DID-based Auth Protocols

Vurucu ve Akillica Alt Başlık

Abdulhamit Kumru

Blokzincir Laboratuvarı

2020

License Attribution-ShareAlike 4.0 International

Agenda

DID Auth Development

- DID TLS (Agust 2017) *unresolved*
  - TLS mutual Auth with DIDs
- RWOT 6 DID Auth (March 2018)
  - Introduction to DID Auth

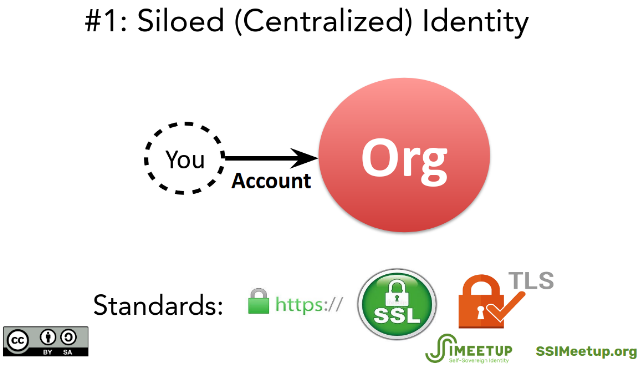
DID Auth Development

- DID Authentication WG (Jan 14 2020)
  - did-auth-jose (October 2018, DIF) *archived sept 22*
  - DID SIOP (Sep 12, 2019, DIF) *Current focus*
    - successor of did-auth-jose

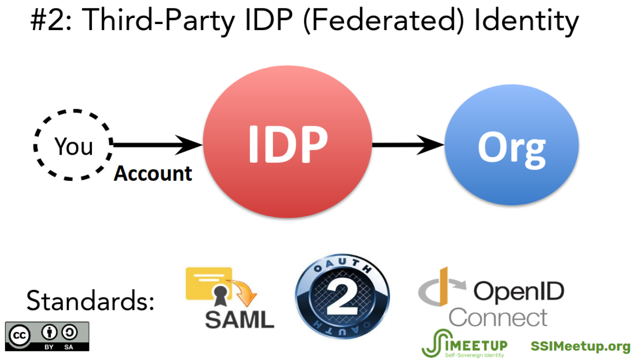
Auth Protocols & DID Auth

- OAuth & OpenID Connect
- Single Sign On
- Self-Issued OpenID Connect Provider DID Profile (did-siop, DIF)

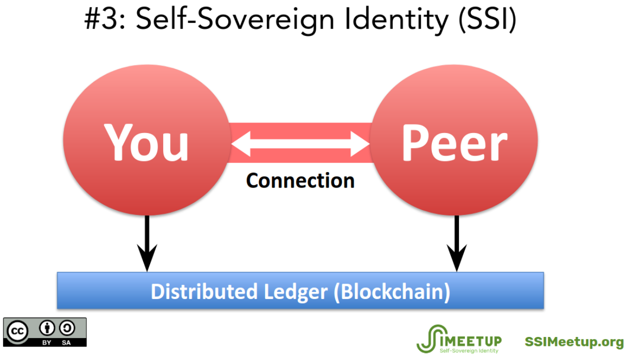
Centralised ID



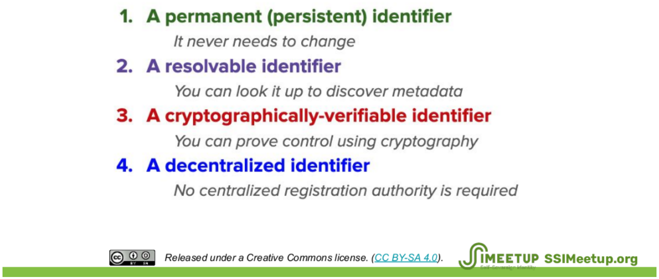
Federated ID


#2: Third-Party IDP (Federated) Identity
You
Account
IDP
Org
Standards:
SAML
OAUTH 2
OpenID Connect
SIMEETUP Self-Sovereign Identity
SSIMeetup.org
CC BY SA

Self-Sovereign Identity (SSI)



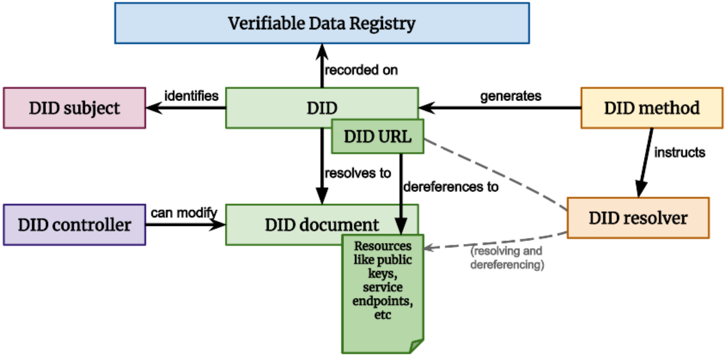
Four Core Properities of DID



1. **A permanent (persistent) identifier**
   It never needs to change
2. **A resolvable identifier**
   You can look it up to discover metadata
3. **A cryptographically-verifiable identifier**
   You can prove control using cryptography
4. **A decentralized identifier**
   No centralized registration authority is required

Released under a Creative Commons license. (CC BY-SA 4.0). SIMEETUP SSIMeetup.org

Architecture Overview



DIDs and DID URLs

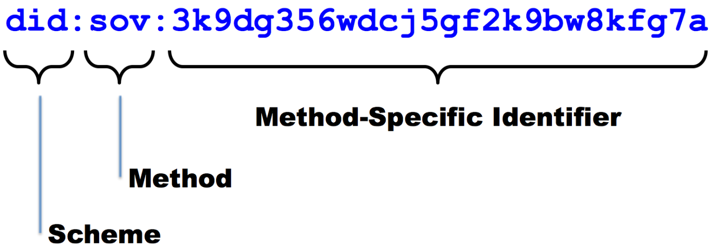
Architecture Overwiew

A DID, or Decentralized Identifier, is a URI composed of three parts: ***the scheme*** “did:”, a ***method identifier***, and a unique, ***method-specific identifier*** generated by the DID method.

DIDs are resolvable to DID documents. A DID URL extends the syntax of a basic DID to incorporate other standard URI components (path, query, fragment) in order to locate a particular resource.

DID Format

Architecture Overwiew


did:sov:3k9dg356wdcj5gf2k9bw8kfg7a
Method-Specific Identifier
Method
Scheme

DID Subjects

Architecture Overwiew

The subject of a DID is, by definition, the entity identified by the DID. The DID subject may also be the DID controller. Anything can be the subject of a DID: person, group, organization, physical thing, logical thing, etc.

DID Controllers

Architecture Overwiew

The controller of a DID is the entity (person, organization, or autonomous software) that has the capability—as defined by a DID method—to make changes to a DID document. This capability is typically asserted by the control of a set of cryptographic keys used by software acting on behalf of the controller, though it may also be asserted via other mechanisms. Note that a DID may have more than one controller, and the DID subject can be the DID controller, or one of them.

Verifiable Data Registries

Architecture Overwiew

In order to be resolvable to DID documents, DIDs are typically recorded on an underlying system or network of some kind. Regardless of the specific technology used, any such system that supports recording DIDs and returning data necessary to produce DID documents is called a verifiable data registry. Examples include distributed ledgers, decentralized file systems, databases of any kind, peer-to-peer networks, and other forms of trusted data storage.

DID documents

Architecture Overwiew

DID documents contain metadata associated with a DID. They typically express verification methods (such as public keys) and services relevant to interactions with the DID subject.

Minimal Self-managed DID document Example

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCw..."
  }],
  "service": [{
    "id":"did:example:123456789abcdefghi#vcs",
    "type": "VerifiableCredentialService",
    "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

DID Methods

Architecture Overwiew

DID methods are the mechanism by which a particular type of DID and its associated DID document are created, resolved, updated, and deactivated using a particular verifiable data registry. DID methods are defined using separate DID method specifications.

DID resolvers and DID resolution

Architecture Overwiew

A DID resolver is a software and/or hardware component that takes a DID (and associated input metadata) as input and produces a conforming DID document (and associated metadata) as output. This process is called DID resolution.

detailed spec w3c-ccg.github.io/did-resolution/

DID Resolve Example

```
did:example:1234;version-id=4#keys-1 # resolves to

{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi#keys-1",
  "type": "RsaVerificationKey2018",
  "publicKeyPem": "-----BEGIN PUB...0101010..END PUB -----\r\n"
}
```

Identifier

This section describes the formal syntax for DIDs and DID URLs. The term “generic” is used to differentiate the syntax defined here from syntax defined by specific DID methods in their respective specifications.

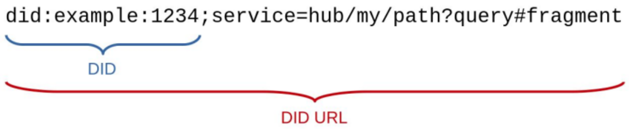
DID Syntax

- The generic DID scheme is a URI scheme conformant with [RFC3988].
- The DID scheme and method name **MUST** be an ASCII lowercase string.

```
# Ethr-DID
did:ethr:0xb9c5714089478a327f09197987f16f9e5d936e8a
```

A DID is expected to be persistent and immutable. That is, a DID is bound exclusively and permanently to its one and only subject. Even after a DID is deactivated, it is intended that it never be repurposed.

DID URL Syntax



```
did-url = did path-abempty [ "?" query ] [ "#" fragment ]
```

DID Parameters

DID URL Syntax

The DID URL syntax supports a simple format for parameters based on the query component. Adding a DID parameter to a DID URL means that the parameter becomes part of the identifier for a resource.

DID Parameters

DID URL Syntax

Relative Reference

A relative URI reference according to RFC3986 Section 4.2 that identifies a resource at a service endpoint, which is selected from a DID document by using the service parameter. Support for this parameter is **REQUIERED**

Relative Reference Example

DID URL Syntax

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:1234",
  "verificationMethod": [{
    "id": "did:example:1234#key-1",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:1234",
    "publicKeyBase58": "H3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwn..."
  }, ...],
  "authentication": [
    // relative DID URL to ‘did:example:1234#key-1
    "#key-1"
  ]
}
```

DID Parameters

DID URL Syntax

service parameter

Identifies a service from the DID document by service ID. Support for this parameter is **REQUIRED**

```
did:foo:21tDAKCERh95uGgKbJNHYp?service=agent
```

DID Parameters

DID URL Syntax

version-id parameter

Identifies a specific version of a DID document to be resolved (the version ID could be sequential, or a UUID, or method-specific). Support for this parameter is **OPTIONAL**

DID Parameters

DID URL Syntax

version-time parameter

Identifies a certain version timestamp of a DID document to be resolved. Support for this parameter is **OPTIONAL**

```
did:foo:21tDKJNHYp?version-time=2002-10-10T17:00:00Z
```

DID Parameters

DID URL Syntax

hl patameter

A resource hash of the DID document to add integrity protection, as specified in Hashlink RFC. This parameter is *non-normative*

```
 url encoded hash link
hl:zm9YZpCjPLPJ4Epc:z3TSgXTuaHxY2ts...7DYuQ9QTPQyLHy
```

DID URL Syntax

Path

A DID path is identical to a generic URI path

```
did:example:123456/path
```

Query

A DID query is derived from a generic URI query and **MUST** conform to DID URL Syntax rules.
If a DID query is present, it **MUST** be used with DID Parameters.

```
did:example:123456?query=true
```

DID URL Syntax

Fragment

A DID fragment is used as method-independent reference into a DID document or external resource. DID fragment syntax and semantics are identical to a generic URI fragment and **MUST** conform to RFC 3986

```
did:example:123#agent # service endpoint
did:example:123#public-key-0 # verification method
```

Relative DID URLs

A relative DID URL is any URL value in a DID document that does not start with `did:<method-name>:<method-specific-id>`.

```
// ... relative DID URL to ‘did:example:1234#key-1‘
"authentication": [ "#key-1" ]
// ...
```

Example DID URLs

```
# A DID URL with a 'service' DID parameter
did:foo:21tDAKCERh95uGgKbJNHYp?service=agent
# A DID URL with a 'version-time' DID parameter
did:foo:21tD...gKbJNHYp?version-time=2002-10-10T17:00:00Z

did:example:1234/
did:example:1234#keys-1
did:example:1234;version-id=4#keys-1
did:example:1234/my/path?query#fragment
did:example:1234;service=hub/my/path?query#fragment
```

Core Properties

- id
- authentication
- controller
- service
- verificationMethod
- assertionMethod
- keyAgreement
- capabilityDelegation
- capabilityInvocation

id Property

DID Subject

The DID subject is denoted with the ***id*** property at the top level of a DID document.

- The DID subject is the entity that the DID document is about
- DID documents **MUST** include the id property at the top level.

```
{
  "id": "did:example:21tDAKCERh95uGgKbJNHYp"
}
```

alsoKnownAs

- A DID subject can have *multiple identifiers* for different purposes, or at different times.
- The assertion that two or more DIDs (or other types of URI) identify the same DID subject can be made using the ***alsoKnownAs*** property.

Control

Authorization is the mechanism used to state how operations are performed on ***behalf*** of the DID subject. ***A DID controller is authorized*** to make changes to the respective DID document.

Note: Authorization vs Authentication !

DID Document With a Controller Property

```
{
  "@context": "https://www.w3.org/ns/did/v1",
  "id": "did:example:123456789abcdefghi",
  "controller": "did:example:bcehfew7h32f32h7af3",
  "service": [{

    "type": "VerifiableCredentialService",
    "serviceEndpoint": "https://example.com/vc/"
  }]
}
```

Verification Methods

A DID document can express verification methods, such as cryptographic keys, which can be used to authenticate or authorize interactions with the DID subject or associated parties. A DID document MAY include a verificationMethod property.

- The information expressed often includes globally unambiguous identifiers and public key material, which can be used to verify digital signatures.
- Verification methods might take many parameters. An example of this is a set of five cryptographic keys from which any three are required to contribute to a threshold signature.
- In order to maximize interoperability, support for public keys as verification methods is restricted

verificationMethod Property

Verification Methods

- The properties **MUST** include the ***id, type, controller, and specific verification method properties*** , and MAY include additional properties.
- The value of the ***id*** property for a verification method **MUST be a URI**.

Note: Verification method controller(s) and DID controller(s)

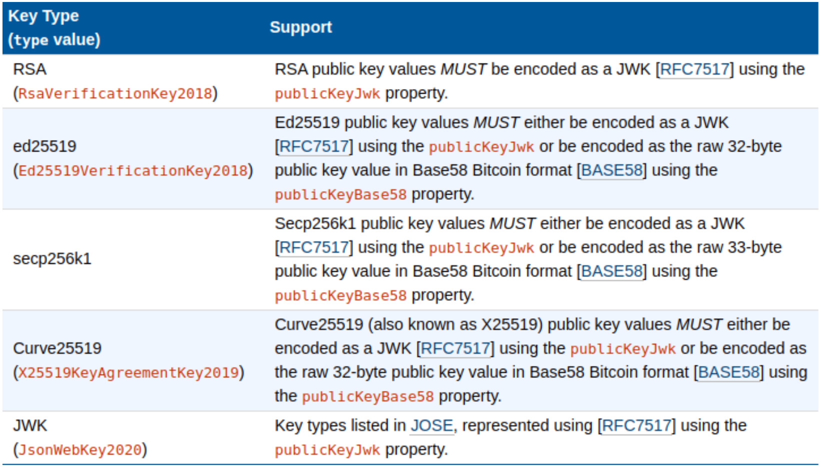
As well as the ***verificationMethod*** property, verification methods can be embedded in or referenced from properties associated with various verification relationships

Embedding and referencing verification methods

```
{ ...  "authentication": [
  // this key is referenced
  it may be used with more than one verification relationship
  "did:example:123456789abcdefghi#keys-1",
  // this key is embedded
  and may *only* be used for authentication
  {
    "id": "did:example:123456789abcdefghi#keys-2",
    "type": "Ed25519VerificationKey2018",
    "controller": "did:example:123456789abcdefghi",
    "publicKeyBase58": "H3C2AV...z3wXmqPV"
  }
], ... }
```

Key types and formats

Verification Methods



Key Type (type value)	Support
RSA (`RsaVerificationKey2018`)	RSA public key values *MUST* be encoded as a JWK [RFC7517] using the `publicKeyJwk` property.
ed25519 (`Ed25519VerificationKey2018`)	Ed25519 public key values *MUST* either be encoded as a JWK [RFC7517] using the `publicKeyJwk` or be encoded as the raw 32-byte public key value in Base58 Bitcoin format [BASE58] using the `publicKeyBase58` property.
secp256k1	Secp256k1 public key values *MUST* either be encoded as a JWK [RFC7517] using the `publicKeyJwk` or be encoded as the raw 33-byte public key value in Base58 Bitcoin format [BASE58] using the `publicKeyBase58` property.
Curve25519 (`X25519KeyAgreementKey2019`)	Curve25519 (also known as X25519) public key values *MUST* either be encoded as a JWK [RFC7517] using the `publicKeyJwk` or be encoded as the raw 32-byte public key value in Base58 Bitcoin format [BASE58] using the `publicKeyBase58` property.
JWK (`JsonWebKey2020`)	Key types listed in JOSE, represented using [RFC7517] using the `publicKeyJwk` property.

Verification Relationships

A verification relationship expresses the relationship between the DID subject and a verification method.

Different verification relationships enable the associated verification methods to be used for different purposes

Verification Relationships

Authentication

The ***authentication*** verification relationship is used to specify how the DID subject is expected to be authenticated, such as for the purposes of logging into a website

Assertion

The ***assertionMethod*** verification relationship is used to specify how the DID subject is expected to express claims, such as for the purposes of issuing a Verifiable Credential

Key Agreement

The ***keyAgreement*** verification relationship is used to specify how to encrypt information to the DID subject, such as for the purposes of establishing a secure communication channel with the recipient

Verification Relationships

Capacity Invocation

The ***capabilityInvocation*** verification relationship is used to specify a mechanism that might be used by the DID subject to invoke a cryptographic capability, such as the authorization to access an HTTP API

Capacity Delegation

The ***capabilityDelegation*** verification relationship is used to specify a mechanism that might be used by the DID subject to delegate a cryptographic capability to another party, such as delegating the authority to access a specific HTTP API to a subordinate

Service Endpoints

Service endpoints are used in DID documents to express ways of communicating with the DID subject or associated entities. Services listed in the DID document can contain information about privacy preserving messaging services, or more public information, such as social media accounts, personal websites, and email addresses although this is discouraged

One of the primary purposes of a DID document is to enable discovery of service endpoints. A service endpoint can be any type of service the DID subject wants to advertise, including decentralized identity management services for further discovery, authentication, authorization, or interaction

Various service endpoints

```
{ // ...
  "service": [{
    "id": "did:example:123456789abcdefghi#openid",
    "type": "OpenIdConnectVersion1.0Service",
    "serviceEndpoint": "https://openid.example.com/"
  }, {
    "id": "did:example:123456789abcdefghi#vcr",
    "type": "CredentialRepositoryService",
    "serviceEndpoint": "https://repository.example.com/service/8
  }, {
    "id": "did:example:123456789abcdefghi#xdi",
    "type": "XdiService",
    "serviceEndpoint": "https://xdi.example.com/8377464"
  }]
// ... }
```

Core Representations

All concrete representations of a DID document are serialized using a deterministic mapping that is able to be unambiguously parsed into the data model defined in this specification

Producers MUST indicate which representation of a document has been used via a media type in the document's metadata. Consumers MUST determine the representation of a DID document via the content-type DID resolver metadata field (see § 8.1 DID Resolution), not through the content of the DID document alone.

- JSON
- JSON-LD
- Concise Binary Object Representation (CBOR)

Methods

DID methods provide the means to implement did core specification on different *verifiable data registries*.

- The DID method specification **MUST** specify how to generate the *method-specific-id* component of a DID.
- The *method-specific-id* value **MUST** be able to be generated without the use of a centralized registry service.
- Each DID method **MUST** define how authorization is implemented, including any necessary cryptographic operations.

Note: Unique DID method names

Method Operations

Create

The DID method specification **MUST** specify how a DID controller creates a DID and its associated DID document on the verifiable data registry, including all cryptographic operations necessary to establish proof of control.

Read/Verify

The DID method specification **MUST** specify how a DID resolver uses a DID to request a DID document from the verifiable data registry, including how the DID resolver can verify the authenticity of the response.

Method Operations

Update

The DID method specification **MUST** specify how a DID controller can update a DID document on the verifiable data registry, including all cryptographic operations necessary to establish proof of control, or state that updates are not possible

Deactivate

The DID method specification **MUST** specify how a DID controller can deactivate a DID on the verifiable data registry, including all cryptographic operations necessary to establish proof of deactivation, or state that deactivation is not possible.
Note: Check Out Method Security & Privacy Requirements

DID Resolution

The DID resolution functions resolve a DID into a DID document by using the “Read” operation of the applicable DID method.

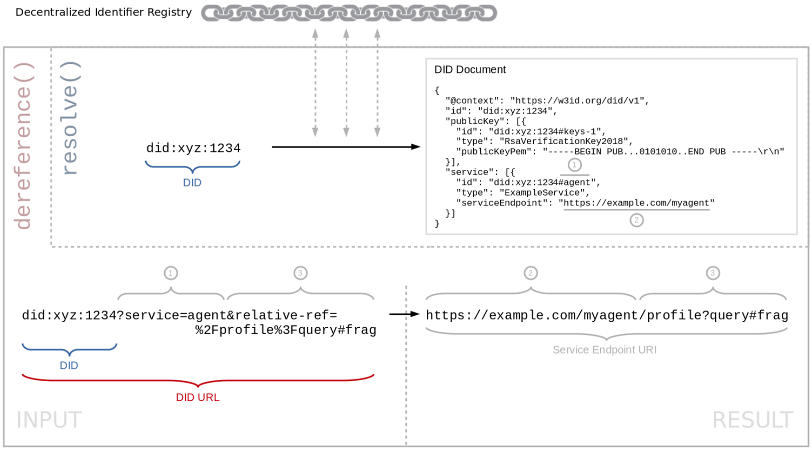
```
resolve ( did, did-resolution-input-metadata )
     -> ( did-resolution-metadata, did-document,
     did-document-metadata )
```

DID URL Dereferencing

The DID URL dereferencing function dereferences a DID URL into a resource with contents depending on the DID URL's components, including the DID method, method-specific identifier, path, query, and fragment

```
dereference ( did-url, did-url-dereferencing-input-metadata )
         -> ( did-url-dereferencing-metadata,
         content-stream, content-metadata )
```

DID URL Dereferencing


Decentralized Identifier Registry
dereference()
resolve()
did:xyz:1234
DID
DID Document
1
2
1
3
2
3
did:xyz:1234?service=agent&relative-ref=
%2Fprofile%3Fquery#frag
DID
DID URL
https://example.com/myagent/profile?query#frag
Service Endpoint URI
INPUT
RESULT

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:xyz:1234",
  "publicKey": [{
    "id": "did:xyz:1234#keys-1",
    "type": "RsaVerificationKey2018",
    "publicKeyPem": "-----BEGIN PUB...0101010..END PUB -----\r\n"
  }],
  "service": [{
    "id": "did:xyz:1234#agent",
    "type": "ExampleService",
    "serviceEndpoint": "https://example.com/myagent"
  }]
}
```

Implementations

- dif/did-common-java

DID Auth Development

- DID TLS (Agust 2017) *unresolved*
  - TLS mutual Auth with DIDs
- RWOT 6 DID Auth (March 2018)
  - Introduction to DID Auth

DID Auth Development

- DID Authentication WG (Jan 14 2020)
  - did-auth-jose (October 2018, DIF) *archived sept 22*
  - DID SIOP (Sep 12, 2019, DIF) *Current focus*
    - successor of did-auth-jose

DID TLS (Agust 2017, Hyperledger) *unresolved*

DID TLS feature proposed as indy SDK feature.

- Using chain anchored keys to facilitate mutual authentication via TLS.
- Extends the TLS SNI Specification and provides additional methods for certificate validation that does not rely on established Root Certificate Authorities.

Links

- github.com/TelegramSam/DID-TLS
- jira.hyperledger.org/browse/IS-268

DID TLS (Agust 2017, Hyperledger) *unresolved*

Issues

- Setting custom SNI hint from client
- Inability to disable client certificate validation
- Lack of certificate validation callbacks

Future Work

- OpenSSL: Allow non-validated client certificates. (removes signer tls sni hint requirement.)
- OpenSSL: Add support for newer elliptic curve certificates.
- SNI Spec: Updated to use different server_type than HostName
- HTTP Libraries: Easier methods to specify SNI hint.

DID TLS (Agust 2017, Hyperledger) *unresolved*

- Current Agent2Agent communication in Aries Cloud Agent is Inbound and Outbound TCP ports with *DIDcomm*
- DIF Auth WG Open is currently focused on developing DID OpenID Connect Provider (did-siop)

Rebooting Web-of-Trust (RWoT) 6 DID Auth

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
Core idea: proving control of a DID to relying party

Owner vs. Controller

RWoT 6 DID Auth

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".

DID Authn and Verifiable Credentials

RWoT 6 DID Auth

- DID Auth and Verifiable Credentials exchange are separate.
- Verifiable Credentials exchange is an extension to (or part of) DID Auth.
- DID Auth is a certain kind of Verifiable Credential. bunu tamamla

Authentication of a DID

RWoT 6 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.

Challenge

RWoT 6 DID Auth

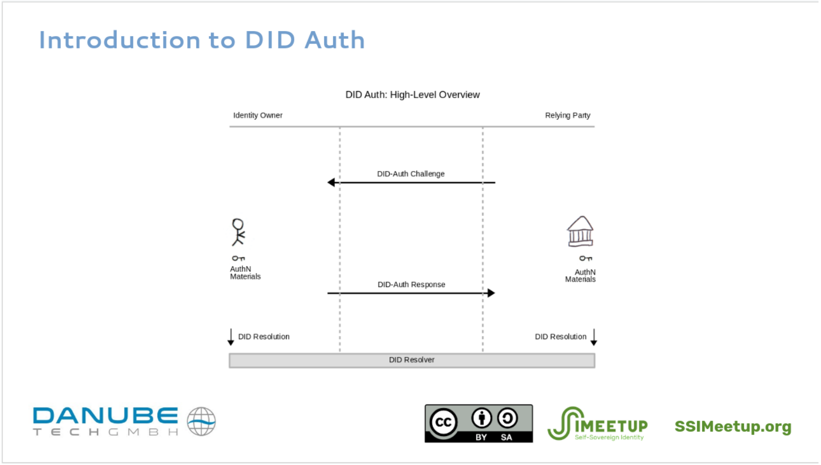
The way an identity owner or their agent encounters an authentication challenge, as well as the format of the challenge, will vary depending on the situation. For example, they can come across a “Sign in with DID Auth” button or a QR code on a website

Response

RWoT 6 DID Auth

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.)

Generic DID Auth Architecture


Introduction to DID Auth

Javascript Object Signing and Encryption (JOSE)

- Used in DID Auth and DID SIOP
- Adapted in many similar area

JSON Web Algorithms (JWA)

JWA specification registers cryptographic algorithms and identifiers to be used with the JSON Web Signature (JWS), JSON Web Encryption (JWE), and JSON Web Key (JWK) specifications

JSON Web Signature (JWS)

JSON Web Signature (JWS) represents content secured with digital signatures or Message Authentication Codes (MACs) using JSON-based data structures

Javascript Object Signing and Encryption (JOSE)

JSON Web Encryption (JWE)

JSON Web Encryption (JWE) represents encrypted content using JSON-based data structures

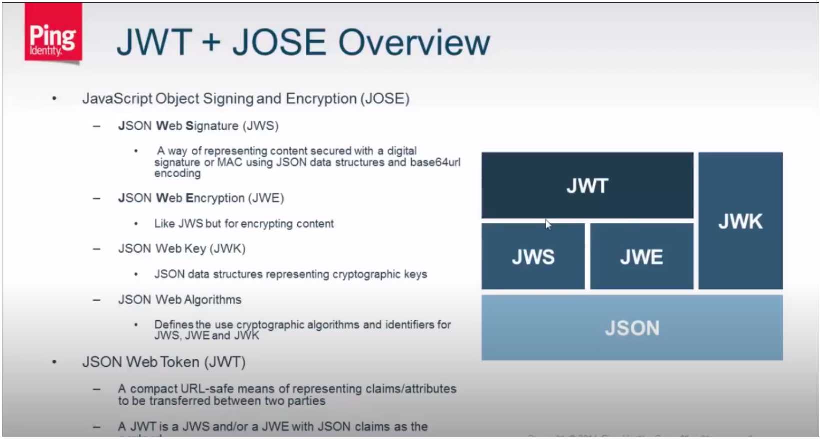
JSON Web Key (JWK)

A JSON Web Key (JWK) is a JavaScript Object Notation (JSON) data structure that represents a cryptographic key

JSON Web Token (JWT)

JSON Web Token (JWT) is a compact, URL-safe means of representing claims to be transferred between two parties.

Javascript Object Signing and Encryption (JOSE)



JWT + JOSE Overview

- JavaScript Object Signing and Encryption (JOSE)
  - JSON Web Signature (JWS)
    - A way of representing content secured with a digital signature or MAC using JSON data structures and base64url encoding
  - JSON Web Encryption (JWE)
    - Like JWS but for encrypting content
  - JSON Web Key (JWK)
    - JSON data structures representing cryptographic keys
  - JSON Web Algorithms
    - Defines the use cryptographic algorithms and identifiers for JWS, JWE and JWK
- JSON Web Token (JWT)
  - A compact URL-safe means of representing claims/attributes to be transferred between two parties
  - A JWT is a JWS and/or a JWE with JSON claims as the

DID Authn Challenge

RWoT 6 DID Auth

JWT challenge example, uPort

```
{ "typ": "JWT", "alg": "ES256K" }
{
  "iss": "2oeXufHGDpU51bfKBsZDdu7Je9weJ3r7sVG",
  "iat": 1525865398,
  "requested": [
    "name", // ...
  ],
  "permissions": [ "notifications" ],
  "callback": "https://.../api/v1/topic/Go...Bft7PZ9",
  "exp": 1525865998,
}
```

Challenge Transports

RWoT 6 DID Auth

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
- Custom Protocol Handler
- Mobile Deep Link
- Custom Protocol Handler
- Invoke User Agent's JavaScript API
- Form Redirect
- Device-to-device Communication

DID Authn Challenge

RWoT 6 DID Auth

JWT challenge resoponse example

```
{
  "header": {
    "typ": "JWT",
    "alg": "ES256"
  },
  "payload": {
    "iss": "did:example:123456789abcdefg",
    "sub": "did:example:123456789abcdefg",
    "iat": 1479850830,
    "exp": 1511305200,
  },
  "signature": "..."
}
```

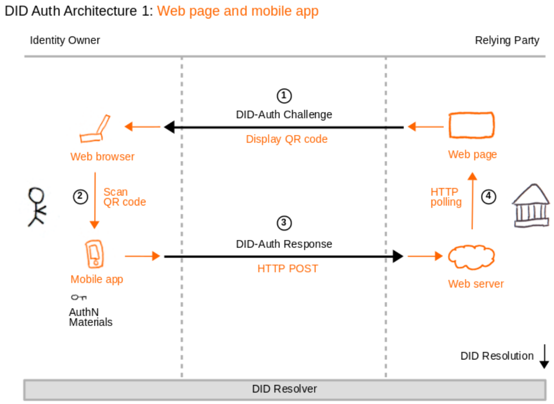
Response Transports

RWoT 6 DID Auth

- HTTP POST to Callback URL
- Scan QR Code from Mobile App
- Fulfill JavaScript Promise
- Device-to-device Communication

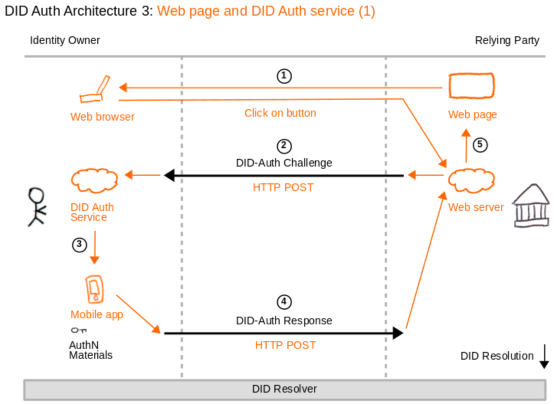
Auth Architecture Web page and mobile app

RWoT 6 DID Auth



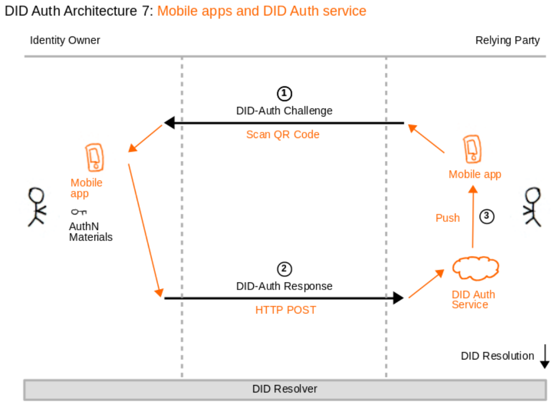
Auth Architecture Web page and DID Auth service

RWoT 6 DID Auth



Auth Architecture Mobile apps and DID Auth services

RWoT 6 DID Auth



Relation to Other Technologies

RWoT 6 DID Auth

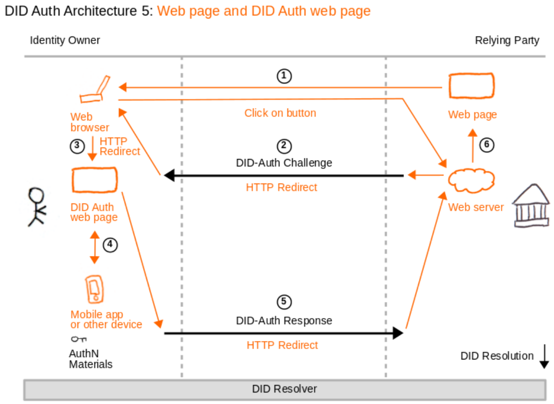
- Other Public Key Infrastructure (PGP, SSH, etc.)
- WebAuthn (FIDO Authentication)

OpenID Connect *(DIF Auth Current Focus)*

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.

Relation to Other Technologies

RWoT 6 DID Auth



Next Steps

RWoT 6 DID Auth

- Agree on scope of DID Auth.
- Agree on supported formats for challenges and responses.
- Agree on supported protocols and flows.

indy_auth

Internship Project, Application of DID Auth scenarios with DIDs using `did:sov` / Hyperledger Indy.

Gitlab Repo

bag.org.tr/proje/abdulhamit.kumru/indy_tls

Presentations

https://bag.org.tr/proje/abdulhamit.kumru/indy_tls/tree/master/presentations

indy_auth

indy_auth

did_tls

- Generic DID Auth over TCP using generic JSON Object
- DH Session key generated
- TLS Socket created.

Tools

- Python ssl
- indy-sdk

Other Works

- http signatures
- auth encryption

indy-sdk

indy_auth

Everything needed to build applications that interact with an Indy distributed identity ledger.

libindy

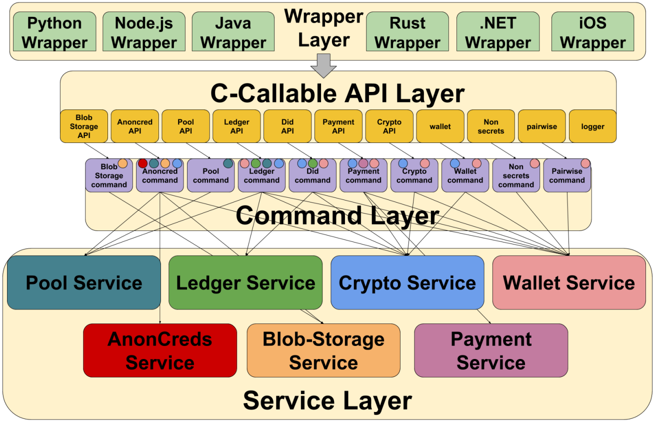
The major artifact of the SDK is a C-callable library that provides the basic building blocks for the creation of applications on the top of Hyperledger Indy

Libindy wrappers

A set of libindy wrappers for developing Indy-based applications in your favorite programming language

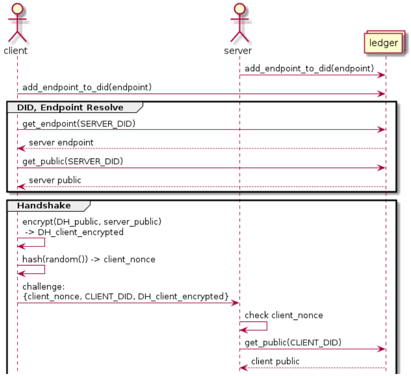
indy-sdk overview

indy_auth



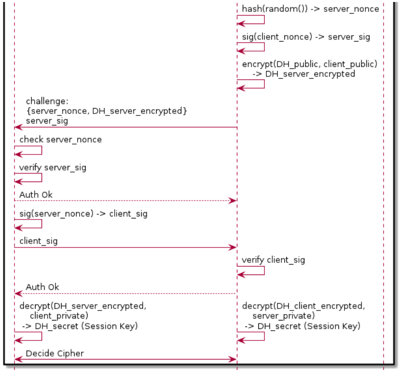
did_tls

indy_auth



did_tls

indy_auth


hash(random()) -> server_nonce
sig(client_nonce) -> server_sig
encrypt(DH_public, client_public) -> DH_server_encrypted
challenge: {server_nonce, DH_server_encrypted} server_sig
check server_nonce
verify server_sig
Auth Ok
sig(server_nonce) -> client_sig
client_sig
verify client_sig
Auth Ok
decrypt(DH_server_encrypted, client_private) -> DH_secret (Session Key)
decrypt(DH_client_encrypted, server_private) -> DH_secret (Session Key)
Decide Cipher

Decentralized Identity Foundation (DIF)

Provide a neutral and inclusive place for the decentralized identity community to collaborate

Working Groups

- Authentication
- Claims and Credentials
- DID Communication
- ...

did-auth-jose (October 2018, DIF) *archived sept 22*

did-auth-jose is a library that provides JOSE (Javascript Object Signing and Encryption) encryption, decryption, signing, and verifying capabilities through a key and algorithm extensible model, as well as two authentication flows for use with decentralized identities (DIDs).

evolved to did-siop

Authentication Flow

did-auth-jose

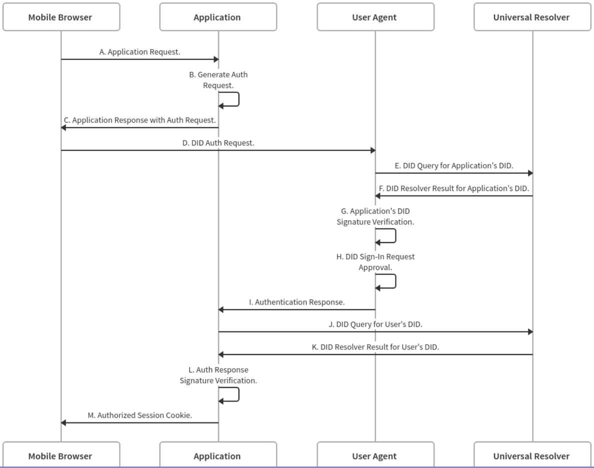
DID Authentication uses two to three JSON Web Tokens (JWT) per request.

The first is an outer JSON Web Encryption (JWE), and the second is an inner JSON Web Signature (JWS), both utilizing the public private key pair of each DID retrieved from their DID Document.

An optional third JWT access token may be included in a JWS header. This format ensures the content is encrypted end to end and independently verifiable. Each JWS contains a nonce header to associate requests with responses

Authentication Flow

did-auth-jose


Mobile Browser
Application
User Agent
Universal Resolver
A. Application Request.
B. Generate Auth Request.
C. Application Response with Auth Request.
D. DID Auth Request.
E. DID Query for Application's DID.
F. DID Resolver Result for Application's DID.
G. Application's DID Signature Verification.
H. DID Sign-In Request Approval.
I. Authentication Response.
J. DID Query for User's DID.
K. DID Resolver Result for User's DID.
L. Auth Response Signature Verification.
M. Authorized Session Cookie.
Mobile Browser
Application
User Agent
Universal Resolver

DIF Authentication Working Group (Jan 14 2020)

DID Authentication Profile for SIOP

SIOP DID AuthN flavor to use OpenID Connect (OIDC) together with the strong decentralization, privacy and security guarantees of DID for everyone who wants to have a generic way to integrate SSI wallets into their web applications.

- Status: DRAFT, WIP
- Use Case: Use your identity wallet to authenticate against a Web Application

SIOP DID is an unapproved DIF working group draft specification being developed within the Decentralized Identity Foundation (DIF).

DIF Authentication Working Group (Jan 14 2020)

Encrypted Envelope

This concept is borrowed from the HL Aries project to create a standardized means of authenticated general message passing between DID controllers. DIF provides an implementation of pack/unpack that intends to meet the requirements of the DIF community.

- Status: PROPOSAL
- Use Case: Secure communication between DID controllers.

decentralized-identity/DIDComm-js

Javascript version of the cryptographic envelope of DIDComm (pack, unpack).

Auth Protocols & DID Auth

- OAuth & OpenID Connect
- Single Sign On
- Self-Issued OpenID Connect Provider DID Profile (did-siop, DIF)

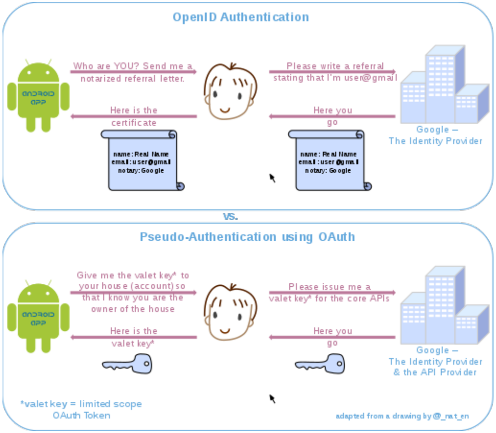
OAuth

OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords

OAuth

- OAuth is an authorization protocol, rather than an authentication protocol
- OAuth is directly related to OpenID Connect (OIDC), since OIDC is an authentication layer built on top of OAuth 2.0
- OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner

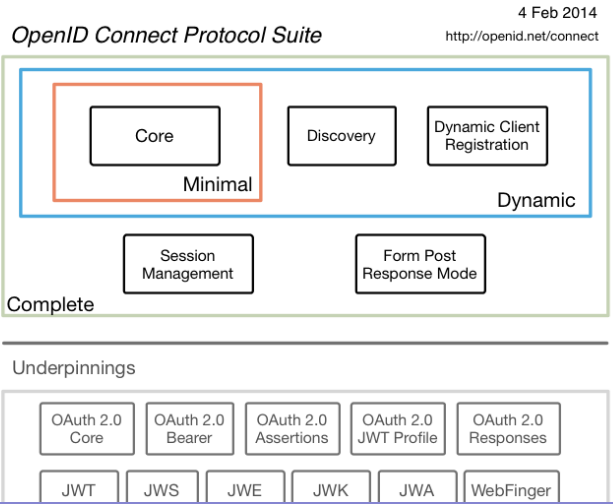
OAuth
OpenID vs OAuth


OpenID Authentication
Who are YOU? Send me a notarized referral letter.
Here is the certificate
name: Real Name email: user@gmail notary: Google
Please write a referral stating that I'm user@gmail
Here you go
name: Real Name email: user@gmail notary: Google
Google – The Identity Provider
vs.
Pseudo-Authentication using OAuth
Give me the valet key* to your house (account) so that I know you are the owner of the house
Here is the valet key*
Please issue me a valet key* for the core APIs
Here you go
Google – The Identity Provider & the API Provider
*valet key = limited scope OAuth Token
adapted from a drawing by @_nat_en

OpenID Connect

- *OpenID Connect 1.0 is a simple identity layer on top of the OAuth 2.0 protocol.* It allows Clients to verify the identity of the End-User based on the authentication performed by an Authorization Server, as well as to obtain basic profile information about the End-User in an interoperable and REST-like manner
- In technical terms, OpenID Connect specifies a RESTful HTTP API, using JSON as a data format
- Based on OAuth 2.0, REST, JSON, JWT, JOSE

OpenID Connect



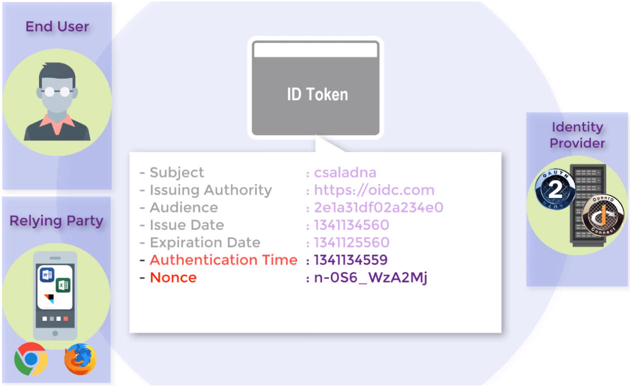
OpenID Connect

OpenID provider (OP)

An identity provider, or OpenID provider (OP) is a service that specializes in registering OpenID URLs or XRIs. OpenID enables an end-user to communicate with a relying party

ID Token

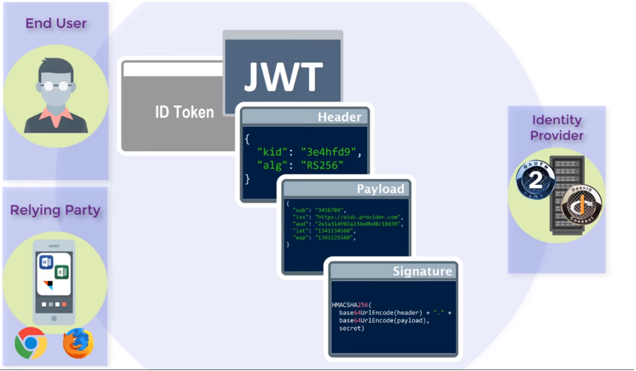
OpenID Connect



- Subject	: csaladna
- Issuing Authority	: https://oidc.com
- Audience	: 2e1a31df02a234e0
- Issue Date	: 1341134560
- Expiration Date	: 1341125560
- Authentication Time	: 1341134559
- Nonce	: n-0S6_WzA2Mj

JWT Token

OpenID Connect



```
{
  "kid": "3e4hfd9",
  "alg": "RS256"
}
```

```
{
  "sub": "3456789",
  "iss": "https://oidc.provider.com",
  "aud": "2e1a31df02a234e0bd8c18d39",
  "iat": "1341134560",
  "exp": "1341125560",
}
```

```
HMACSHA256(
  base64UrlEncode(header) + "." +
  base64UrlEncode(payload),
  secret)
```

OpenID Connect

ID Token

```
{
    "iss": "https://self-issued.me",
    "nonce": "n-0S6_WzA2Mj",
    "exp": 1311281970,
    "iat": 1311280970,
    "sub_jwk" : {
        "crv":"secp256k1",
        "kid":"did:example:0xcd#verikey-1",
        "kty":"EC",
        "x":"7KEKZa5xJPh7WVqHJyUpb2MgEe3nA8Rk7eUlXsmBl-M",
        "y":"3zIgl_ml4RhapyEm5J7lvU-4f5jiBvZr4KgxUjEhl9o"
    },
    "sub": "9-aYUQ7mgL2SWQ_LNTeVN2rtw7xFP-3Y2EO9WV22cF0",
    "did": "did:example:0xcd"
}
```

OpenID Connect

Security

- Registration between RP and OP is mandatory, can be done with public metadata exchange and selfregistration
- JSON messages can be signed and/or encrypted with the help of asymmetric keys (public keys published in JWKS) or symmetric keys (client secret)

Single Sign On

Single sign-on (SSO) is an authentication scheme that allows a user to log in with a single ID and password to any of several related, yet independent, software systems.

Web Single Sign On protocols

- Based on the principle of an authentication server, a lot of SSO standards have been created:
  - CoSign (Weblogin), Pubcookie, Webauth, CAH, CAS, WebID, BrowserID (Persona), SAML, WS-*, Liberty alliance, SAML 2, Shibboleth, OpenID, OpenID Connect...
- But nowadays, only a few are really used:
  - CAS, SAML 2, OpenID Connect

CAS

The Central Authentication Service (CAS) is a single *sign-on* protocol for the web.

Its purpose is to permit a user to access multiple applications while providing their credentials (such as userid and password) only once. It also allows web applications to authenticate users without gaining access to a user's security credentials, such as a password.

Security

- No obligation to declare CAS clients in CAS server (open mode)
- Trust between CAS client and CAS server relies on CAS server certificate validation

SAML

Security Assertion Markup Language is an open standard for exchanging authentication and authorization data between parties, in particular, between an identity provider and a service provider

- An important use case that SAML addresses is web-browser single sign-on

Security

- Registration between SP and IDP is mandatory, can be done with public metadata exchange
- XML messages can be signed and/or encrypted with the help of asymmetric keys (public keys published in metadata)

Protocol Comparison

- CAS: simple protocol, no strong security, fits internal usage
- SAML: complex protocol, very used for SaaS authentication, good security, well established
- ***OpenID Connect:*** easy adoption with new technologies (JSON/REST/OAuth2), mobile ready, good security, still not wide spread

self-issued openid connect provider

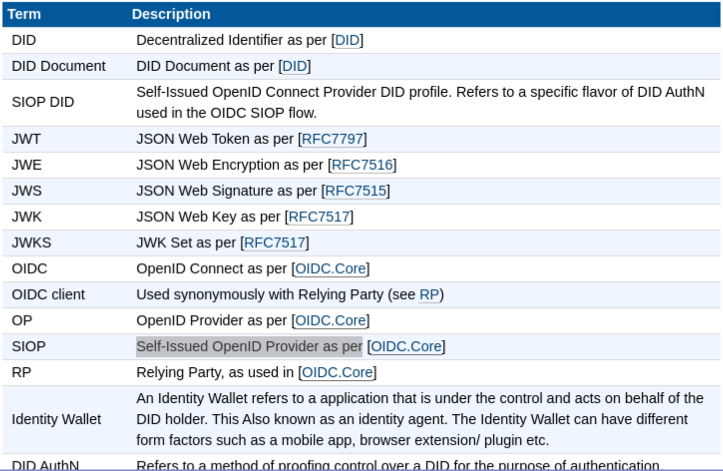
- A normal provider such as Google, is available at an HTTP endpoint. Requests to normal providers use the http:// protocol.
- A self-issued provider is usually installed on the end-user's device. Requests to self-issued providers use the openid:// protocol.

self-issued openid connect provider DID Profile (did-siop, DIF)

The work on DIF SIOP DID Profile specification has moved to OIDF AB WG to work on a new SIOP v2 specification that will either introduce breaking changes to the DIF SIOP DID Profile specification or will replace it with an implementation guide document on how to use SIOP v2 in an SSI context.

DID SIOP Terminology

DID SIOP



Term	Description
DID	Decentralized Identifier as per [DID]
DID Document	DID Document as per [DID]
SIOP DID	Self-Issued OpenID Connect Provider DID profile. Refers to a specific flavor of DID AuthN used in the OIDC SIOP flow.
JWT	JSON Web Token as per [RFC7797]
JWE	JSON Web Encryption as per [RFC7516]
JWS	JSON Web Signature as per [RFC7515]
JWK	JSON Web Key as per [RFC7517]
JWKS	JWK Set as per [RFC7517]
OIDC	OpenID Connect as per [OIDC.Core]
OIDC client	Used synonymously with Relying Party (see RP)
OP	OpenID Provider as per [OIDC.Core]
SIOP	Self-Issued OpenID Provider as per [OIDC.Core]
RP	Relying Party, as used in [OIDC.Core]
Identity Wallet	An Identity Wallet refers to a application that is under the control and acts on behalf of the DID holder. This Also known as an identity agent. The Identity Wallet can have different form factors such as a mobile app, browser extension/ plugin etc.
DID AuthN	Refers to a method of proofing control over a DID for the purpose of authentication.

Introduction

DID SIOP

An everyday use case that the Decentralized Identity community identified is the sign-up or login with web applications. Nowadays, this is often achieved through social login schemes such as Google Sign-In. *While the Decentralized Identity community has serious concerns about social login, the underlying protocol, OIDC, does not have these flaws by design.* SIOP DID provides great potential by leveraging an Identity Wallet, e.g., as a smartphone app, on the web. This will increase and preserve the user's privacy by preventing third-parties from having the ability to track which web applications a user is interacting with.

Introduction

DID SIOP

Purpose

The main purpose is to sign up with/ login to an RP (Relaying Party), i.e., web application. It assumes the user operates a mobile or desktop browser or a browser-based app that can respond to SIOP requests according to this specification.

Goals

- Staying backward compatible with existing OIDC clients and OPs (OpenID Provider) that implement the SIOP specification which is part of the OIDC core specification as per [OIDC.Core] to reach a broader community.
- Adding validation rules for OIDC clients that have DID AuthN support to make full use of DIDs.
- Not relying on any intermediary such as a traditional centralized public or private OP while still being OIDC-compliant.

Protocol Flow

DID SIOP

- First, the user clicks on the sign up or login UX element. The RP will then generate the redirect to openid:// which will be handled by the SIOP.
- The SIOP will generate the based on the specific DID method that is supported. The will be signed and optionally encrypted and will be provided according to the requested response mode.

Protocol Flow

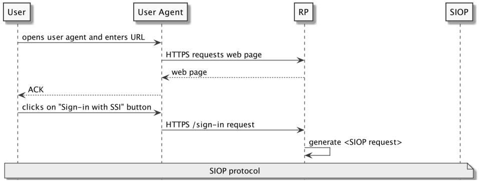
DID SIOP

- Unlike the OIDC Authorization Code Flow as per [OIDC.Core], the **SIOP will not return an access token to the RP**
- **SIOP also differs from Authorization Code Flow by not relying on a centralized and known OP**. The SIOP can be unknown to the RP until the user starts to interact with the RP using its Identity Wallet
- OIDC Authorization Code Flow is *still a useful approach* and *should be used whenever the OP is known, and OP discovery is possible, e.g.*, exchanged or pre-populated DID Document containing an openid element in the service section.
- *The SIOP flow allows to integrate Identity Wallets with plain OIDC clients if they implemented the SIOP specification*. In contrast, using DID AuthN as the authentication means in the OIDC Authorization Code Flow would *require integration with the OP vendor itself*.

Protocol Flow

DID SIOP

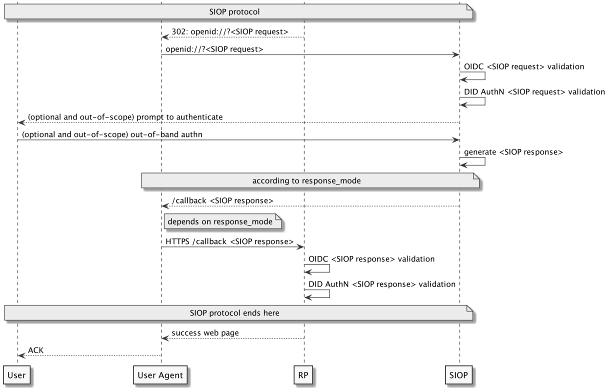
Example SIOP flow



Protocol Flow

DID SIOP

Example SIOP flow


SIOP protocol
302: openid://?<SIOP request>
openid://?<SIOP request>
OIDC <SIOP request> validation
DID AuthN <SIOP request> validation
(optional and out-of-scope) prompt to authenticate
(optional and out-of-scope) out-of-band authn
generate <SIOP response>
according to response_mode
/callback <SIOP response>
depends on response_mode
HTTPS /callback <SIOP response>
OIDC <SIOP response> validation
DID AuthN <SIOP response> validation
SIOP protocol ends here
success web page
ACK
User
User Agent
RP
SIOP