

Introduction to DID Auth

A White Paper from Rebooting the Web of Trust VI

by Markus Sabadello, Kyle Den Hartog, Christian Lundkvist, Cedric Franz,
Alberto Elias, Andrew Hughes, John Jordan, and Dmitri Zagidulin

ABSTRACT

The term DID Auth has been used in different ways and is currently not well-defined. We define DID Auth as a ceremony where an *identity owner*, with the help of various components such as web browsers, mobile devices, and other agents, proves to a *relying party* that they are in control of a DID. This means demonstrating control of the DID using the mechanism specified in the DID Document's "authentication" object. This could take place using a number of different data formats, protocols, and flows. DID Auth includes the ability to establish mutually authenticated communication channels and to authenticate to web sites and applications. Authorization, Verifiable Credentials, and Capabilities are built on top of DID Auth and are out of scope for this document. This paper gives an overview of the scope of DID Auth, supported protocols and flows, and the use of components of the DID Documents that are relevant to authentication, as well as formats for challenges and responses.

RESOURCES

This paper is a continuation of ongoing work by [Rebooting the Web of Trust](#) and other communities. Previous work includes:

- [RwOT IV: James Monaghan's topic paper "DID Auth"](#)
- [RwOT VI: Markus Sabadello's topic paper "DID Auth: Scope, Formats, and Protocols"](#)
- [RwOT VI: Kyle Den Hartog's topic paper "DID-Auth protocol"](#)

Portions of the work on this paper have been funded through a Code With Us contract awarded by the BC Developers' Exchange (<https://bcdevexchange.org/>) of the Province of British Columbia, Canada.



PTBVENTURES



sovrin

Named Sponsors for the Rebooting the Web of Trust VI Design Workshop

DEFINITIONS

Authentication: The ceremony where an *identity owner* proves to a *relying party* that the *identity owner* controls a DID, using a mechanism that is described in the DID's associated DID Document.

Authorization: A process of establishing the rights and privileges of an *identity owner* to perform certain actions, including operations on a DID itself, or in another context.

Decentralized Identifier (DID): A globally unique identifier that does not require a centralized registration authority because it is registered with distributed ledger technology or another form of decentralized network. (see [here](#))

DID Document: A structured document containing metadata that describes a DID, including authentication materials such as public keys and pseudonymous biometrics, that an entity can use to authenticate, i.e. to prove control of the DID. A DID Document may also contain other attributes or claims describing the entity. (see [here](#))

DID Record: The combination of a DID and its associated DID Document.

Identity Owner: The individual, organization or thing who created the DID, is identified by the DID that is the subject of the DID Document, and who has the ultimate authority to update or revoke the DID.

Relying Party: The individual, organization or thing that authenticates an *identity owner* using a DID Auth protocol. Also called "Verifier" in other specifications.

Verifiable Credentials: A set of one or more claims that are statements made by an issuer about a subject that is tamper-resistant and whose authorship can be cryptographically verified (see [here](#)).

INTRODUCTION

Scope

This paper defines data formats and challenge and response transports allowing an *identity owner* to prove control of a DID to a *relying party*. Proof of control of a DID is a technical interaction that may be a precursor to establishing a longer term relationship between two parties. A successful DID Auth interaction may create the required conditions to allow the parties to exchange further data in a trustworthy way. This further data could include streams of raw data, from sensors to the exchange of Verifiable Credentials. This further exchange of data is out of scope of the DID Auth protocol itself.

DID Auth may be a one-way interaction where party A proves control of a DID to party B, or a two-way interaction where mutual proof of control of DIDs is achieved. In the latter case, party A proves control of DID A to party B and party B proves control of DID B to party A.

It is in the purview of the two parties engaged in the interaction to determine the need to have a one-way or two-way DID Auth interaction. It is also in the purview of the two parties to determine if further exchanges of data such as Verifiable Credentials may be necessary to establish the nature of the relationship between the two parties.

Implementers may decide to subsume a DID Auth interaction within a higher layer interaction such as the exchange of Verifiable Credentials, which could simultaneously prove control of a DID and offer Verifiable Credentials for some transaction-specific purpose.

DID AUTH AND VERIFIABLE CREDENTIALS

Even though DID Auth is about proving control of a DID, the exchange of Verifiable Credentials associated with a DID is related to DID Auth. The relationship between DID Auth and Verifiable Credentials could be thought of in several conceptual ways:

1. **DID Auth and Verifiable Credentials exchange are separate:** At the beginning of an interaction between two parties, they need to authenticate (mutually, or just in one direction). After this is done, a protocol for exchange of Verifiable Credentials can be executed, so that the two parties can learn more about each other (and then perhaps make authorization decisions).
2. **Verifiable Credentials exchange is an extension to (or part of) DID Auth:** In this approach, proving control of an identifier, and proving possession of Verifiable Credentials are closely related, and a single protocol is used for both purposes. The Verifiable Credentials are an "optional field" in the protocol. In order to "only" prove control of an identifier, an empty set of Verifiable Credentials is exchanged.
3. **DID Auth is a certain kind of Verifiable Credential:** It is possible to think of DID Auth as an exchange of the most trivial Verifiable Credential imaginable: a self-issued Verifiable Credential that states "I am me". From this perspective, the separation between DID Auth and exchange of "other" Verifiable Credentials is blurred, and both are part of a single universal protocol.

DID Record Creation

DID Auth requires authentication material that is generated during DID Record Creation. As stated in the DID specification, the steps to create a DID Record compliant with DID Auth are:

1. Generate a **NEW_DID** as specified in the relevant DID method specification.
2. Generate a **NEW_DID_DOCUMENT** as specified in the relevant DID method specification.
 1. Set the `id` property to the value of **NEW_DID** (the DID subject).
3. Choose one or more authentication type(s) from the array of proof mechanisms.
 1. Record the `type` property in an `authentication` object of the **NEW_DID_DOCUMENT**.
4. Generate authentication material for use at a later time during authentication of the **NEW_DID**. The authentication type determines how to generate authentication material for a proof mechanism.
5. Communicate and store the authentication material, either directly or as derived material, in the **NEW_DID_DOCUMENT** and for storage by the *identity owner*. If the chosen proof mechanism is based on asymmetric keys, the authentication material in the **NEW_DID_DOCUMENT** is recorded in a `publicKey` object in the DID Document.

Example authentication and `publicKey` objects in a DID Document:

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    "type": "RsaSignatureAuthentication2018",
    "publicKey": "did:example:123456789abcdefghi#keys-1"
  }, {
    "type": "Ed25519SignatureAuthentication2018",
```

```

    "publicKey": "did:example:123456789abcdefghi#keys-2"
  }],
  "publicKey": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "RsaVerificationKey2018",
    "owner": "did:example:123456789abcdefghi",
    "publicKeyPem": "-----BEGIN PUBLIC KEY...END PUBLIC KEY-----\r\n"
  }, {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2018",
    "owner": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }]
}

```

DID Auth can use different transports for exchanging challenges and responses between an *identity owner* and a *relying party*. One such transport uses an HTTP POST call to a DID Auth service endpoint. This service endpoint can be discovered from a DID Document.

Example DID Auth service endpoint in a DID Document:

```

{
  "@context": "https://w3id.org/did/v1",
  "id": "did:example:123456789abcdefghi",
  "service": {
    "type": "DidAuthService",
    "serviceEndpoint": "https://auth.example.com/did:example:123456789abcdefg"
  }
}

```

Authentication of a DID

This section describes a conceptual authentication framework of the DID Auth protocol. It relates general concepts of authentication to specific concepts of DID Auth.

Authentication of a DID: Similar to other authentication methods, DID Auth relies on a challenge-response cycle in which a *relying party* authenticates the DID of an *identity owner*. During this cycle, an *identity owner* demonstrates control of their authentication material that was generated and distributed during DID Record Creation through execution of the authentication-proof mechanism.

Challenge: The way an *identity owner* or their agent encounters an authentication challenge, as well as the format of the challenge, will vary depending on the situation. For example, they can come across a "Sign in with DID Auth" button or a QR code on a website. Or, in the case of an API call, the *relying party* may respond to a request by asking for authentication (the HTTP 401 `Unauthorized` response is a classic example, although DID Auth covers many use cases beyond HTTP).

Challenge principles:

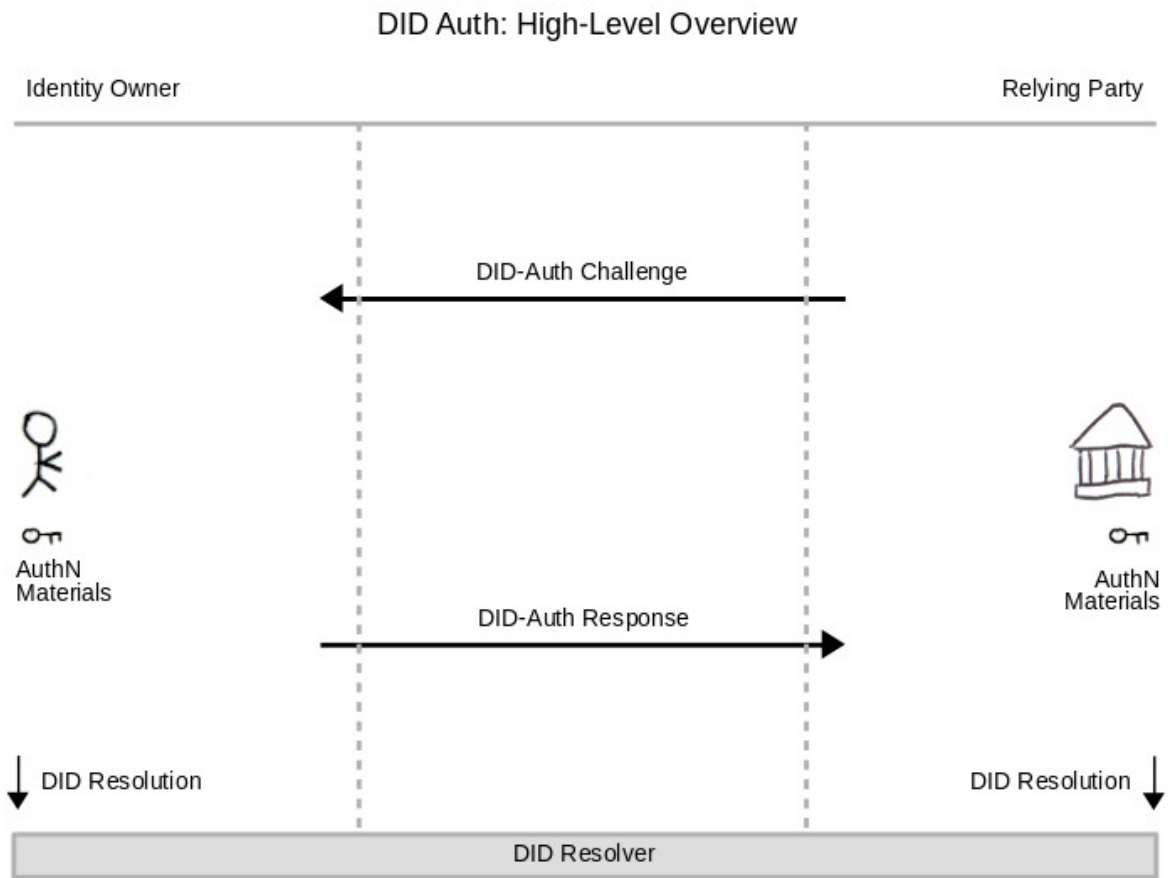
- The *relying party* may or may not know the *identity owner's* DID at the time the challenge is constructed, and therefore the *identity owner's* DID may or may not be included in the challenge.
- If the DID is known at the time of challenge construction, then the *relying party* may use the contents of the DID Document to select preferred authentication methods or service endpoints.
- The challenge that is sent by the *relying party* may or may not itself contain a proof of the *relying party's* control of a DID.
- The *relying party* may or may not need additional transport-specific information about the *identity owner* in order to be able to deliver the challenge (e.g. a DID Auth service endpoint). This additional protocol-specific information may be discoverable from the *identity owner's* DID if it is known to the *relying party*.
- The *relying party* should include a nonce, to prevent replay attacks and to help link the challenge to a subsequent response.

Response: Based on the challenge, the *identity owner* then constructs a response that proves control of their DID. This often involves a cryptographic signature, but can include other proof mechanisms. (As mentioned earlier, the response may also contain Verifiable Credentials that the *relying party* asked for in the challenge.) After receiving the response, the *relying party* resolves the *identity owner's* DID, and verifies that the response is valid for a prior challenge (for example, verifying the response signature by using a `publicKey` object contained in the DID Document).

Response principles:

- The *identity owner* may or may not need additional transport-specific information about the *relying party* in order to be able to deliver the response (e.g. a callback URL). This additional protocol-specific information may be included in the challenge, or it may be discoverable from the *relying party's* DID that is included in the challenge.
- The *relying party* must be able to internally link a response to a prior challenge. This can be done with a nonce or message identifier in the challenge that must also be included in the response. It can also be done by including the entire original challenge in the response.
- Multiple devices, user agents, and other technical components may act on behalf of the *identity owner* to receive and process the challenge. For example, an identity owner's DID Auth service endpoint may receive the challenge and relay it to the *identity owner's* mobile app.
- The *identity owner's* component that sends the response may or may not be the same component as the one that received the challenge. E.g. the challenge may be received as HTTP POST by a DID Auth service, but the response may be sent as HTTP POST by a mobile app (see **DID Auth Architecture 3**).
- The *relying party's* component that receives the response may or may not be the same component as the one that sent the challenge. E.g. the challenge may be sent as deep link by a mobile web page, but the response may be received as HTTP POST by a web server (see **DID Auth Architecture 2**).

Generic DID Auth architecture:



CHALLENGE FORMATS

In a DID Auth interaction, a challenge is transmitted by a *relying party* to an *identity owner*, asking the *identity owner* to return a response that proves their control of a DID. This section documents data formats for such challenges.

JWT format

Example: uPort:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NksifQ.eyJpc3MiOiIyY2VYdWZIR0RwVTUxYmZLQnNaRGR1N0plOXdlSjNyN3NWRyIsImIhdCI6MTUyNTg2NTM5OCwicmVxdWVzdGVkIjpbIm5hbWUiLCJwaG9uZSIsImNvdW50cnkiLCJhdmF0YXkiXSwicGVybnlzc2lvdnMiOlsibm90aWZpY2F0aW9ucyJdLCJjYWxsYmFjayI6Imh0dHBzOi8vY2hhc3F1aS51cG9ydC5tZS9hcGkvdjEvdG9waWV29EVGtmV2hvQmZ0N1BaOSIsIm5ldCI6IjB4NCIsImV4cCI6MTUyNTg2NTk5OCwidHlwZSI6InNoYXJlUmVxIn0.sQnZv63KaiWj7uQzLHLZ2jfZCZTOPz-ei7zqiUeDpjCmAdrKhO_uBujYhERJ6m-vuHcoAKuibXAPoBVHI7_H4A
```

Header and payload decodes to:

```
{
  "typ": "JWT",
  "alg": "ES256K"
}
{
  "iss": "2oeXufHGDpU51bfKBSZDdu7Je9weJ3r7sVG",
  "iat": 1525865398,
  "requested": [
    "name",
    "phone",
    "country",
    "avatar"
  ],
  "permissions": [
    "notifications"
  ],
  "callback": "https://chasqui.uport.me/api/v1/topic/GoDTkfWhoBft7PZ9",
  "net": "0x4",
  "exp": 1525865998,
  "type": "shareReq"
}
```

Example: Jolocom:

```
eyJ0eXAiOiJKV1QiLCJhbGciOiJFUzI1NksifQ.eyJpYXQiOiJlMjg5OTc4NDIyNzUsInJlcXVlc3RlZENyZWRLbnRyYXZzIjpbeyJ0eXB1IjpbIkNyZWRLbnRyYXZzIjpbLCJQcm9vZk9mRW1haWwDcmVkdW50aWZlIjpbImNvdW50cmFpbnRzIjpbImFuZCI6MTUyNTg2NTM5OCwicmVxdWVzdGVkIjpbIm5hbWUiLCJwaG9uZSIsImNvdW50cnkiLCJhdmF0YXkiXSwicGVybnlzc2lvdnMiOlsibm90aWZpY2F0aW9ucyJdLCJjYWxsYmFjayI6Imh0dHBzOi8vY2hhc3F1aS51cG9ydC5tZS9hcGkvdjEvdG9waWV29EVGtmV2hvQmZ0N1BaOSIsIm5ldCI6IjB4NCIsImV4cCI6MTUyNTg2NTk5OCwidHlwZSI6InNoYXJlUmVxIn0.sQnZv63KaiWj7uQzLHLZ2jfZCZTOPz-ei7zqiUeDpjCmAdrKhO_uBujYhERJ6m-vuHcoAKuibXAPoBVHI7_H4A
```

Header and payload decodes to:

```
{
  "typ": "JWT",
  "alg": "ES256K"
}
{
  "iat": 1528997842275,
  "requestedCredentials": [
    {
      "type": ["Credential", "ProofOfEmailCredential"],
      "constraints": {
        "and": [
          { "===": [ true, true ] },
          { "!=": [ { "var": "issuer" }, { "var": "claim.id" } ] }
        ]
      }
    }
  ],
  "requesterIdentity":
  "did:jolo:b310d293aeac8a5ca680232b96901fe85988fde2860a1a5db69b49762923cc88",
  "callbackURL": "https://demo-sso.jolocom.com/proxy/authentication/aws6i"
}
```

References:

- <https://github.com/uport-project/specs/blob/develop/messages/sharereq.md>
- <https://demo-sso.jolocom.com/>

JSON-LD Verifiable Credentials format

Example: Verifiable Credentials:

```
{
  "type": ["Credential"],
  "claim": {
    "publicKey"
  }
}
```


JSON-LD Verifiable Credentials format

The following Verifiable Credential contains a claim of a certain public key as a property of the *identity owner*.

```
{
  "type": ["VerifiablePresentation"],
  "issuer": "did:example:123456789abcdefghi",
  "issued": "2018-03-07",
  "claim": {
    "id": "did:example:123456789abcdefghi",
    "publicKey": "did:example:123456789abcdefghi#keys-2"
  },
  "proof": {
    "type": "Ed25519Signature2018",
    "created": "2018-01-01T21:19:10Z",
    "creator": "did:example:123456789abcdefghi#keys-2",
    "nonce": "c0aelc8e-c7e7-469f-b252-86e6a0e7387e",
    "signatureValue": "..."
  }
}
```

CHALLENGE TRANSPORTS

A DID Auth challenge may be delivered by a *relying party* to an *identity owner* in different ways. DID Auth defines a few common ways that this can be done.

DID Auth Service Endpoint

If the *identity owner's* DID is known to the *relying party*, it resolves it to its associated DID Document to find a DID Auth service endpoint and sends an HTTP POST with the challenge to that endpoint.

Scan QR Code from Mobile App

The *relying party* may display a DID Auth challenge encoded as a QR code, which is delivered to the *identity owner* by scanning it with a DID-Auth-compatible mobile app.

Mobile Deep Link

A mobile app or webpage may itself be a *relying party* and deliver a DID Auth challenge to another mobile app that represents the *identity owner*.

Example:

```
did-auth:jwt/...
```

Custom Protocol Handler

Similar to a mobile deep link, a web page can contain a web-based protocol link that contains a DID Auth challenge.

Example:

```
<a href="did-auth:jwt/...">Login with DID Auth</a>
```

Invoke User Agent's JavaScript API

A *relying party* web site may deliver a DID Auth challenge to the *identity owner* by invoking an API via a JavaScript function in the *identity owner's* user agent.

Example: Browser Credential Handler API

```
const credential = await navigator.credentials.get({
  web: {
    VerifiableProfile: {
      name: true
    }
  }
});
console.log('credential received', credential);
```

References:

- <https://credential-verifier.demo.digitalbazaar.com/>

Form Redirect

A *relying party* web site may deliver a DID Auth challenge to the *identity owner* by redirecting to a DID Auth web site that acts on the *identity owner*'s behalf (similar to a classic IdP in OpenID Connect or SAML protocols).

Example:

```
<form action="https://auth.example.com/did:example:123456789abcdefg" method="post">
  <input type="hidden" name="challenge" value="...">
  <input type="submit" value="Submit!">
</form>
```

Device-to-device Communication

If both the *relying party* and *identity owner* meet physically with devices they control, the *relying party* can deliver a DID Auth challenge to the *identity owner* via direct communication between devices, using Bluetooth, NFC, WiFi, etc.

RESPONSE TRANSPORTS

HTTP POST to Callback URL

The DID Auth response may be delivered to the *relying party* in the form of an HTTP POST to a callback URL. This callback URL may be known to the *identity owner* in advance, it may have been included in the challenge, or it may be discoverable from the *relying party's* DID.

Scan QR Code from Mobile App

The *identity owner* may encode a DID Auth response as a QR code, which is delivered to the *relying party* by scanning it with a DID-Auth-compatible mobile app.

Fulfill JavaScript Promise

If the DID Auth challenge was delivered to the *identity owner* via a JavaScript API, then the response may be returned to the *relying party* via fulfillment of a JavaScript promise.

Device-to-device Communication

If both the *relying party* and *identity owner* meet physically with devices they control, the *identity owner* can deliver a DID Auth response to the *relying party* via direct communication between devices, using Bluetooth, NFC, WiFi, etc.

ARCHITECTURES

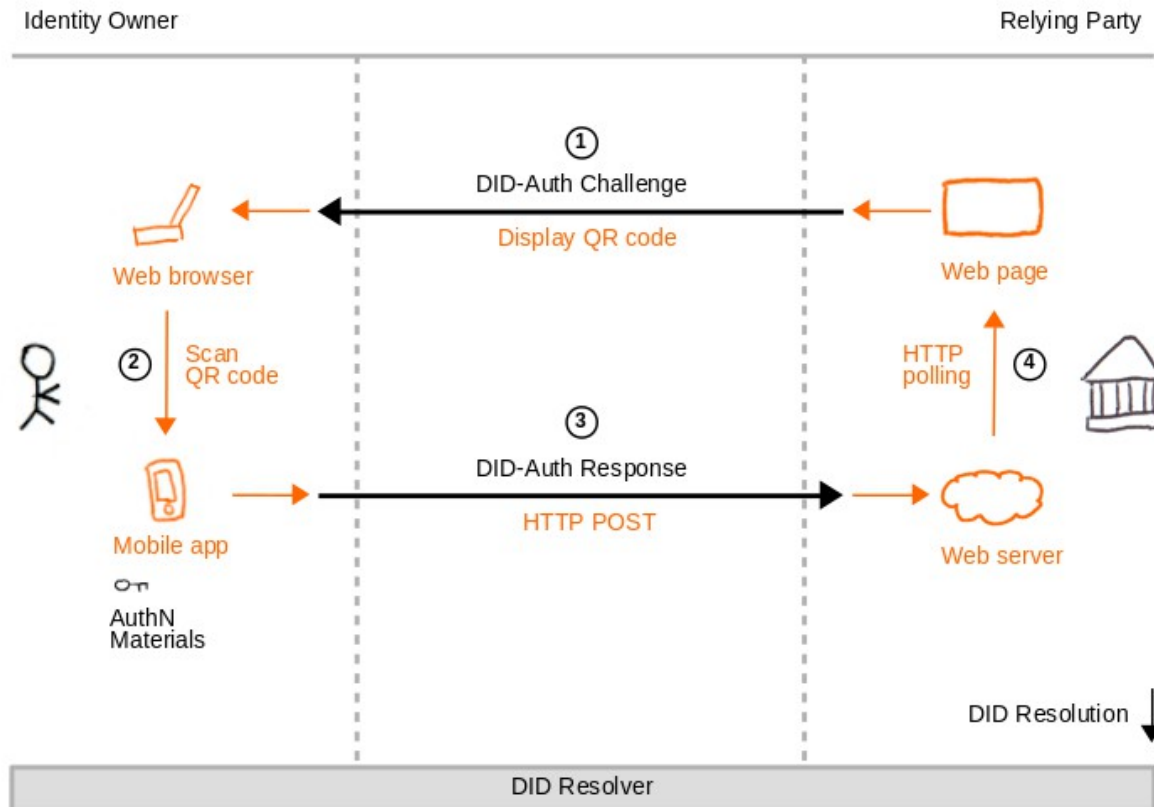
Based on the above challenge and response formats and transports, it is possible to construct architectures for various complete DID Auth interactions. When selecting an appropriate combination, it may be useful to keep in mind that the example architectures in this section mainly differ along four dimensions:

1. Is the DID known to the *relying party* at the time of challenge construction?
2. What is the transport mechanism of the DID Auth challenge?
3. What is the location of authentication material? (i.e. where are the secrets stored?)
4. What is the transport mechanism of the DID Auth response?

Therefore, the following DID Auth architectures should be understood only as some examples of possible combinations of design options.

DID Auth Architecture 1: Web Page and Mobile App

DID Auth Architecture 1: Web page and mobile app



DID Auth Diagram 1

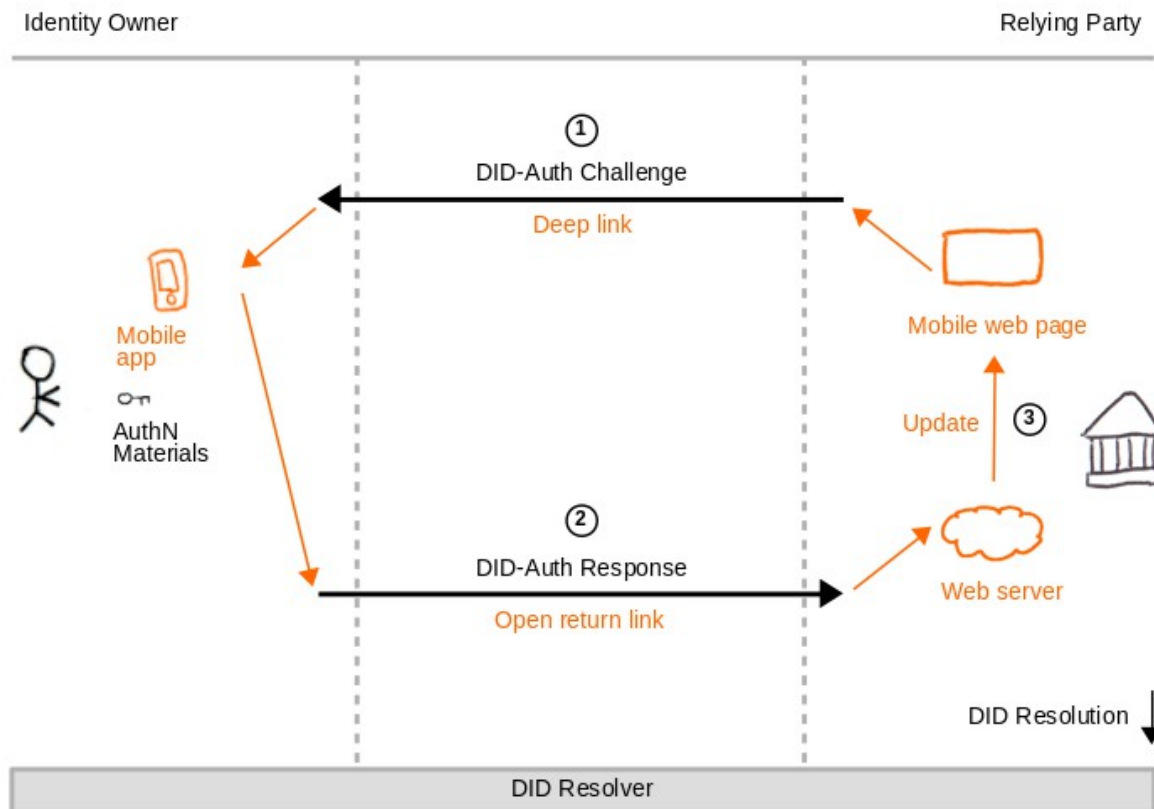
1. Relying party's **web page** displays a QR code (with challenge) to *identity owner's web browser*.
2. { *Identity owner's mobile app* scans QR code (with challenge) from *identity owner's web browser*. }
3. *Identity owner's mobile app* sends HTTP POST (with response) to *relying party's web server*.
4. { *Relying party's web server* is polled with HTTP GET (with response) by *relying party's web page*. }

References:

- <https://github.com/uport-project/specs/blob/develop/messages/sharereq.md>
- <https://demo-sso.jolocom.com/>

DID Auth Architecture 2: Mobile Web Page and Mobile App

DID Auth Architecture 2: Mobile web page and mobile app



DID Auth Diagram 2

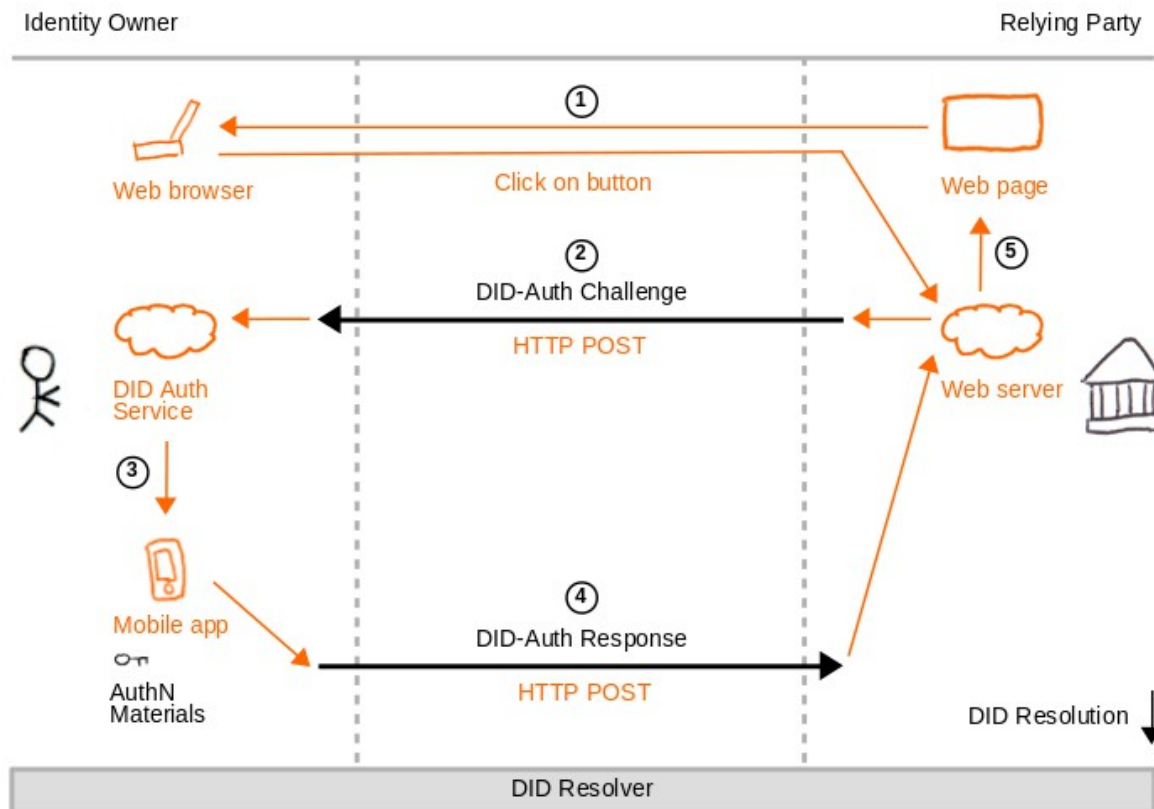
1. Relying party's **mobile web page** redirects via deep link (with challenge) to identity owner's **mobile app**.
2. Identity owner's **mobile app** opens return link (with response) to relying party's **web server**.
3. { Relying party's **web server** updates relying party's **mobile web page**. }

References:

- <https://github.com/uport-project/specs/blob/develop/messages/sharereq.md>

DID Auth Architecture 3: Web Page and DID Auth Service (1)

DID Auth Architecture 3: Web page and DID Auth service (1)



DID Auth Diagram 3

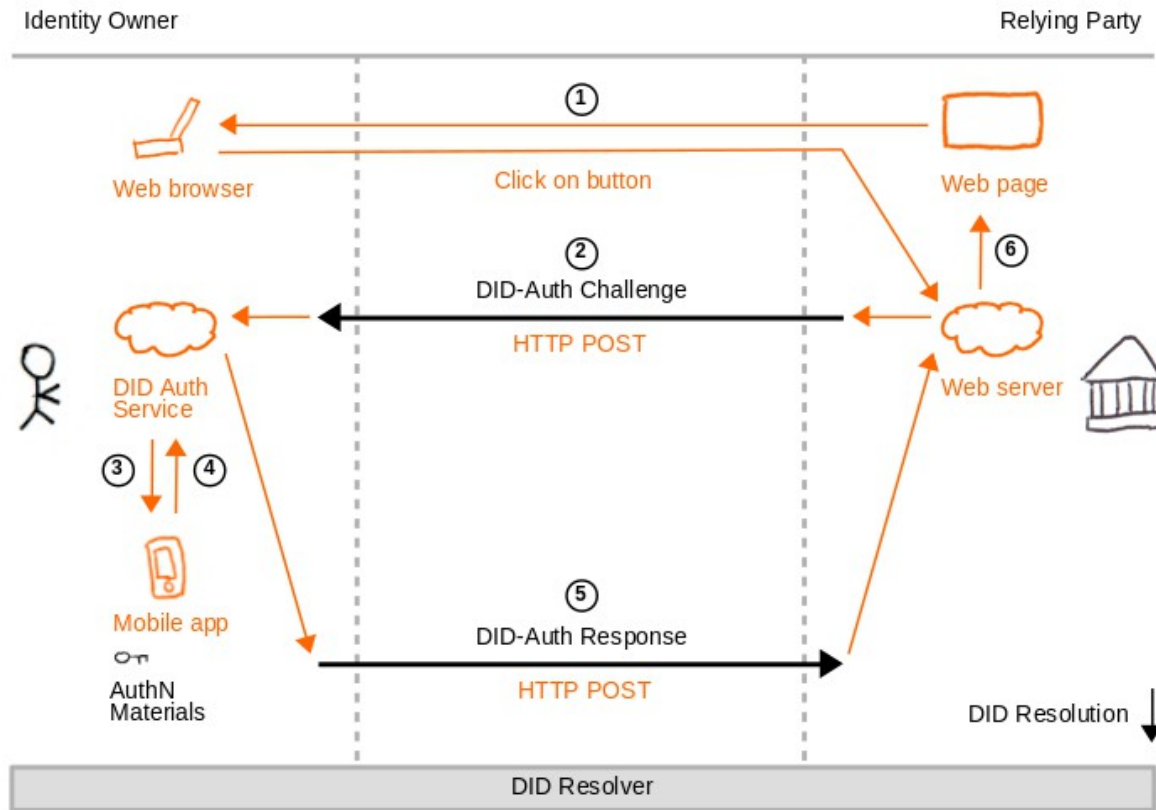
1. { Relying party's **web page** contains a link or button that calls the *relying party's web server*. }
2. *Relying party's web server* sends HTTP POST (with challenge) to *identity owner's DID Auth service*.
3. { *Identity owner's DID Auth service* sends push notification (with challenge) to *identity owner's mobile app*. }
4. *Identity owner's mobile app* sends HTTP POST (with response) to *relying party's web server*.
5. { *Relying party's web server* is polled with HTTP GET (with response) by *relying party's web page*. }

References:

- <https://github.com/WebOfTrustInfo/rebooting-the-web-of-trust-spring2018/blob/master/draft-documents/Identity%20Hub%20Attestation%20Handling.md>

DID Auth Architecture 4: Web Page and DID Auth Service (2)

DID Auth Architecture 4: Web page and DID Auth service (2)

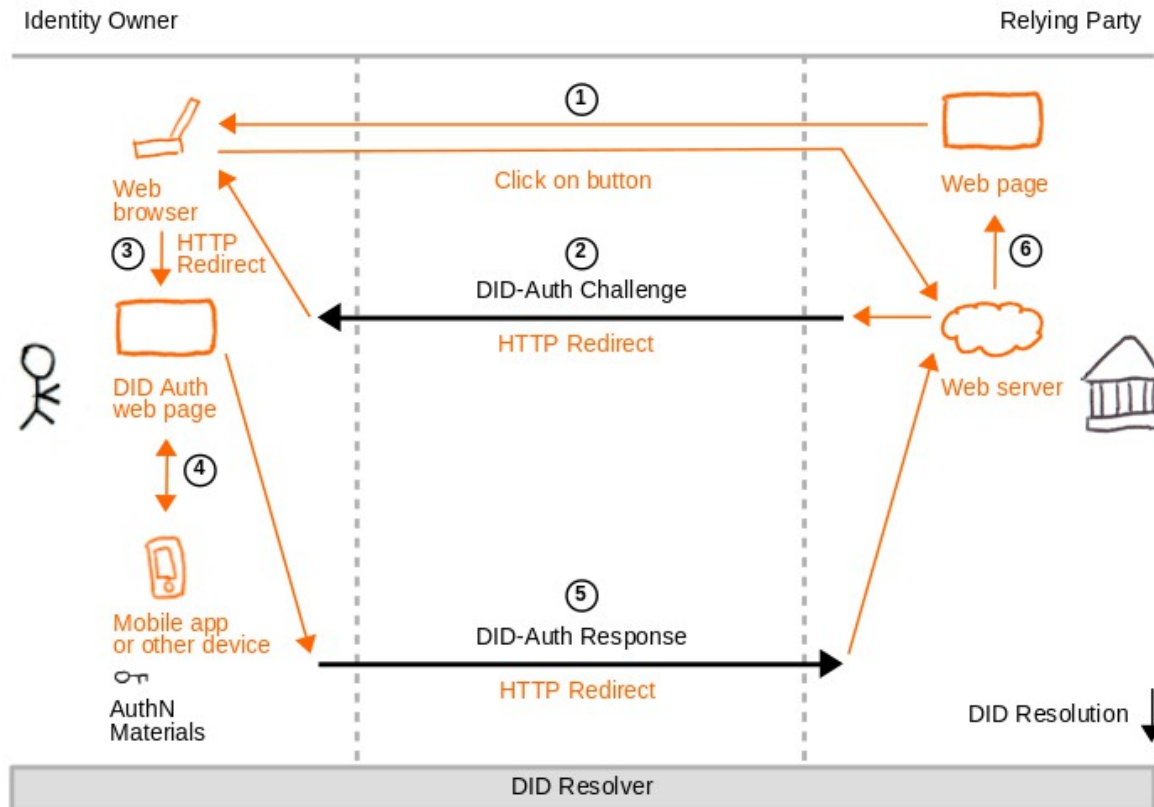


DID Auth Diagram 4

1. { Relying party's **web page** contains a link or button that calls the *relying party's web server*. }
2. Relying party's **web server** sends HTTP POST (with challenge) to *identity owner's DID Auth service*.
3. { Identity owner's **DID Auth service** sends push notification (with challenge) to *identity owner's mobile app*. }
4. { Identity owner's **mobile app** sends HTTP POST (with response) to *identity owner's DID Auth service*. }
5. Identity owner's **DID Auth service** sends HTTP POST (with response) to *relying party's web server*.
6. { Relying party's **web server** is polled with HTTP GET (with response) by *relying party's web page*. }

DID Auth Architecture 5: Web Page and DID Auth Web Page

DID Auth Architecture 5: Web page and DID Auth web page

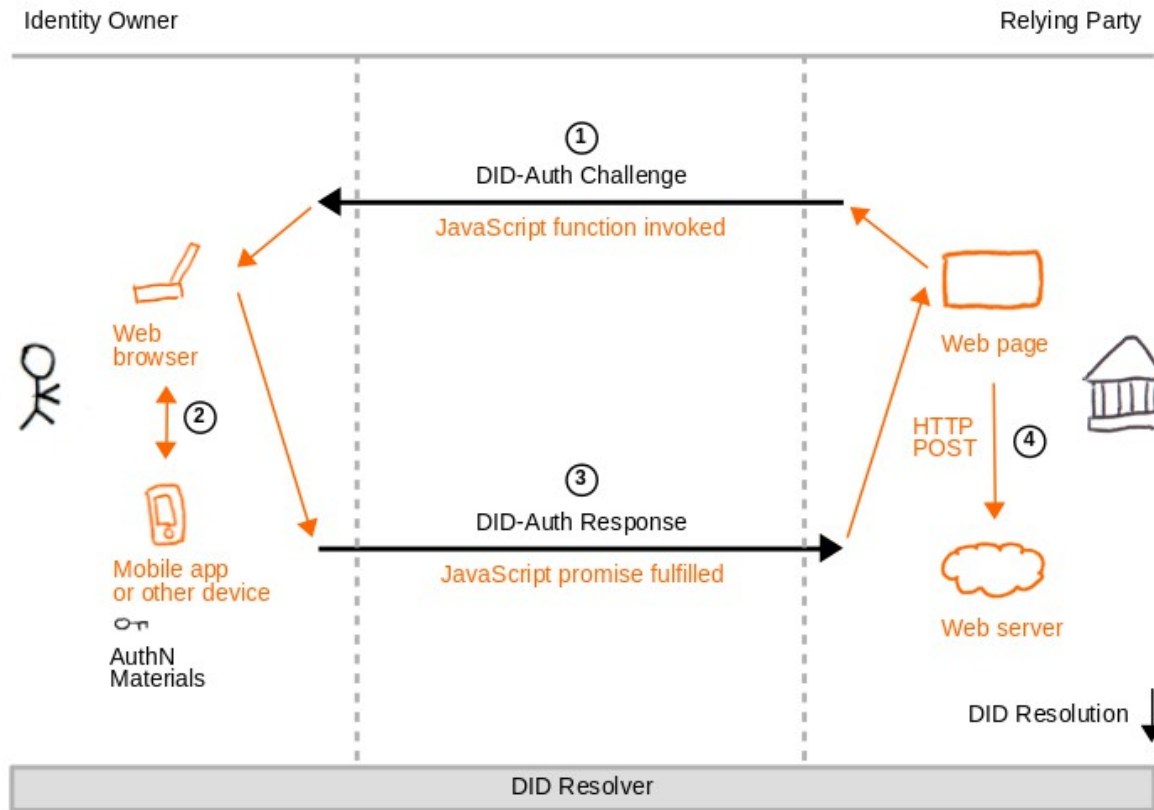


DID Auth Diagram 5

1. { Relying party's **web page** contains a link or button that calls the *relying party's web server*. }
2. *Relying party's web server* sends HTTP Redirect (with challenge) to *identity owner's web browser*.
3. { *Identity owner's web browser* follows HTTP Redirect (with challenge) to *identity owner's DID Auth web page*. }
4. { *Identity owner's DID Auth web page* optionally interacts (with challenge and response) with *identity owner's mobile app or other device*. }
5. *Identity owner's DID Auth web page* follows HTTP Redirect (with response) to *relying party's web server*.
6. { *Relying party's web server* updates *relying party's web page*. }

DID Auth Architecture 6: Web Page and Web Browser

DID Auth Architecture 6: Web page and web browser



DID Auth Diagram 6

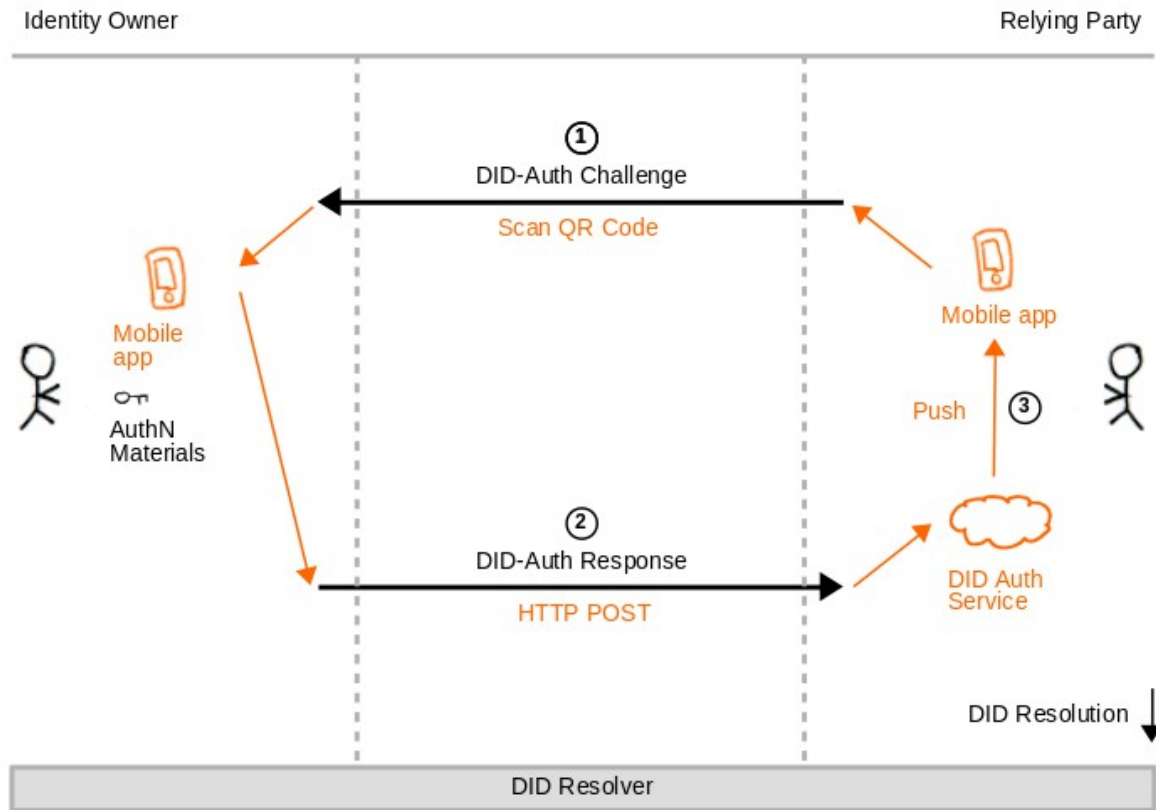
1. Relying party's **web page** calls JavaScript function (with challenge) on *identity owner's web browser*.
2. { *Identity owner's web browser* optionally interacts (with challenge and response) with *identity owner's mobile app or other device*. }
3. *Identity owner's web browser* triggers JavaScript promise (with response) on *relying party's web page*.
4. { *Relying party's web page* sends HTTP POST (with response) to *relying party's web server*. }

References:

- [Credential Handler API](#)
- [DID Auth browser add-on](#)

DID Auth Architecture 7: Mobile Apps and DID Auth Service

DID Auth Architecture 7: Mobile apps and DID Auth service

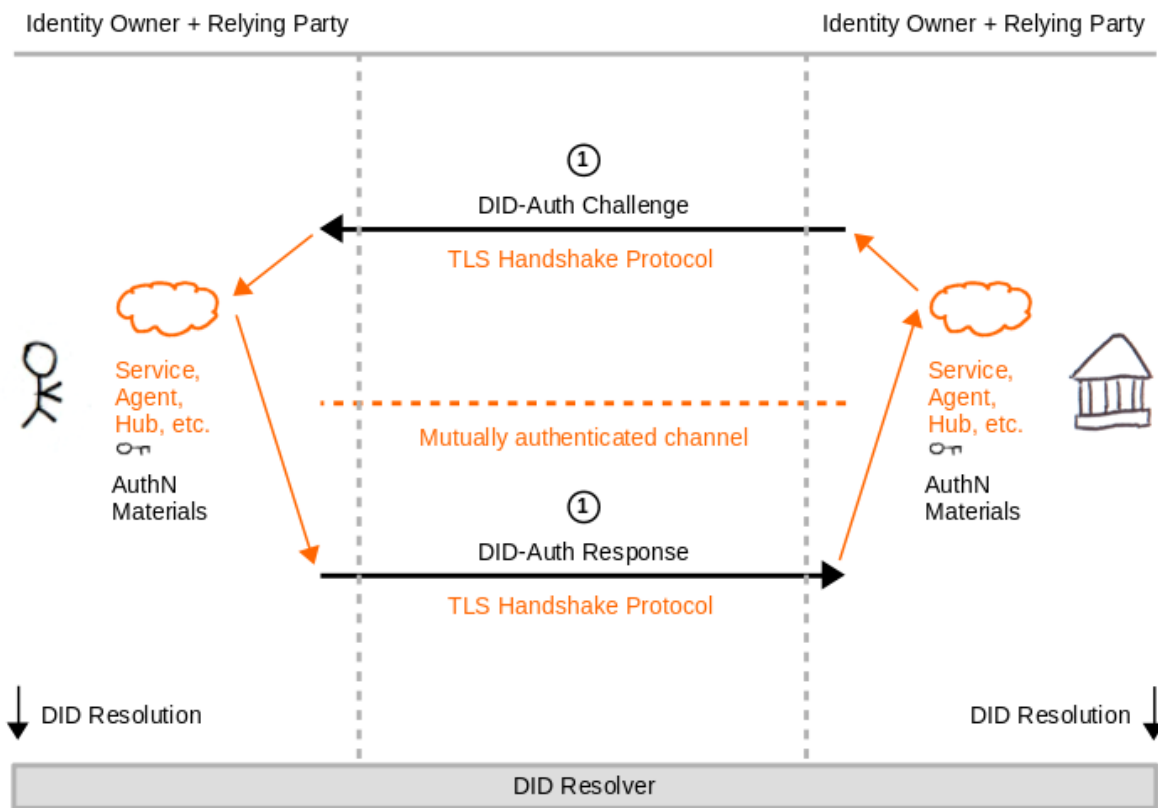


DID Auth Diagram 7

1. *Relying party's mobile app* displays a QR code (with challenge) that is scanned by *identity owner's mobile app*.
2. *Identity owner's mobile app* sends HTTP POST (with response) to *relying party's DID Auth service*.
3. { *Relying party's DID Auth service* sends push notification (with response) to *relying party's mobile app*. }

DID Auth Architecture 8: DID-TLS

DID Auth Architecture 8: DID-TLS



DID Auth Diagram 8

1. Two **services** (or agents, or hubs, etc.) act as both *identity owner* and *relying party* and engage in TLS handshake where mutual challenges and responses are exchanged and verified.

Note that while TLS specifications support mutual authentication handshakes, in practice some implementations are not well-suited for trust models that are not based on traditional certificate authorities.

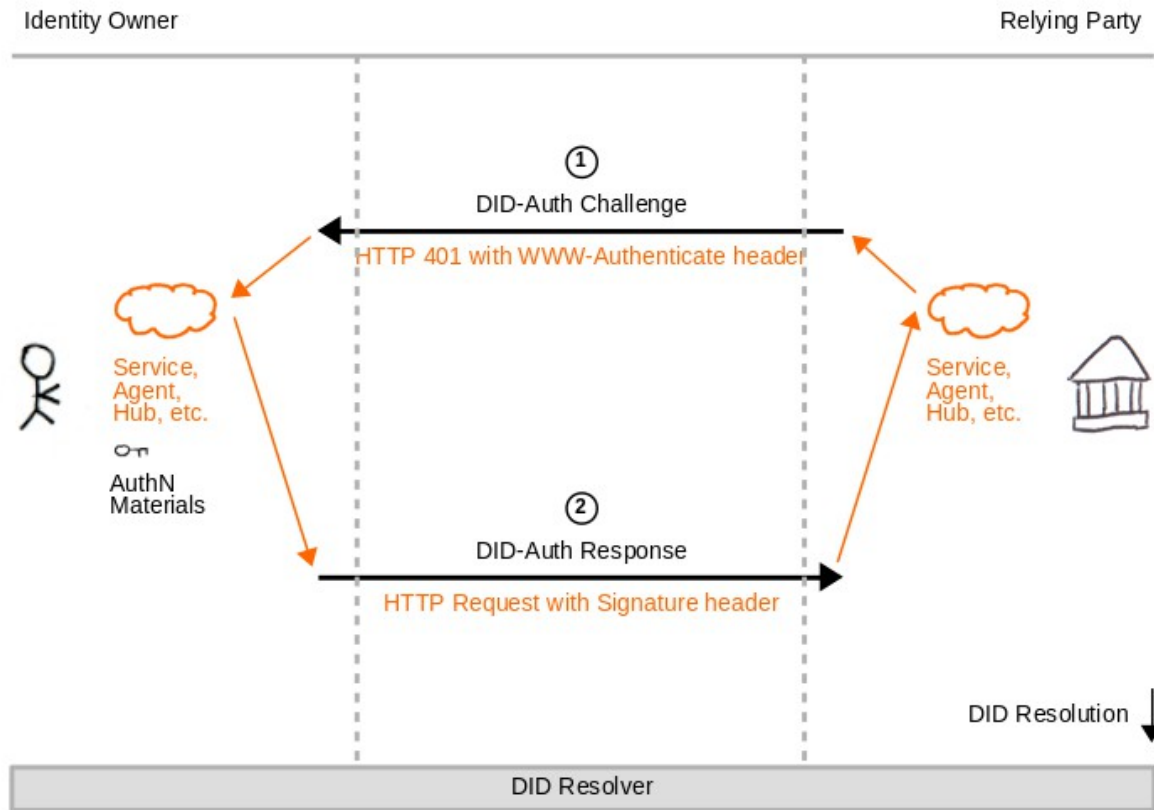
Besides TLS, other protocols can also be used to establish secure channels with mutual authentication based on DIDs, such as CurveZMQ, or custom handshake protocols.

References:

- [TLS Flex](#)
- [DID-Auth Protocol](#)

DID Auth Architecture 9: HTTP Signatures

DID Auth Architecture 9: HTTP Signatures



DID Auth Diagram 9

1. *Relying party's service (or agent, or hub, etc.)* answers an HTTP request by returning HTTP code 401 with a WWW-Authenticate header to *identity owner's service (or agent, or hub, etc.)*.
2. *Identity owner's service (or agent, or hub, etc.)* sends an HTTP request to *relying party's service (or agent, or hub, etc.)* and includes an HTTP Signature.

Example:

```
POST /api/v1/issuerservices HTTP/1.1
Host: testhost.gov.bc.ca
Accept: */*
Authorization: Signature keyId="did:example:123456789abcdefghi#keys-1",algorithm="rsa-sha256",headers="(request-target) accept user-agent",signature="214BeK0YJ9P2wmMXBjZNNXDMT4prN1c32Zksl1l1kJYkJeLp3z bz4r1WfgC1td103"
```

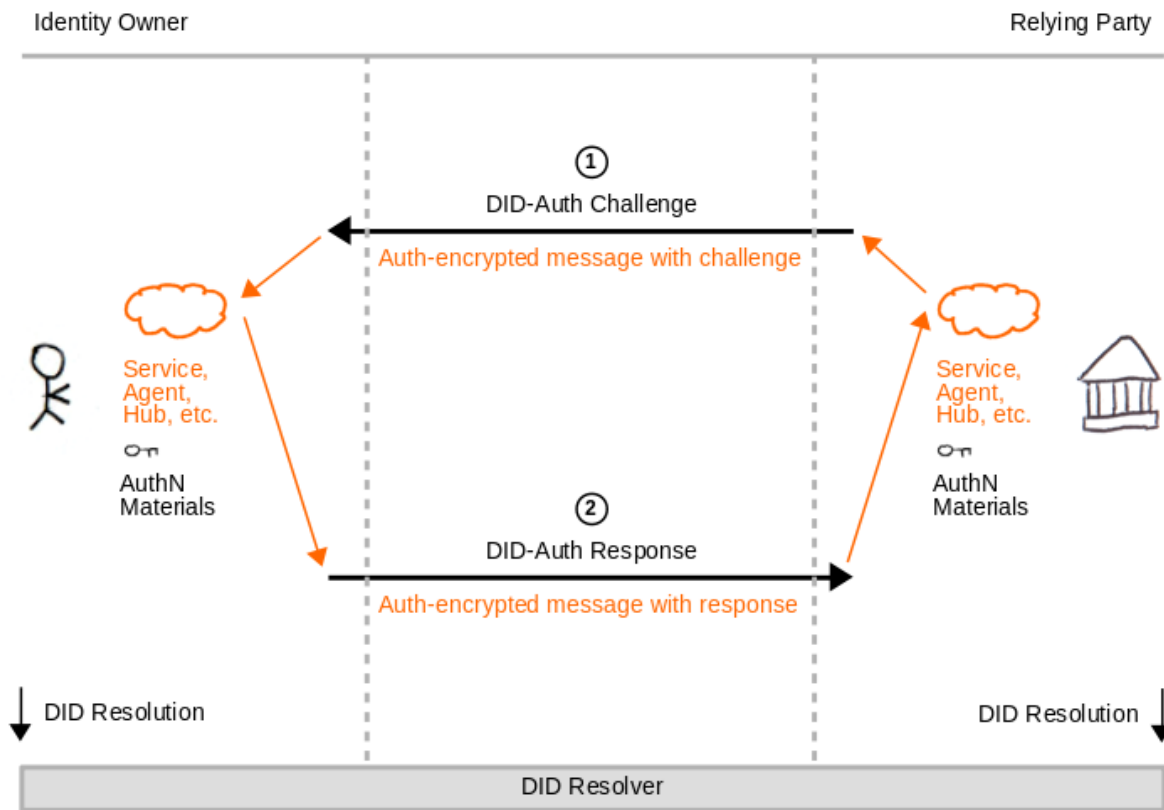
m7AyY734qbau+GswENDXaqxeTaP6LSMLWr6FexWMVgBbMzH1KDMhJlozTMFPkMsG1buDpRKwEPqnX1Yy6ld
HLe8mIJfSAEUy5P/Hf3y1b1kI8XyHNVbChFJLiUkOocF7XsFuTfoB+MJSEUqJDnuKibiF+Ap9rxI7J7Uroe
6EjaVYqLXnGbpU8j8Oxn5QzGBZFCA/j6XgHy4NK9fG9pcCyyAPGzSYi1RWjDWFyS0RDQAXFBBNgyskXAgss
KuVS2AFwPvXcHb5mhvKFUYMvMESg=="

References:

- [HTTP Signatures](#)
- [DID Auth HTTP Proxy](#)

DID Auth Architecture 10: Authenticated Encryption

DID Auth Architecture 10: Authenticated Encryption



DID Auth Diagram 10

1. *Relying party's service (or agent, or hub, etc.)* sends an auth-encrypted challenge to *identity owner's service (or agent, or hub, etc.)*. This challenge is then decrypted and authenticated by *identity owner's service (or agent, or hub, etc.)* as part of the decryption process.
2. *Identity owner's service (or agent, or hub, etc.)* then auth-encrypts a response message and sends it to *relying party's service (or agent, or hub, etc.)*, which decrypts and authenticates the response of the challenge.

References:

- [NaCl crypto_box based on Curve25519, Salsa20, and Poly1305](#)

RELATION TO OTHER TECHNOLOGIES

Biometrics

Biometrics are recognizable and verifiable data that is unique, specific to an *identity owner*, and available to every human being. Authentication with biometrics involves matching a non-reversible biometric template (or IBV, Initial Biometric Vector) against biometric input data (or CBV, Candidate Biometric Vector).

Biometrics can be used in several ways in conjunction with DID Auth:

1. Biometrics such as fingerprints or voice can protect secret authentication material (such as a private key) on an *identity owner's* physical device, which is used to construct a response to the *relying party's* challenge. In this case, biometrics are only used locally to protect the *identity owner's* part of the DID Auth challenge-response cycle; the proof mechanism listed in a DID Document does not directly involve biometrics.
2. Biometric protocols such as [IEEE 2410-2015 "BOPS"](#) or [Web Authentication](#) can be adapted to augment DID Auth flows by using biometrics in a standard way. In both protocols, no biometric data is exchanged between the *relying party* and *identity owner*.
3. Biometrics can also play a more direct role as a proof mechanism that is listed in a DID Document and used during a DID Auth challenge-response cycle to prove control of a DID. In this case, certain biometric data may be exchanged between a *relying party* and an *identity owner*; biometric service providers such as [iRespond](#) can assist the process by offering to perform the biometric matching procedure via a remote service.

In any case, since biometrics are sensitive data with special properties (i.e. their semi-public nature and the inability to revoke them), certain principles must be carefully followed (e.g. no storage of biometrics in centralized silos, no storage of biometrics on a blockchain).

References:

- [Six Principles for Self-Sovereign Biometrics](#)
- [Remove biometric templates from DID spec elements #62](#)

Other Public Key Infrastructure (PGP, SSH, etc.)

Many existing applications and services that today depend on public-key cryptography for authentication could be augmented to use DIDs as identifiers and to use DID resolution for retrieving a DID's associated public key instead of registering public keys as identifiers directly, e.g.:

- An e-mail may be signed using MIME Security and OpenPGP formats, but using key material associated with DIDs instead of static public keys that are exchanged in advance.
- An SSH server may rely on a list of DIDs in a `~/.ssh/authorized_dids` file instead of relying on a list of public keys in the traditional `~/.ssh/authorized_keys` file.

WebAuthn

WebAuthn is a specification for a JavaScript API that enables FIDO Authentication in the browser. The architecture resembles **DID Auth Architecture 6** in this paper.

In WebAuthn, a *relying party* associates a public key with an *identity owner* during a registration process. In subsequent login processes, the *identity owner* proves control of that same public key. Different public keys can be used for different

relying parties based on an "origin".

WebAuthn Registration: Register(Account, Origin)

WebAuthn Registration Response: RegisterResponse(PublicKeyCredential, Attestation, Origin)

WebAuthn Login: Sign(Challenge, Origin)

WebAuthn Login Response: SignResponse(SignedChallenge, Origin)

In order to adapt WebAuthn to support DIDs, a *relying party* should associate DIDs rather than public keys with *identity owners* (by using a DIDCredential instead of a PublicKeyCredential). During the login process, the *identity owner* includes their DID in the login response, which the *relying party* uses to look up the current public key from the DID Document.

WebAuthn+DID Registration: Register(Account, Origin)

WebAuthn+DID Registration Response: RegisterResponse(DIDCredential, Attestation, Origin)

WebAuthn+DID Login: Sign(Challenge, Origin)

WebAuthn+DID Login Response: SignResponse(DID, SignedChallenge, Origin)

Additional Notes:

- WebAuthn+DID must ensure that the publicKey objects in the DID Document correspond to the keys used by the FIDO authenticators.
- WebAuthn+DID requires no service endpoints in the DID Document.
- Ideally, a different DID should be used for each WebAuthn "origin".

References:

- [IIW #26 Session Notes "WebAuthn + DID Auth"](#)
- [Web Authentication: An API for accessing Public Key Credentials Level 1](#)

OpenID Connect

OpenID Connect (OIDC) is an authentication protocol built on the OAuth 2.0 protocol. In its most common web-based form, an end-user's user agent is redirected by a *relying party* (OAuth 2.0 client) to an OpenID Provider (OAuth 2.0 authorization server), which authenticates the end-user and redirects them back to the *relying party*. The architecture resembles **DID Auth Architecture 5** in this paper.

OpenID Connect is highly modular and customizable. It can potentially relate to DID Auth in the following ways:

DID Auth as OIDC Local Authentication Method

Within established OIDC deployments, DID Auth can serve as a drop-in "local authentication method", i.e. a method that

determines how the OpenID Provider authenticates the end-user (to replace username and password).

References:

- [Authentication Context Class](#)

DID Auth as Alternative OIDC Provider Discovery

For use cases where the DID is known beforehand, DID resolution can serve as an alternative mechanism for OIDC provider discovery (alternative to OpenID Connect Discovery / WebFinger) via service endpoint.

Example OpenID Connect service endpoint in DID Document:

```
{
  "service": [{
    "id": "did:example:123456789abcdefghi;openid",
    "type": "OpenIdConnectVersion1.0Service",
    "serviceEndpoint": "https://openid.example.com/"
  }]
}
```

References:

- [OpenID Connect Discovery](#)
- [WebFinger](#)

DID Auth with JWT Attributes and OIDC Session Management

As a format for the DID Auth challenge-response cycle, DID Auth could re-use the well-specified JWT attributes used in OAuth2 and OIDC (e.g. `exp`, `iat`, and others) as well as the concept of OIDC request objects where an authentication request is expressed as a self-contained JWT.

In addition, although session management is not a focus of this document, DID Auth implementers might find the OIDC Session Management specification and the OIDC User Consent step helpful (in terms of establishing a long-lived session *after* the DID Auth ceremony).

References:

- [OIDC: Passing Request Parameters as JWTs](#)
- [IIW #26 Session Notes "Open ID v. FIDO v. SSI"](#)
- [IIW #26 Session Notes "DID Auth Workflows \(Part 2\)"](#)

DID Auth as Token Endpoint Authentication Method

As part of the OAuth2 / OIDC workflow, *relying party* clients must authenticate themselves to the authorization server (i.e. the token issuing endpoint) using a variety of methods. Since a PKI-based authentication method is already supported by OIDC (i.e. the `private_key_jwt` method), DID Auth can fulfill a role as a secure client-authentication mechanism during this step.

SECURITY AND PRIVACY CONSIDERATIONS

Support for Multiple DIDs

During a DID Auth interaction, a *relying party* may request that the *identity owner's* user agent display a DID selection dialog where the *identity owner* is able to pick the DID to authenticate with to the *relying party*. The DID Auth challenge and response are then done using that DID.

An *identity owner's* user agent may also automatically choose (or even dynamically create) different DIDs for each *relying party*. This idea is also known as directed identity. If a different DID is used for each *relying party*, then they are called pairwise-pseudonymous DIDs. This measure is helpful (but not always sufficient) to avoid correlation of the *identity owner*.

An *identity owner* may also choose to use different DIDs for the same *relying party* depending on context.

Automatic Authentication

Depending on the challenge and response transports, and depending on an *identity owner's* user agent, it may be possible to automate some steps in a DID Auth interaction. For example, a DID Auth browser extension can automatically complete a DID Auth interaction to log in to a website using a single click on a button.

Re-authentication and Step-up

Even after a DID Auth challenge-response cycle has been completed, a *relying party* may require an *identity owner* to re-authenticate in certain cases, e.g. when a high-value transaction such as transferring money is initiated.

In such cases, subsequent protocol interactions may also require the exchange of Verifiable Credentials in addition to basic DID Auth, in order to establish the desired level of trust.

DID Resolution

DID Auth depends on the ability to resolve a DID to its associated DID Document. Therefore all security considerations associated with DID resolution must be taken into account. Among other topics, this includes caching behavior of a DID resolver, as well as metadata about the DID resolution process (e.g. whether the DID Document has been retrieved via a blockchain full node or via an untrusted intermediary lookup service).

References:

- [DID Resolution v0.1](#)

Owner vs. Controller

This paper heavily uses the term *identity owner*. This helps to emphasize clearly how DID Auth is fundamentally different from earlier authentication protocols, which have traditionally revolved around "identity providers". Note however that the *identity owner* (the entity identified by the DID) may in some situations not be the same entity who controls the DID and who is able to perform authentication. This may happen in cases of security breaches (e.g. theft of a private key that controls the DID), or in intentional situations such as digital guardianship, where for some reason the *identity owner* is not able to control their own DID.

Single Log-out

DID Auth should provide ways to end all sessions and relationships that have been established for an authenticated DID. An *identity owner* should be able to trigger single log-out manually using their user agent. Single log-out should also happen automatically if a DID is revoked, which means that *relying parties* must monitor authenticated DIDs for revocation.

Hardware Wallet

The DID Auth challenge may be processed by a hardware wallet that upon an *identity owner's* physical interaction will create a response and send it back to the *relying party*.

DID Auth gateway services

In order to ease adoption of DID Auth through the use of standard protocols, gateway services could be developed that expose the OpenID Connect and/or OAuth2 protocols and perform (parts of) a DID Auth *relying party*. In such cases, the privacy and trust implications of the use of a gateway service must be carefully considered.

ADDITIONAL CREDITS

Lead Author: Markus Sabadello

Authors: Kyle Den Hartog, Christian Lundkvist, Cedric Franz, Alberto Elias, Andrew Hughes, John Jordan, Dmitri Zagidulin

Contributors: Eugeniu Rusu, Adam Powers, John Callahan, Joe Andrieu

About Rebooting the Web of Trust

This paper was produced as part of the [Rebooting the Web of Trust VI](#) design workshop. On March 6th to 8th, 2018, over 40 tech visionaries came together in Santa Barbara, California to talk about the future of decentralized trust on the internet with the goal of writing 3-5 white papers and specs. This is one of them.

Named Sponsors List: Sovrin Foundation, PTB Holdings

Workshop Credits: Christopher Allen (Founder), Joe Andrieu, PMP (Producer and Facilitator), Shannon Appelcline (Editor-in-chief), Erica Connell (Event Coordinator), Claire Rumore (Graphical Recorder), and The Narrative Loft (Venue)

RWOT Leadership Team: Christopher Allen, Joe Andrieu, Kim Hamilton Duffy, Manu Sporny, and Heather Vescent

Thanks to our other contributors and sponsors!

What's Next?

The design workshop and this paper are just starting points for Rebooting the Web of Trust. If you have any comments, thoughts, or expansions on this paper, please post them to our GitHub issues page:

<https://github.com/WebOfTrustInfo/rebooting-the-web-of-trust-spring2018/issues>

The next Rebooting the Web of Trust design workshop is scheduled for the week of September 24th, 2018 in the Toronto area. If you'd like to be involved or would like to help sponsor these events, email:

rwot-leadership@googlegroups.com
