18
7.8. Step 8: Server AuthorizeCallback with authorization ID	18
8. Security Considerations	18
8.1. Mechanism Strength	18
8.2. Requirement for a Confidential Channel	19
8.3. Replay Attacks	19
8.4. Man-in-the-Middle Attacks and Channel Binding	20
8.5. Server Spoofing and Mutual Authentication	20
8.6. Choosing and Trusting DID Resolvers	21
8.7. Key Revocation, Rotation, and DID Method Properties	21
8.8. Non-Repudiation	22
8.9. Authentication vs. Authorization	22
8.10. Private Key Protection	22
8.11. Security of the Optional VC/VP Extension	23

8.12. Denial of Service	23
9. Implementations	24
10. Acknowledgements	25
Author's Address	25

1. Introduction

Many Internet protocols require authentication. Common approaches include username/password schemes (as used in IMAP or XMPP), static public key authentication (as used in SSH), and federated identity protocols (as used in OpenID Connect). Each of these approaches has well-known limitations: passwords can be stolen or guessed, static public keys provide no mechanism for revocation, and federated schemes introduce a dependency on a central identity provider.

Decentralized Identifiers (<https://www.w3.org/TR/did-1.1/>) are a class of globally unique identifier designed to be created and controlled directly by their subjects, without requiring a central registration authority. A DID resolves to a DID Document - a machine-readable document that contains cryptographic key material and other metadata about the DID subject. DID Documents are anchored in a Verifiable Data Registry: a system - such as a distributed ledger, decentralized file system, or DNS zone - that provides a trustworthy, tamper-evident record of DID state. The controller of a DID can prove that control by signing data with the private key corresponding to a public key published in the DID Document, without needing permission from any third party.

The Simple Authentication and Security Layer (<https://www.rfc-editor.org/rfc/rfc4422.html>) is an extensible framework that decouples authentication mechanisms from the application protocols that use them. By defining a SASL mechanism, a new authentication approach can be made available to any SASL-enabled protocol - including IMAP, SMTP, LDAP, XMPP, and others - without modifying those protocols individually.

This specification defines "DID-CHALLENGE", a SASL mechanism that allows a client to authenticate using a DID. The SASL client takes the role of a DID controller; the SASL server takes the role of a DID Resolver and verifier. Authentication proceeds by the server issuing a challenge (a nonce, timestamp, and realm), the client signing that challenge with its DID's private key, and the server verifying the signature against the public key material retrieved from the client's DID Document. Because authentication is based on key ownership rather than a shared secret, a compromise of the server's credential store does not yield material that could be used to impersonate clients.

This specification also defines an optional extension that adds support for Verifiable Credentials (VCs) and Verifiable Presentations (VPs). VCs are signed statements issued by a trusted third party (an Issuer) about a subject - for example, attesting to a person's name, age, professional qualification, or membership in an organisation. After completing the initial DID-based authentication exchange, the server may issue one or more VC/VP Challenges requesting that the client present credentials of a specified type. The client responds with a Verifiable Presentation: a signed envelope containing the requested credentials and binding them to the authenticated DID. This enables the server to make fine-grained, attribute-based access-control decisions beyond simple identity verification.

Readers seeking to implement this mechanism should be familiar with the SASL framework (RFC4422 (<https://www.rfc-editor.org/rfc/rfc4422.html>)), the W3C DIDs v1.1 - DID Syntax (<https://www.w3.org/TR/did-1.1/#did-syntax>) specification, and the W3C DID Resolution v1.0 (<https://www.w3.org/TR/did-resolution/>) specification. Familiarity with the W3C Verifiable Credentials Data Model v2.0 (<https://www.w3.org/TR/2025/REC-vc-data-model-2.0-20250515/#types>) specification is required for implementations that use the optional VC/VP extension.

2. SASL mechanism name

The name of the DID-based SASL mechanism is "DID-CHALLENGE".

3. Authentication

This section describes the interaction between a SASL client and SASL server that use the "DID-CHALLENGE" mechanism.

3.1. The Authentication Exchange

The "DID-CHALLENGE" mechanism is a server-first mechanism: the server sends the first piece of authentication data (see Section 3.3) without waiting for any initial client message beyond the mechanism selection.

The exchange consists of the following steps:

```
C: Request authentication exchange
S: DID Challenge
C: DID Response
S: Outcome of authentication exchange
```

The mechanism is capable of transferring an authorization identity string (see Section 3.2), which the client MUST include in the DID Response (see Section 3.4).

The server is not expected to provide additional data when indicating a successful outcome. On failure, the server MUST terminate the exchange and SHOULD provide an appropriate error indication to the client in accordance with the enclosing protocol's SASL profile.

As security layers, the mechanism provides authentication and integrity protection of the authorization identity during the exchange, by means of a cryptographic signature over the server-generated challenge (see Section 3.2). It does not provide a general-purpose security layer over the application data stream after authentication completes; confidentiality and integrity of post-authentication traffic MUST be provided by the underlying transport, such as (RFC8446 (<https://www.rfc-editor.org/rfc/rfc8446.html>)).

The use of TLS is therefore strongly RECOMMENDED whenever this mechanism is employed (see Section 8.2).

3.2. Authorization Identity String

In the "DID-CHALLENGE" mechanism, the authorization identity string (<https://www.rfc-editor.org/rfc/rfc4422#section-3.4.1>) is a DID as defined by W3C DIDs v1.1 - DID Syntax (<https://www.w3.org/TR/did-1.1/#did-syntax>), and percent-encoded as defined by RFC3986 - Section 2.1 (<https://www.rfc-editor.org/rfc/rfc3986#section-2.1>).

Example authorization identity string:

```
did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D
```

3.3. DID Challenge

The DID Challenge has the following format:

```
"<" nonce "." timestamp "@" realm ">"
```

Where:

- * nonce is a server-generated random string. It MUST be unique across all challenges issued by the server. The nonce MUST be generated by a cryptographically strong (pseudo) random number generator and MUST contain at least 64 bits of entropy. The nonce MUST NOT contain the characters ".", "@", "<", ">", or SP, as these are used as delimiters in the challenge format.

- * timestamp is the number of milliseconds elapsed since the Unix epoch (1970-01-01T00:00:00Z), encoded as a decimal integer with no leading zeros. The server MUST set this field to the current time at the moment the challenge is generated.
- * realm is the SASL realm of the server. It identifies the service context to which the challenge belongs and is included in the signed material to prevent cross-service signature reuse. The realm MUST NOT contain the characters "@", "<", ">", or SP.

Example:

```
<7795631894096664932.1765144656954@java-sasl-xmpp-server>
```

In this example, the nonce is "7795631894096664932", the timestamp is "1741267200000" (2025-03-06T12:00:00Z in milliseconds), and the realm is "java-sasl-xmpp-server".

3.4. DID Response

The DID Response has the following format:

did SP signature

Where:

- * did is the client's Decentralized Identifier (DID), percent-encoded as defined in Section 3.2. This is the SASL authorization identity string supplied by the client. The DID MUST be resolvable to a DID Document that contains at least one verification method with an "authentication" verification relationship (see W3C DIDs v1.1 - Verification Relationships (<https://www.w3.org/TR/did-1.1/#verification-relationships>)).
- * signature is the base64url encoding (RFC4648 (<https://www.rfc-editor.org/rfc/rfc4648.html>)) of the raw bytes of the digital signature, without padding characters ("="). The signature MUST be computed over the entire DID Challenge string (including the enclosing angle brackets) as specified in Section 3.3.

The signing algorithm MUST correspond to the key type of the verification method in the DID document (e.g., Ed25519 for keys of type "Multikey" with a Multibase-encoded Ed25519 public key).

The two fields MUST be separated by exactly one space character. Leading and trailing whitespace in the DID Response MUST NOT be present.

Example:

```
did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D frEko8nWU-rfArpMZsMVbXpg4xCh  
aQIv_MCmIAmHD3OCWwYvL7CDOedMbezMs4pmGGuzpkRH2QX8UMa-RFToBg
```

3.5. Client Verification

Upon receiving the DID Challenge, the client MUST perform the verification steps listed below, in the order given. If any step fails, the client MUST immediately treat the exchange as an authentication failure, MUST NOT proceed to subsequent steps, and MUST terminate the authentication exchange with an appropriate error indication.

- * Parse the DID Challenge. Verify that the DID Challenge conforms to the grammar defined in Section 3.3. A challenge that does not conform MUST cause the client to abort the authentication exchange.
- * Extract the nonce, timestamp, and signature fields.
- * Verify the realm. Verify that the realm in the received DID Challenge matches the realm of the service the client intends to authenticate to. A realm mismatch MUST cause the client to abort the authentication exchange.

3.6. Server Verification

Upon receiving the DID Response, the server MUST perform the verification steps listed below, in the order given. If any step fails, the server MUST immediately treat the exchange as an authentication failure, MUST NOT proceed to subsequent steps, and MUST terminate the authentication exchange with an appropriate error indication.

- * Parse the DID Response. Verify that the DID Response conforms to the grammar defined in Section 3.4. A response that does not conform MUST cause the server to abort the authentication exchange.
- * Extract the did and signature fields.
- * Verify the nonce. Verify that the nonce embedded in the DID Challenge has not previously been accepted in a completed authentication exchange. The server MUST maintain a record of all nonces issued and accepted within the active timestamp window for this purpose. A repeated nonce MUST be treated as a replay attack and the exchange rejected.

- * Verify the timestamp. Verify that the timestamp embedded in the DID Challenge, interpreted as milliseconds since the Unix epoch, represents a time within the server's acceptance window. The RECOMMENDED acceptance window is no more than 300 seconds (5 minutes) in the past, and no more than 5 seconds in the future (to accommodate minor clock skew between client and server). Server clocks SHOULD be synchronized via NTP or an equivalent mechanism. A timestamp outside the acceptance window MUST be treated as an authentication failure.
- * Resolve the DID. Resolve the did field to a DID document using a trust validated DID resolver, in accordance with the W3C DID Resolution v1.0 (<https://www.w3.org/TR/did-resolution/>) specification. If resolution fails for any reason, or if the DID is deactivated, the server MUST treat this as an authentication failure.
- * Retrieve authentication verification methods. From the resolved DID Document, retrieve all verification methods that have an "authentication" verification relationship, in accordance with the W3C DIDs v1.1 - Verification Relationships (<https://www.w3.org/TR/did-1.1/#verification-relationships>) specification. If no such verification methods are present, the server MUST treat this as an authentication failure.
- * Verify the signature. Decode the signature field using base64url decoding without padding. Using each candidate verification method retrieved in the previous step, attempt to verify the decoded signature against the entire DID Challenge string (including the enclosing angle brackets), treated as an opaque octet string. The signing algorithm used for each attempt MUST correspond to the key type of the candidate verification method. If no verification method is able to verify the signature, the server MUST treat this as an authentication failure.

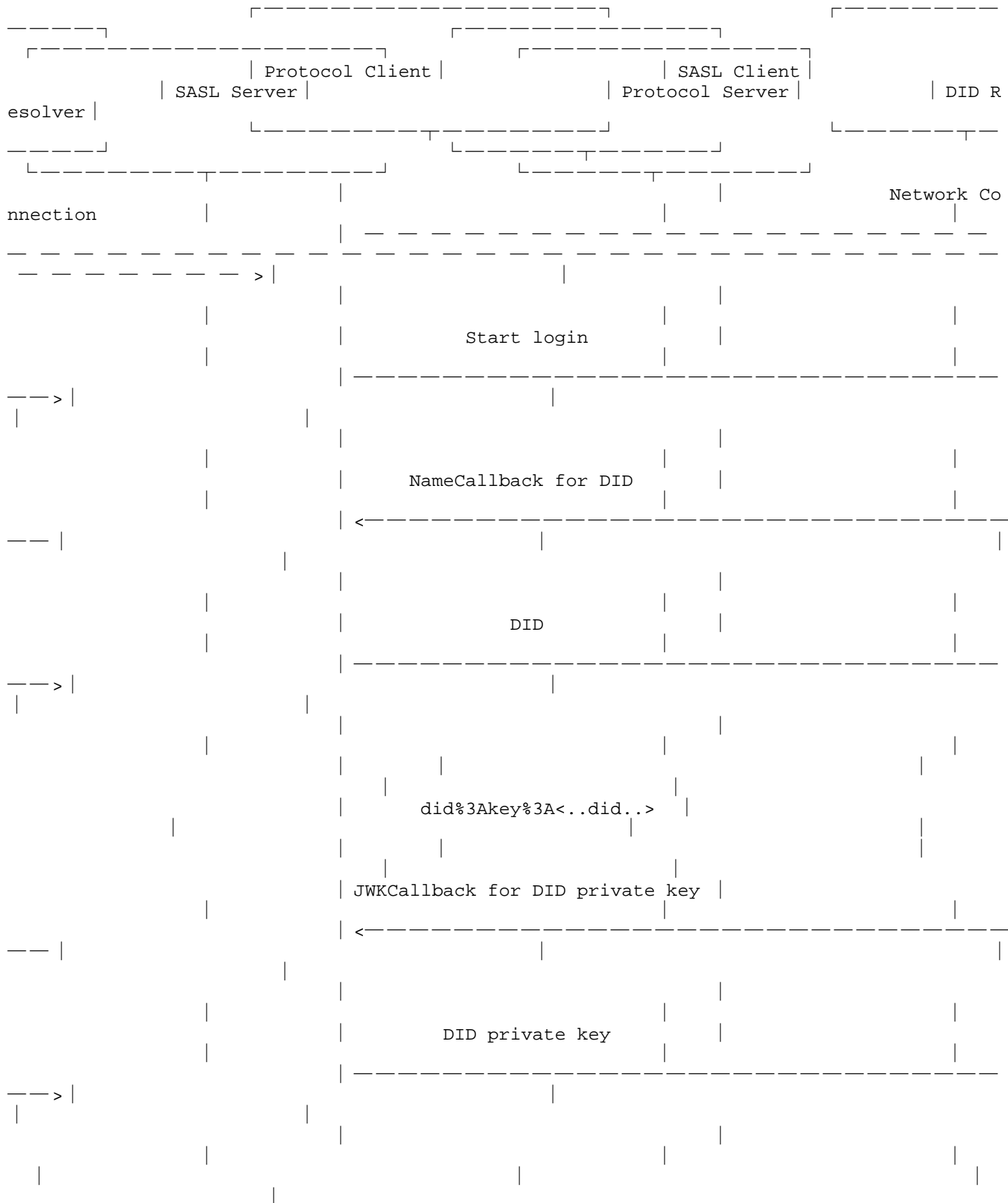
If all steps succeed, the server MUST use the authenticated DID as the authorization identity. The server MUST then invoke whatever authorization check is required by the enclosing application (e.g., the `AuthorizeCallback` in the SASL framework) before granting access.

4. SASL Exchange with DIDs

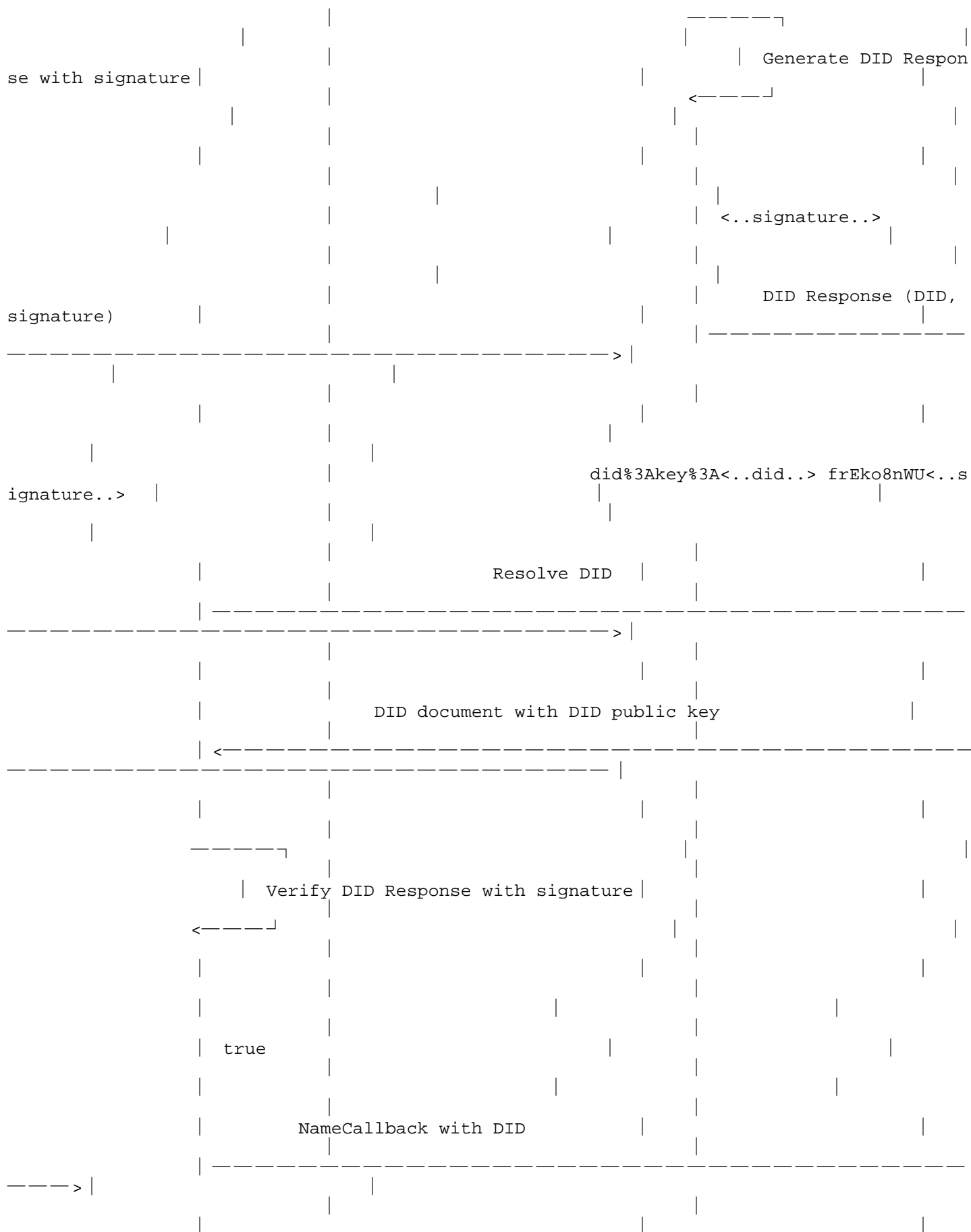
This section illustrates the detailed steps of the SASL exchange.

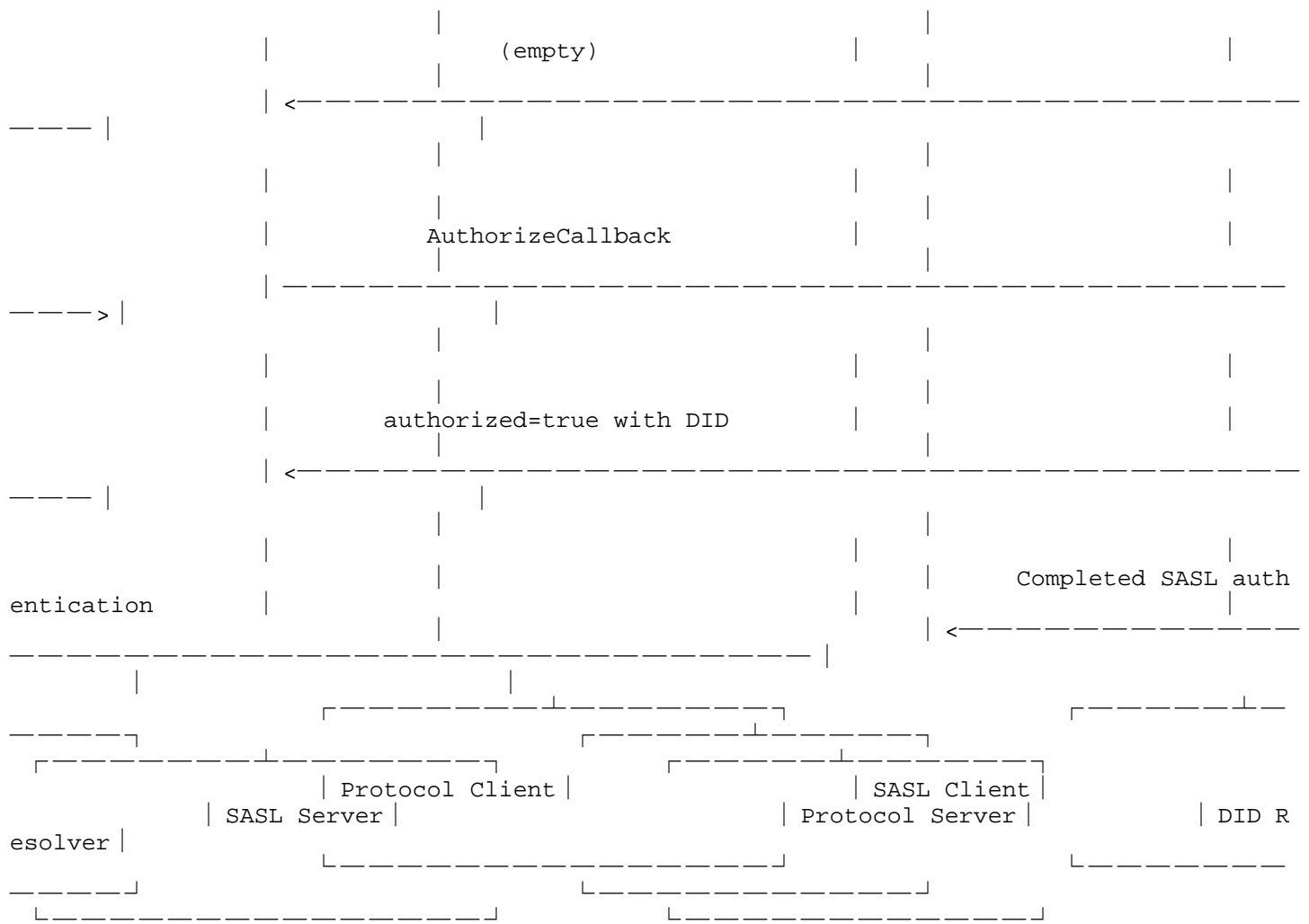
The flow includes the DID Challenge (see Section 3.3) and DID Response (see Section 3.4) steps.

NGE SASL mechanism" "The DID-CHALLENGE SASL mechanism"









5. (Optional) Authentication with VCs/VPs

This section defines an optional extension of the "DID-CHALLENGE" SASL mechanism which adds support for Verifiable Credentials (VCs) and Verifiable Presentations (VPs).

5.1. The Authentication Exchange (with VC/VP support)

The exchange consists of the following steps (expanding on Section 3):

```
C: Request authentication exchange
S: DID Challenge
C: DID Response
S: VC/VP Challenge
C: VC/VP Response
S: Outcome of authentication exchange
```

The steps VC/VP Challenge and VC/VP Response may be repeated multiple times.

5.2. VC-VP Challenge

The VC/VP Challenge follows the following format:

```
"<" nonce "." timestamp "." vc-type "@" realm ">"
```

Where:

- * For nonce, the same rules apply as in Section 3.3.
- * For timestamp, the same rules apply as in Section 3.3.
- * For realm, the same rules apply as in Section 3.3.
- * vc-type MUST be a type of a Verifiable Credential as defined in W3C Verifiable Credentials Data Model v2.0 - Types (<https://www.w3.org/TR/2025/REC-vc-data-model-2.0-20250515/#types>).

Example:

```
<7795631894096664932.1765144656954.DegreeCredential@java-sasl-xmpp-server>
```

5.3. VC-VP Response

The VC/VP Response follows the following format:

vp

Where:

- * vp MUST be a Verifiable Presentation as defined in W3C Verifiable Credentials Data Model v2.0 - Verifiable Presentations (<https://www.w3.org/TR/2025/REC-vc-data-model-2.0-20250515/#verifiable-presentations>).

Example:

```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://www.w3.org/ns/credentials/examples/v2"
  ],
  "id": "urn:uuid:3978344f-8596-4c3a-a978-8fcaba3903c5",
  "type": ["VerifiablePresentation"],
  "verifiableCredential": [{
    "id": "did:key:z6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D"
    "type": ["DegreeCredential"]
  }]
}
```

5.4. Server Verification

Upon receiving the VC/VP Response, the server MUST perform the verification steps listed below, in the order given. If any step fails, the server MUST immediately treat the exchange as an authentication failure, MUST NOT proceed to subsequent steps, and MUST terminate the authentication exchange with an appropriate error indication.

- * Parse the VC-VP Response. Verify that the VC-VP Response conforms to the grammar defined in Section 5.3. A response that does not conform MUST cause the server to abort the authentication exchange.
- * Verify the nonce and the timestamp following the same rules as in Section 3.6.
- * Verify that the "holder" property of the VP field matches the did in Section 3.3.
- * Verify that the "type" property of the VP field matches the requested vc-type field in the Section 5.3.

- ## 6. (Optional) SASL Exchange with DIDs and VCs/VPs

The flow includes the DID Challenge (see Section 3.3), DID Response (see Section 3.4), VC/VP Challenge (see Section 5.2), and VC/VP Response (see Section 5.3).

```

sequenceDiagram
    participant Net
    participant SASL_Client as SASL Client
    participant SASL_Server as SASL Server
    participant Protocol_Client as Protocol Client
    participant Protocol_Server as Protocol Server
    participant DID_Resolver as DID Resolver

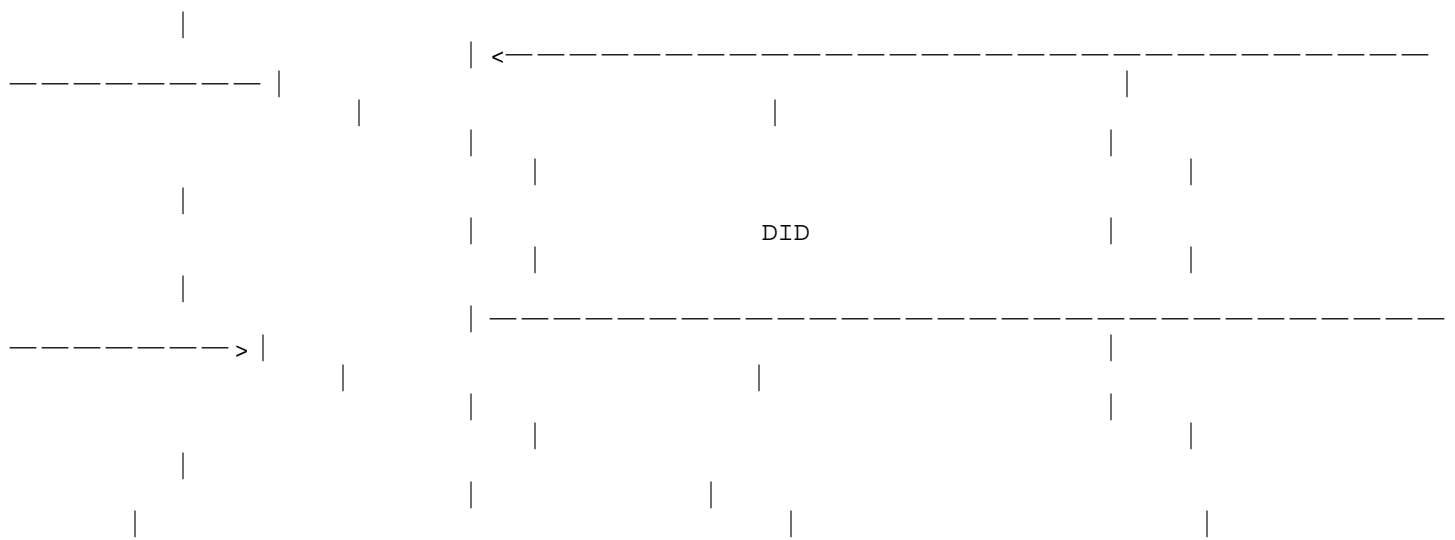
    Net->>SASL_Client: 
    SASL_Client->>SASL_Server: 
    SASL_Server->>Protocol_Client: 
    Protocol_Client->>Protocol_Server: 
    Protocol_Server->>DID_Resolver: 
    DID_Resolver->>Protocol_Client: 
    Protocol_Client->>SASL_Server: 
    SASL_Server->>SASL_Client: 
    SASL_Client->>Net: 

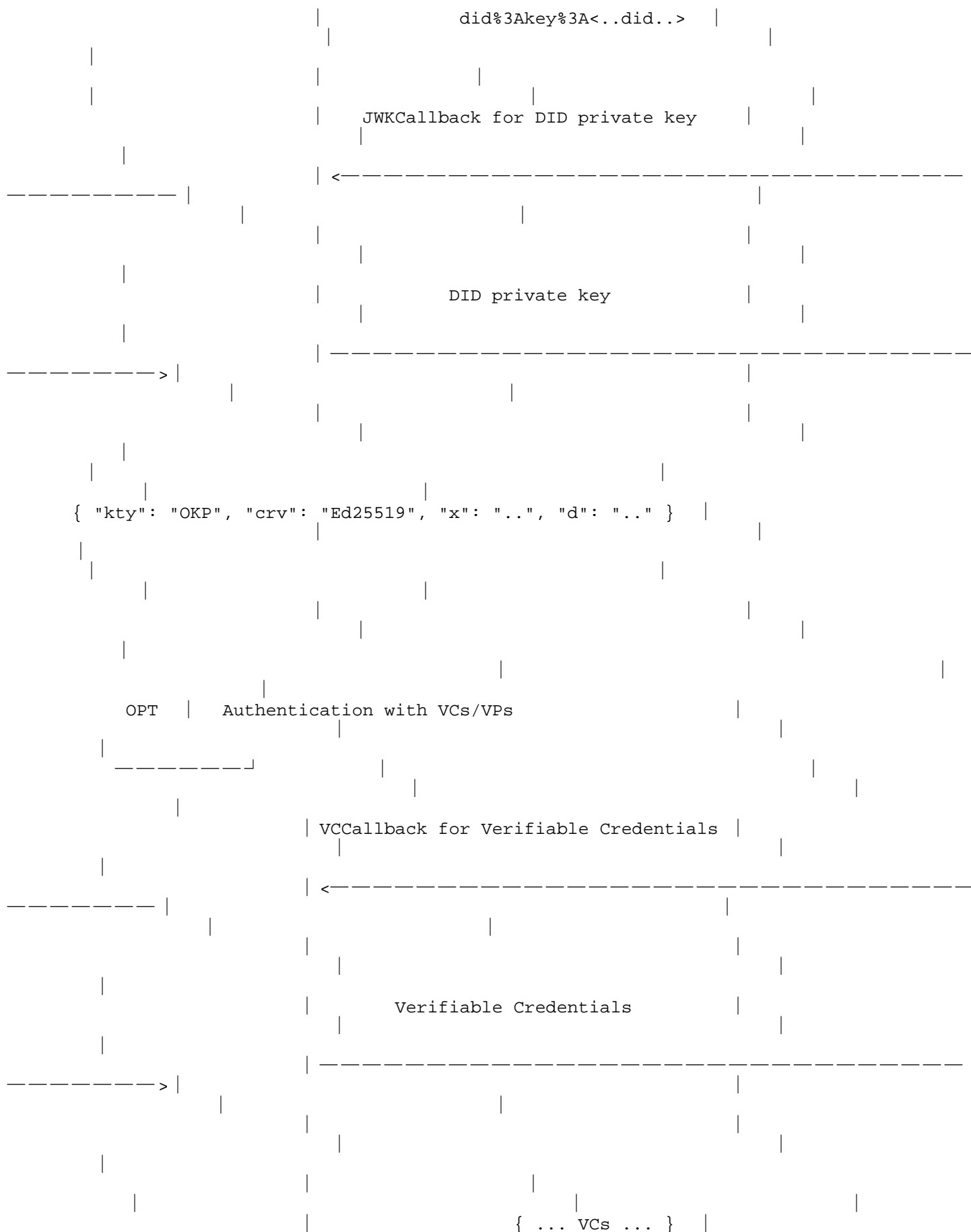
```

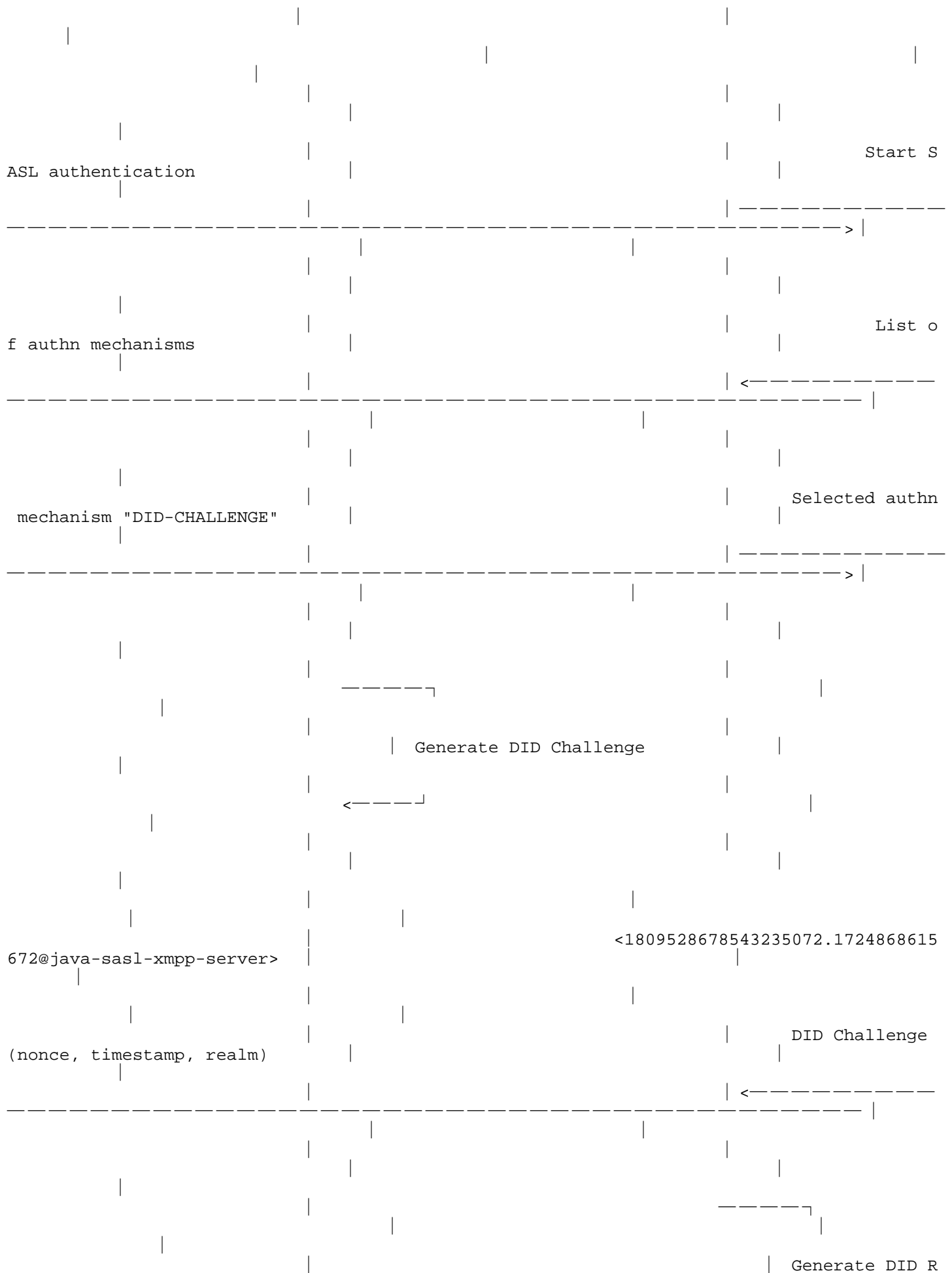
The diagram illustrates the interaction between a DID Resolver, Protocol Client, SASL Server, SASL Client, and Protocol Server. The process begins with a connection from the Net to the SASL Client, followed by a connection from the SASL Client to the SASL Server. The SASL Server then connects to the Protocol Client, which connects to the Protocol Server. The Protocol Server sends a message to the DID Resolver, which then sends a message back to the Protocol Client. The Protocol Client sends a message to the SASL Server, which then sends a message back to the SASL Client. Finally, the SASL Client sends a message to the Net.

Key events in the sequence include:

- Start login
- NameCallback for DID







esponse with signature

|

|

|

|

|

|

|

|

<-----|

|

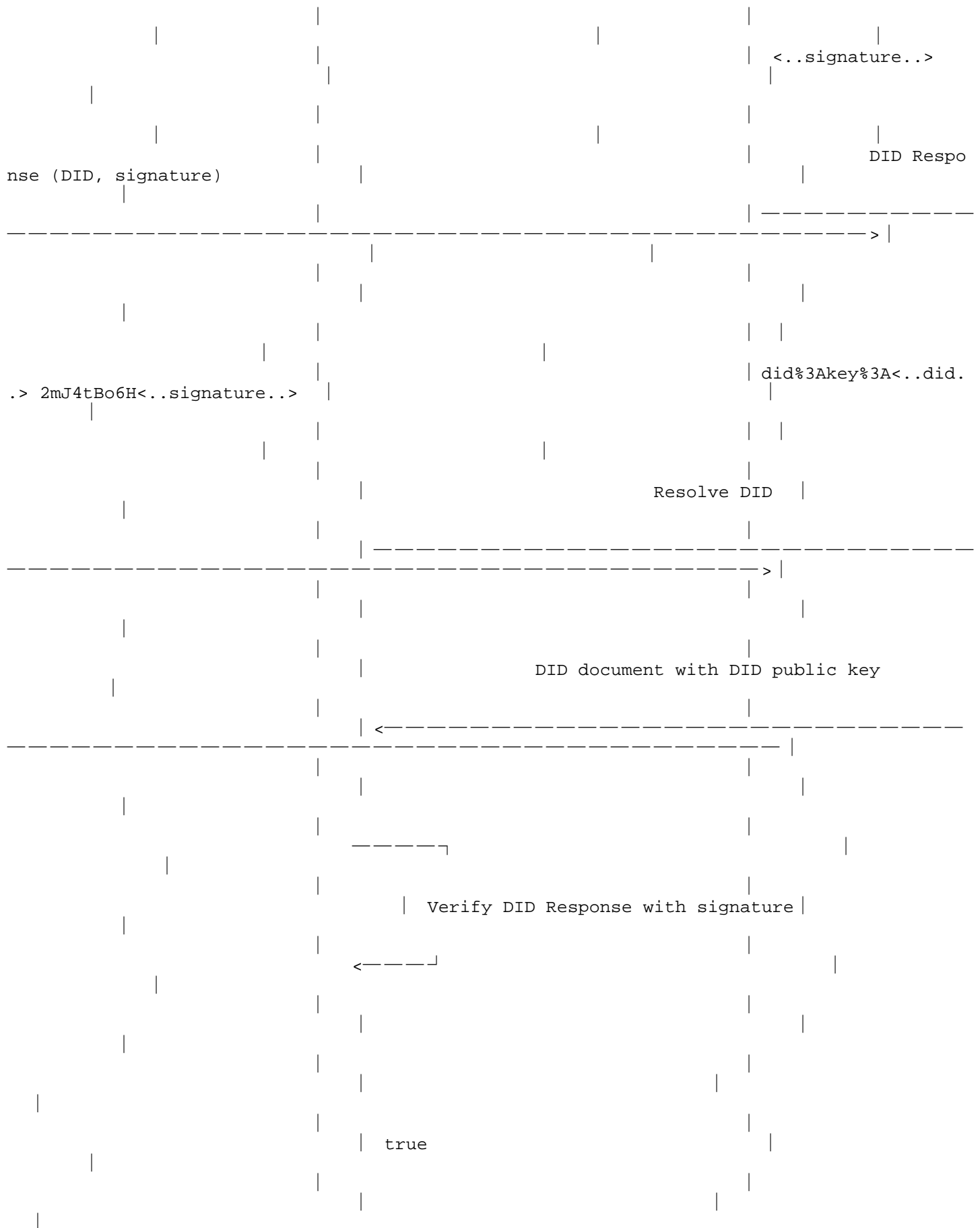
|

|

Sabadello

Expires 2 December 2026

[Page 14]



OPT Authentication with VCs/VPs

Generate VC/VP Challenge

al@java-sasl-xmpp-server>

<1809528678543235072.1724868615672.DegreeCredenti

ce, timestamp, vc.type, realm)

VC/VP Challenge (non

Response with proof

Generate VC/VP

<..vp..>

Response (VP)

VC/VP

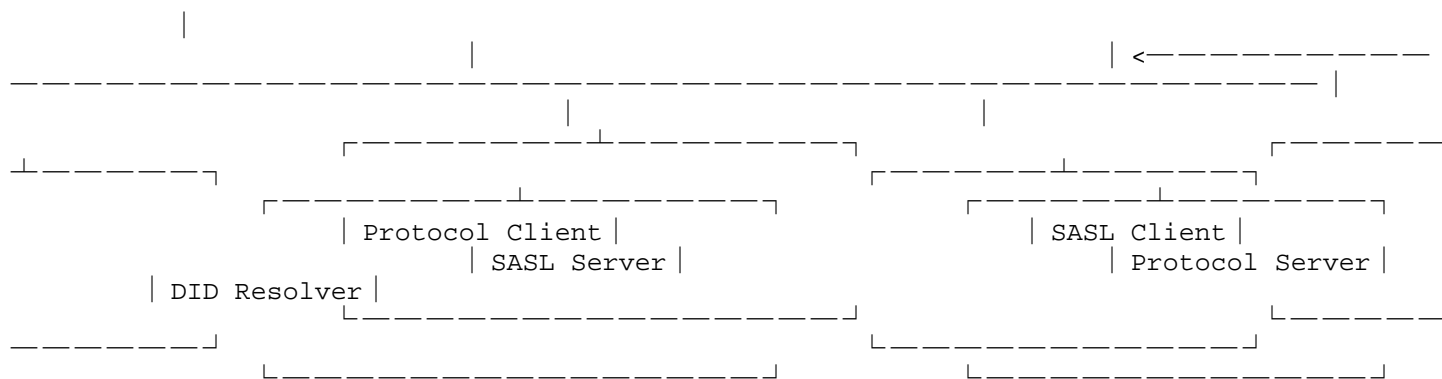
<...vp...>

Sabadello

Expires 2 December 2026

[Page 15]





7. Example Exchange

7.1. Step 1: Client NameCallback for DID

When the client is initialized, it obtains a DID to be used for authentication.

```
-- CLIENT CALLBACK: NameCallback
```

```
>C Client DID: --- defaultName: null, name: null
getName() -> did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D
C> DID: --- defaultName: null, name: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh
5kR3dH51D
```

7.2. Step 2: Client JWKCallback for Private Key

When the client is initialized, it obtains a private key that will be used for signing challenges.


```
-- CLIENT CALLBACK: JWKCallback
```

```
>C Client private key: --- defaultText: (JWK), text: null
getTextInputJWK() -> {
  "kid": "did:key:z6MkfePUhxLV6cM54cgZ4bGmnEdTNm3Wdf4arwh5kR3dH51D#z6MkfePUhxLV6cM54cgZ4bGmnEdTNm3Wdf4arwh5kR3dH51D",
  "kty": "OKP",
  "crv": "Ed25519",
  "x": "EbV6-hVmDiD3DKTUgsf2SjjnO7t0ttwMhStQ5JyCFhw",
  "d": "vGjHIZzZxS3R4mo-V0I_S72ULXDqa2INqkAtuvqJUN8"
}
C> Private key: --- defaultText: (JWK), text: {
  "kid": "did:key:z6MkfePUhxLV6cM54cgZ4bGmnEdTNm3Wdf4arwh5kR3dH51D#z6MkfePUhxLV6cM54cgZ4bGmnEdTNm3Wdf4arwh5kR3dH51D",
  "kty": "OKP",
  "crv": "Ed25519",
  "x": "EbV6-hVmDiD3DKTUgsf2SjjnO7t0ttwMhStQ5JyCFhw",
  "d": "vGjHIZzZxS3R4mo-V0I_S72ULXDqa2INqkAtuvqJUN8"
}
```

7.3. Step 3: Server -> Client Challenge

The server initiates the authentication flow by generating and sending a challenge. The challenge contains a nonce, timestamp, and realm.

```
-- SERVER -> CLIENT: Challenge
<4513455346757278126.1757192932938@java-sasl-xmpp-server>
```

7.4. Step 4: Client Signature

The client signs the challenge using the DID's private key.

```
-- CLIENT
Created signature for challenge <4513455346757278126.1757192932938@java-sasl-xmpp-server>
: frEko8nWU-rfArpMZsMVbXpg4xChaQIv_MCmIAmHD3OCWwYvL7CDOedMbezMs4pmGGuzpkRH2QX8UMa-RFTToBg
```

7.5. Step 5: Client -> Server Response

The client response to the server with the DID and the signed challenge.

```
-- CLIENT -> SERVER: Response
did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3Wdf4arwh5kR3dH51D frEko8nWU-rfArpMZsMVbXpg4xChaQIv_MCmIAmHD3OCWwYvL7CDOedMbezMs4pmGGuzpkRH2QX8UMa-RFTToBg
```

7.6. Step 6: Server NameCallback with DID

The server obtains the DID from the client's response.

-- SERVER CALLBACK: NameCallback

```
>S DID: --- defaultName: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D, n
ame: null
checkName(did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D) --> did%3Akey%3Az
6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D
S> DID: --- defaultName: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D, n
ame: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D
```

7.7. Step 7: Server Verification

The server verifies the signature in the client's response by resolving the client's DID to a DID document, which contains public keys need for the verification.

-- SERVER

```
Verified signature frEko8nWU-rfArpMZsMVbXpg4xChaQIv_MCMIAmHD3OCWwYvL7CDOedMbezMs4pmGGuzpk
RH2QX8UMa-RFToBg for challenge <4513455346757278126.1757192932938@java-sasl-xmpp-server>:
true
```

7.8. Step 8: Server AuthorizeCallback with authorization ID

The server determines the DID as the "authorized ID", concluding the authentication flow.

-- SERVER CALLBACK: AuthorizeCallback

```
>S --- authenticationID: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D, au
thorizationID: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D, authorizedID
: null, isAuthorized: false
S> --- authenticationID: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D, au
thorizationID: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D, authorizedID
: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D, isAuthorized: true
```

authorizationId: did%3Akey%3Az6MkfePUhxLV6cM54cgZ4bGmnEdTNm3WDf4arwh5kR3dH51D

8. Security Considerations

This section addresses the security properties of the DID-CHALLENGE SASL mechanism and the threats it is, and is not, designed to counter. Implementers SHOULD also consult the security considerations of the SASL framework (RFC4422 (<https://www.rfc-editor.org/rfc/rfc4422.html>)), the W3C Decentralized Identifiers v1.1 (<https://www.w3.org/TR/did-1.1/>) specification, and, when the optional VC/VP extension is used, the W3C Verifiable Credentials Data Model 2.0 (<https://www.w3.org/TR/2025/REC-vc-data-model-2.0-20250515/#types>) specification.

8.1. Mechanism Strength

The DID-CHALLENGE mechanism authenticates clients by asymmetric cryptography rather than by transmitting a password or a password-derived value. This eliminates an entire class of server-side risks present in password-based SASL mechanisms such as PLAIN or DIGEST-MD5: a compromise of the server's credential store yields no material that can be used to impersonate clients.

The security of the mechanism depends on the following properties holding simultaneously: (a) the signature algorithm is computationally infeasible to forge; (b) the client's private key has not been compromised; (c) the DID resolver consulted by the server returns an authentic DID document (see Section 8.6; and (d) the authentication exchange is protected from observation and tampering by a lower-layer security protocol (see Section 8.2). If any of these properties fails to hold, the security guarantees of the mechanism are reduced or eliminated entirely.

8.2. Requirement for a Confidential Channel

The DID-CHALLENGE mechanism does not itself provide a security layer (confidentiality or integrity protection of the application-layer data stream after authentication). The client transmits its DID and a cryptographic signature in the clear at the SASL layer. An eavesdropper learns the client's DID, which may be linkable to the client's real-world identity, and obtains a valid signature over a server-chosen challenge string.

The DID-CHALLENGE mechanism **MUST NOT** be used over an unprotected channel. Implementations **MUST** employ TLS (RFC8446 (<https://www.rfc-editor.org/rfc/rfc8446.html>)) or an equivalent protocol providing both confidentiality and integrity before initiating a DID-CHALLENGE exchange.

When the optional VC/VP extension (see Section 5) is used, this requirement is especially critical. Verifiable Presentations may contain sensitive personal attributes — such as name, date of birth, or professional credentials — that are transmitted in the clear at the SASL layer and **MUST** be protected by the underlying confidentiality layer.

8.3. Replay Attacks

The DID Challenge includes a nonce and a timestamp to prevent replay attacks. The nonce **MUST** be generated by a cryptographically strong (pseudo) random number generator and **MUST** be unique per challenge. The server **MUST** maintain a record of all nonces issued within the active timestamp window and **MUST** reject any DID Response whose nonce has already been accepted. A server that reuses nonces or fails to track them renders the replay defence ineffective.

The timestamp provides a complementary time-bounded validity window. The server **MUST** reject any DID Response whose challenge timestamp lies outside a configured acceptance window, with a **RECOMMENDED** default of no more than 5 minutes. Server clocks **SHOULD** be synchronized via NTP or an equivalent mechanism, since excessive clock skew will cause legitimate authentications to be rejected or, if compensated by widening the window, increase replay exposure.

Both controls apply equally to the VC/VP Challenge and VC/VP Response defined in Section 5. Servers **MUST** track VC/VP nonces independently and apply the same timestamp validation.

8.4. Man-in-the-Middle Attacks and Channel Binding

Because the client signs a server-supplied challenge, a man-in-the-middle adversary who can intercept and substitute the challenge could induce the client to produce a signature the adversary then uses to authenticate to the real server. Running the exchange over TLS substantially raises the bar for this attack. To eliminate it entirely, implementations **SHOULD** incorporate a TLS channel binding value (see RFC5929 (<https://www.rfc-editor.org/rfc/rfc5929.html>)) into the signed material, so that a signature produced within one TLS session cannot be transferred to another.

The realm field in the challenge binds the signature to a specific service context. Clients **MUST** verify that the realm in the received challenge matches the service they intend to authenticate to before computing the DID Response, and **MUST** abort the exchange on a mismatch.

8.5. Server Spoofing and Mutual Authentication

The DID-CHALLENGE mechanism provides unilateral authentication: the client proves its identity to the server, but the server does not prove its identity to the client beyond what is provided by the underlying transport. A malicious server can issue a legitimate-looking challenge and collect a valid DID Response.

Clients **MUST** validate the server's TLS certificate against a trusted certification authority or equivalent trust anchor before initiating a DID-CHALLENGE exchange. Clients **MUST NOT** proceed if certificate validation fails. Deployments with stronger mutual-authentication requirements **MAY** combine DID-CHALLENGE with a DID-based server-authentication step at the application layer, though this is outside the scope of this specification.

8.6. Choosing and Trusting DID Resolvers

The server verifies the client's signature using public key material obtained by resolving the client's DID. A malicious or compromised DID resolver that returns a fraudulent DID document could substitute attacker-controlled key material, allowing impersonation of an arbitrary DID. As discussed in W3C DIDs v1.1 - Choosing DID Resolvers (<https://www.w3.org/TR/did-1.1/#choosing-did-resolvers>), there is no universal authority that mandates a correct resolver implementation for a given DID method; server implementers MUST select DID resolver software they have independently verified and trust.

The network path between the server and its DID resolver SHOULD be protected by TLS. Where the DID method supports it, the integrity of the retrieved DID document SHOULD be verified using content integrity mechanisms before its key material is used. Servers SHOULD restrict the set of accepted DID methods to those whose resolver implementations and underlying registries have undergone independent security review.

8.7. Key Revocation, Rotation, and DID Method Properties

A DID controller who suspects key compromise SHOULD immediately update the DID document to revoke or rotate the affected verification method. There is an inherent window of exposure between the moment of compromise and the moment the revocation propagates to the server's resolver; its duration depends on registry propagation speed and the server's cache refresh policy. Servers MUST NOT rely indefinitely on cached DID documents, and SHOULD treat a DID resolution failure as an authentication failure rather than silently falling back to stale cached data.

DID methods differ significantly in their security properties. Methods such as "did:key" encode the public key directly in the identifier and support neither revocation nor rotation; a compromised private key cannot be remediated and the DID must be abandoned entirely. Methods anchored in distributed ledgers or similar registries support revocation but introduce availability and integrity dependencies on that infrastructure. Methods based on DNS (such as "did:web") inherit the DNS attack surface, including susceptibility to hijacking.

Servers SHOULD maintain an explicit list of accepted DID methods and SHOULD prefer those whose specifications have undergone independent security review, as required by W3C DIDs v1.1 - Security Requirements (<https://www.w3.org/TR/did-1.1/#security-requirements>).

8.8. Non-Repudiation

The DID Response is a cryptographic signature over a challenge that encodes a unique nonce, a timestamp, and the server's realm. Provided the client's private key is used exclusively by the DID controller and has not been compromised, this signature constitutes evidence that the DID controller authenticated to the specified server at approximately the time encoded in the challenge. This property, discussed in W3C DIDs v1.1 - Non-Repudiation (<https://www.w3.org/TR/did-1.1/#non-repudiation>), supports non-repudiation of authentication events. Deployments that require non-repudiation for compliance or forensic purposes SHOULD log and archive authentication exchanges accordingly.

8.9. Authentication vs. Authorization

Successful completion of the DID-CHALLENGE exchange proves that the client controls a private key corresponding to a verification method listed under the "authentication" relationship in its DID document. This proves control of the DID; it does not by itself confer any authorization to access resources on the server. Servers MUST maintain and enforce an authorization policy that maps authenticated DIDs to permitted operations, independently of the authentication outcome.

8.10. Private Key Protection

The security of DID-CHALLENGE rests entirely on the secrecy of the client's private key. An adversary who obtains the private key can authenticate as the corresponding DID until the DID document is updated to revoke the associated verification method — and, for DID methods that do not support revocation, indefinitely.

Client implementations MUST protect private keys in a manner commensurate with the sensitivity of the resources being accessed. Suitable measures include hardware security modules (HSMs), operating-system-provided secure key storage, or encrypted software key stores protected by a strong passphrase. Private keys MUST NOT be stored in plaintext. Implementers MUST ensure that the JWKCallback interface does not expose the private key to unauthorized processes or log files.

8.11. Security of the Optional VC/VP Extension

When the optional VC/VP extension is used, the server MUST additionally verify: that the VP proof is valid and was produced using a key with an "assertionMethod" relationship in the client's DID document; that the VP "holder" property matches the authenticated DID; that each credential's issuer signature is valid; that no credential has expired or been revoked; and that the credential type matches the type requested in the VC/VP Challenge.

Servers MUST implement credential status checking to detect revoked credentials, and MUST maintain an explicit issuer trust policy, rejecting credentials from issuers not covered by that policy. The trustworthiness of a credential issuer cannot be inferred from the credential itself. Finally, servers SHOULD request only the credential types strictly necessary for the access-control decision being made, to minimise unnecessary disclosure of personal information, particularly given that VPs are transmitted in the clear at the SASL layer (see Section 8.2).

8.12. Denial of Service

The DID-CHALLENGE mechanism introduces potential denial-of-service vectors that do not arise in password-based SASL mechanisms. Implementers SHOULD consider each of the following.

Challenge generation and nonce tracking. Each mechanism-selection message causes the server to generate a nonce and allocate an entry in its nonce-tracking table. An attacker who sends many such messages can exhaust server memory and CPU. Servers MUST enforce a short timeout on incomplete exchanges (RECOMMENDED: 30 seconds from challenge issuance), after which the nonce is discarded and any subsequent messages referencing it rejected. Servers SHOULD rate-limit challenge issuance per source address and SHOULD bound the size of the nonce table.

DID resolution amplification. Every authentication attempt requires an outbound DID resolution request. An attacker who sends many requests using different DIDs forces a corresponding number of outbound network requests, potentially stressing both the server and DID method infrastructure. Servers SHOULD cache recently resolved DID Documents for a short period (subject to the constraints in Section 8.7), rate-limit outbound resolution requests, and impose a resolution timeout.

Cryptographic verification cost. Signature verification is computationally expensive, and the server may need to try multiple candidate keys if the DID Document contains more than one

"authentication" verification method. The ordering of steps in Section 3.6) is therefore deliberate: the cheap, non-cryptographic checks (format, nonce, timestamp) are placed first so that most malformed or replayed requests are rejected before any signature verification is attempted. Servers MAY additionally impose per-source-address limits on signature verification attempts.

VC/VP extension. The optional VC/VP extension adds VP proof verification, per-credential issuer signature verification, and credential status checking to each exchange, all of which may involve further outbound network requests. The same rate-limiting measures above apply. Servers SHOULD additionally cap the number of credentials permitted in a single Verifiable Presentation and reject oversized presentations before performing any cryptographic work. Servers SHOULD cache credential status information briefly to avoid redundant outbound requests during bursts of authentication attempts.

9. Implementations

The following repositories contain various parts of an example implementation:

- * SASL client demonstration components:
<https://github.com/peacekeeper/java-sasl-client-demo>
(<https://github.com/peacekeeper/java-sasl-client-demo>)
- * SASL server demonstration components:
<https://github.com/peacekeeper/java-sasl-server-demo>
(<https://github.com/peacekeeper/java-sasl-server-demo>)
- * SASL local "Hello World" demonstration:
<https://github.com/peacekeeper/java-sasl-local-demo>
(<https://github.com/peacekeeper/java-sasl-local-demo>)
- * Implementation of a DID-based SASL authentication mechanism:
<https://github.com/peacekeeper/java-sasl-did-mechanism>
(<https://github.com/peacekeeper/java-sasl-did-mechanism>)
- * XMPP server (based on Tigase) using the DID-based SASL authentication mechanism: <https://github.com/peacekeeper/java-sasl-xmpp-server> (<https://github.com/peacekeeper/java-sasl-xmpp-server>)
- * XMPP client demo (based on Tigase) using the DID-based SASL authentication mechanism: <https://github.com/peacekeeper/java-sasl-xmpp-client-tigase> (<https://github.com/peacekeeper/java-sasl-xmpp-client-tigase>)

- * XMPP client demo (based on Smack) using the DID-based SASL authentication mechanism: <https://github.com/peacekeeper/java-sasl-xmpp-client-smack> (<https://github.com/peacekeeper/java-sasl-xmpp-client-smack>)
- * XMPP client plugin (based on Spark) using the DID-based SASL authentication mechanism: <https://github.com/peacekeeper/java-sasl-xmpp-client-spark> (<https://github.com/peacekeeper/java-sasl-xmpp-client-spark>)
- * XMPP client application (based on Spark) using the DID-based SASL authentication mechanism: <https://github.com/peacekeeper/java-sasl-xmpp-client-spark> (<https://github.com/peacekeeper/java-sasl-xmpp-client-spark>)

10. Acknowledgements

The author would like to thank the members of the KITTEN working group for their review and feedback on earlier versions of this specification.

This work was funded through NGI0 Commons Fund (<https://nlnet.nl/commonsfund>), a fund established by NLnet (<https://nlnet.nl>) with financial support from the European Commission's Next Generation Internet (<https://ngi.eu>) program.

Learn more at the NLnet project page (<https://nlnet.nl/project/DID-SASL>).

Author's Address

Markus Sabadello
Danube Tech GmbH
Margaretenstrae 70/1/7
A-1050 Wien
Austria
Phone: +43-664-3154848
Email: markus@danubetech.com