Corda DID Method

Unofficial Draft 21 September 2020

Editors

Abbas Ali (R3)
Arati Baliga (Persistent Systems)
Pandurang Kamat (Persistent Systems)
Moritz Platt (R3)

Authors

Pranav Kirtani (Persistent Systems)
Moritz Platt (R3)
Nitesh Solanki (Persistent Systems)

Abstract

The Corda DID method aims to implement Decentralized Identifier architecture in Corda networks. The Corda DID method is a hybrid method, exposing a widely recognized protocol for end users (JSON over HTTPS), while utilizing a Corda-specific means of communication (Corda's peer-to-peer messaging protocol) for data replication.

1. Introduction

To understand the environment in which the Corda DID method operates, the permissioned nature of a Corda network and the point-to-point approach to data replication must be taken into account. While parties in permissionless blockchains remain anonymous and can join and leave at will, any Corda network utilizes a standard PKIX infrastructure for linking public keys to identities. As such, individually deployed entities in the network – nodes – have a strong notion of identity. This concept is instrumental in network communication. Similarly, the data-replication model implemented in Corda is different to that of a conventional public blockchain, which makes all in-ledger data visible to all network participants. In Corda, data are distributed to a configurable subset of network members only.

2. Overall Architecture

The Corda DID method operates in an environment where multiple nodes form a consortium in order to replicate decentralized identity data (cf. figure 1). These consortium nodes replicate decentralized identifier documents to form a network-wide and, ultimately, consistent view of the unity of decentralized identifiers, using the Corda DID method.

Replication is implemented via the Corda peer-to-peer protocol and invoked via the Corda Flow Framework. Since replication between consortium members is not a public aspect of the DID method, it will not be discussed in detail in this specification. It is, however, briefly outlined in the information section.

External parties, i.e., those not part of the consortium, can utilize the Corda DID method to create, read, update and delete DID documents via two APIs:

1. The JSON-over-HTTPS protocol
2. The Corda peer-to-peer protocol

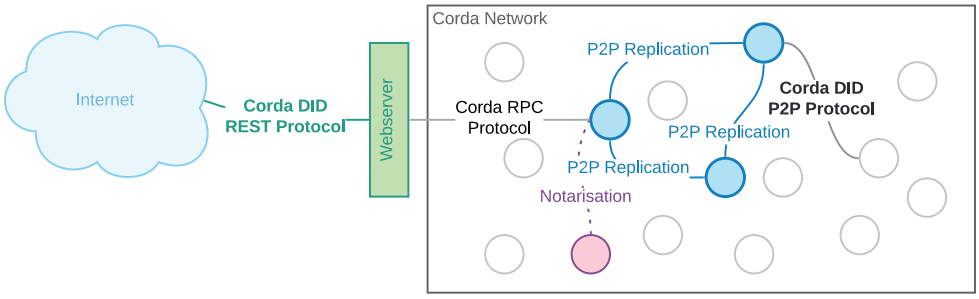

Figure 1: The high-level network architecture of a Corda network with three consortium nodes

At the time of writing, only the first of these (JSON over HTTPS) is a documented aspect of the DID method. The second protocol is intended for expert users who wish to make use of DID within the context of the Corda Flow Framework. A formal specification for this API will be the subject of future work.

2.1 Networks

DID documents are to be replicated between nodes in the same Corda network. Ultimately, the target network for the method is the Corda Network. While the method specification is evolving, an ephemeral environment for experimentation with the concept will be provided.

Network	Stage	Purpose	Consortium Members
Testnet	Test	To enable experimentation with the method in an ephemeral environment. DIDs are not expected to be continued permanently.	To be defined
Corda Network	Live	A production-ready environment replicating DID on a permanent basis.	To be defined

2.1.1 Member Node Endpoints

Consortium-member nodes have to maintain a permanent hostname or IP address for Corda peer-to-peer communication and to expose the HTTP server that operates the API. These hostnames/IP addresses can be different.

2.2 Security Considerations

Denial-of-Service (DoS) Attack

As replicating nodes are required to distribute data to others, they are susceptible to DoS attacks. An attacker who chooses to conduct a DoS attack can relatively easily flood the network with DID creation requests. If unmitigated, this attack can have high impact as the messages necessary to instruct the creation of a DID are trivial to create in bulk. Due to the need for broadcasting, replicating these messages leads to significant computational expense. This attack can be mitigated by employing conventional mitigation techniques.

Withholding Replication Attack

It is the responsibility of consortium members to propagate any instructions to create, update or delete DID to other members. Failure to do so will allow them to effectively censor the DID network. This attack is out

of scope for the following reasons:

1. Consortium members are trusted entities that are unlikely to have an incentive to negatively impact the system's trustworthiness by censoring it.
2. DID holders are free to select any of the consortium members for any DID interaction and can therefore pick a different member should they suspect an attack.

Cessation of Service

As DID documents are not stored on a public ledger, should all consortium members cease operation at the same time, DID data will not be recoverable from the system.

2.3 Privacy Considerations

The Corda DID Method adheres to the *Privacy Considerations* section of the Decentralized Identifiers Core architecture.

3. Methods

End users who aim to *create*, *read*, *update* or *delete* DID documents can do so by interacting with a trusted node of their choosing. The API provided for interaction exposes REST endpoints over HTTP using a JSON-based envelope format closely aligned with the JSON-LD examples found in the DID draft community report.

When users interact with consortium-member nodes, their requests will be handled by a web server that transforms the requests into a format suitable for Corda. The web-server component runs in a process independent of Corda. User calls 'proxied' in this way will invoke a Flow on one of the consortium nodes. As part of this flow, consortium nodes will check that the ID provided by the user is valid and that the message has cryptographic integrity (i.e., that the DID document is signed properly). Once this validation has been successfully completed, DID documents will be replicated from the *trusted node* (i.e., the node the user has chosen to interact with via REST) to all *witness nodes* (i.e., all other nodes in the consortium). Witness nodes will perform a cryptographic integrity check as part of the contract underpinning this transaction.

3.1 Corda DID Format

A Corda DID specifies the `corda` method, a target network (e.g. `testnet` or `tcn`) and a UUID formatted in the canonical string representation:

```
did:corda:(testnet|tcn|private-[a-z]+):
([0-9a-f]{8}\b-[0-9a-f]{4}-[0-9a-f]{4}-[0-9a-f]{4}-\b[0-9a-f]{12})
```

The Corda DID format covers two fixed target networks with fixed sets of consortium nodes. It also provides an examplary network tag for testing.

Target Network	Network Tag	Purpose
Testnet	`testnet`	Signifies that the document can be resolved via Testnet.
Corda Network	`tcn`	Signifies that the document can be resolved via the Corda Network.
Private	`private-*`	Signifies that the document cannot be resolved via a standard mechanism. This tag is intended for use in private networks and in testing.

I.e.:

- `did:corda:testnet:559d1c8f-75dd-477f-b28a-ef9d96c4e802`
- `did:corda:tcn:ffe0f4ff-8740-470d-bb4c-0b642f58e0f5`

- did:corda:private-persistent:d3b91530-67f5-48b8-bf1c-e883b1fea766

3.2 JSON-over-HTTPS API

The DID API is the server component to be deployed by consortium-member nodes in conjunction with the CorDapp. It provides method-specific APIs to *create, read, update* or *delete* DID documents. The Corda DID method achieves proof of ownership of a *document* by requiring proof of ownership of the keys contained in the document. To implement this, any DID document must be wrapped in an envelope. This envelope must contain signatures for all private keys associated with public keys contained in the documents (cf. figure 2).

Envelopes that do not contain signatures for all public keys will be rejected. Envelopes using unsupported

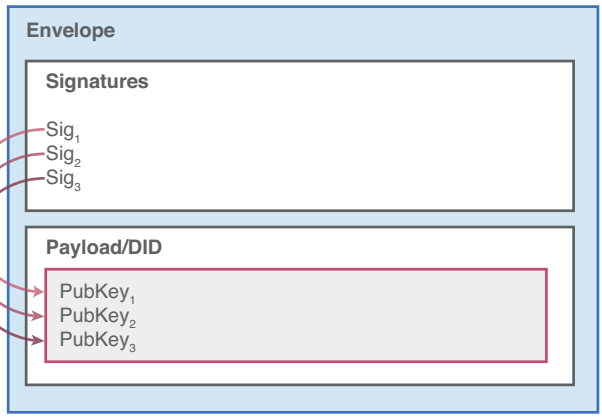

Figure 2: A DID envelope holds the document payload and signatures over the payload by relevant keys

cryptographic suites or unsupported serialization mechanisms will be rejected. There are restrictions on which suites and serialization mechanisms can be used.

3.2.1 Message Format

3.2.1.1 Instruction data

Instruction data tell the API what to do with the document received. They also contain proof of ownership of keys. Instruction data are formatted according to the following schema:

```
{
  "definitions": {},
  "$schema": "http://json-schema.org/draft-07/schema#",
  "type": "object",
  "required": [
    "action",
    "signatures"
  ],
  "properties": {
    "action": {
      "$id": "#/properties/action",
      "type": "string",
      "enum": [
        "create",
        "read",
        "update",
        "delete"
      ]
    },
```

```
    "signatures": {
      "$id": "#/properties/signatures",
      "type": "array",
      "items": {
        "$id": "#/properties/signatures/items",
        "type": "object",
        "required": [
          "id",
          "type",
          "signatureBase58"
        ],
        "properties": {
          "id": {
            "$id": "#/properties/signatures/items/properties/id",
            "type": "string",
            "description": "The ID of the public key that is part of the key pair
signing the document.",
            "examples": [
              "did:corda:testnet:3df6b0a1-6b02-4053-8900-8c36b6d35fa1#keys-1",
              "did:corda:tcn:3df6b0a1-6b02-4053-8900-8c36b6d35fa1#keys-2"
            ],
            "pattern": "^did:corda:(testnet|tcn|private-[a-z]+):([0-9a-f]{8}\\b-[0-9a-f]
{4}-[0-9a-f]{4}-[0-9a-f]{4}-\\b[0-9a-f]{12})$"
          },
          "type": {
            "$id": "#/properties/signatures/items/properties/type",
            "type": "string",
            "description": "The cryptographic suite this key has been generated with.
More formats (RsaSignature2018, EdDsaSASignatureSecp256k1) to follow.",
            "enum": [
              "Ed25519Signature2018"
            ]
          },
          "signatureBase58": {
            "$id": "#/properties/signatures/items/properties/signatureBase58",
            "type": "string",
            "description": "The binary signature in Base58 representation. More
formats to follow.",
            "examples": [
              "54CnhKVqE63rMAeM1b8CyQjL4c8teS1DoyTfZnKXRvEEGWK81YA6BAgQHRah4z1VV4aJpd
2iRHCrPoNTxGXBBoFw"
            ]
          }
        }
      }
    }
  }
}
```

3.2.1.1.1 Supported Encodings

Encoding	Description
`signatureHex`	Hex-encoded signature
`signatureBase64`	Base64-encoded signature as per
`signatureBase58`	Base58-encoded signature

signatureMultibase	Multibase-encoded signature

3.2.1.2 Examples

3.2.1.2.1 Base58

```
{
  "action": "create",
  "signatures": [
    {
      "id": "did:corda:tcn:d51924e1-66bb-4971-ab62-ec4910a1fb98#keys-1",
      "type": "Ed25519Signature2018",
      "signatureBase58": "54CnhKVqE63rMAeM1b8CyQjL4c8teS1DoyTfZnKXRvEEGWK81YA6BAgQHRah
4z1VV4aJpd2iRHCrPoNTxGXBBoFw"
    }
  ]
}
```

3.2.1.2.2 Multibase

```
{
  "action": "create",
  "signatures": [
    {
      "id": "did:corda:tcn:84602311-bd95-4006-968c-01a69d035d64#keys-1",
      "type": "Ed25519Signature2018",
      "signatureMultibase": "bb3i4jlob2pomlx5yjv5adir7r26tkor6iqroosojkqi4wq2kcjtiju3mo
xsrkwmobhtlega27uzuxtncks6yib6otqybfykjyzieqe"
    }
  ]
}
```

3.2.1.2.3 Hex

```
{
  "action": "create",
  "signatures": [
    {
      "id": "did:corda:tcn:03d7411f-ae67-4c89-94b8-de802f017745#keys-1",
      "type": "Ed25519Signature2018",
      "signatureHex": "04242D453FA6191B67308B9454E08EC2D59524063F53F85D11E43151283DF6729
59B9F546CD437FACA914DD15D41F7F6B5FF0AA00ABF5EE91826C70EA83F0E03"
    }
  ]
}
```

3.2.1.2.4 Base64

```
{
  "action": "create",
  "signatures": [
    {
      "id": "did:corda:tcn:4b78d87e-1dee-403d-89d6-d2e12926d309#keys-1",
      "type": "Ed25519Signature2018",
      "signatureBase64": "Kh39kEoMvzfolBimiT/6wGeTys5Leuk/M0im9CligIpRXsJnIx4STphsofZB
nbX198H7AfuVp8IJYyzMwKtaAg=="
    }
  ]
}
```

3.3 HTTP Communication

Envelopes are implemented as `multipart/form-data` HTTP requests with two parts:

Part	Purpose
`instruction`	JSON representation of the instruction
`document`	JSON representation of the DID document

This format is chosen to circumvent issues with canonical document representation for hashing.

3.3.1 Create DID

3.3.1.1 Request

The instruction to create a new DID is issued via a `PUT` HTTP request. Proof of ownership of the document has to be presented in the envelope.

The payload includes:

- The document, consisting of the encoded public key, the type of public key and the controller of the public key;
- The instruction, consisting of action to be taken (create), the encoded signature(s) in the document and the type of signature.

Instruction

```
{
  "action": "create",
  "signatures": [
    {
      "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-1",
      "type": "Ed25519Signature2018",
      "signatureBase58": "2M12aBn5ijmmUyHtTf56NTJsUEUbpbqbAgpsvxsfMa2KrL5MR5rGb4dP37Q
oyRWp94kqreDMV9P4K3QHfE67ypTD"
    }
  ]
}
```

Document

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5",
  "publicKey": [
    {
      "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-1",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5",
      "publicKeyBase58": "GfHq2tTVk9z4eXgyNRg7ikjUaaP1fuE4Ted3d6eBaYSTxq9iokAwcd16hu
8v"
    }
  ]
}
```

Request Example (cURL)

```
curl -X PUT \
http://example.org/did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5 \
  -H 'content-type: multipart/form-data' \
  -F instruction='{
```

```
  "action": "create",
  "signatures": [
 {
   "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-1",
   "type": "Ed25519Signature2018",
   "signatureBase58": "2M12aBn5ijmmUyHtTf56NTJsUEUbpbqbAgpsvxsfMa2KrL5MR5rGb4dP37QoyR
Wp94kqreDMV9P4K3QHfE67ypTD"
 }
  ]
}' \
  -F document'={
  "@context": "https://w3id.org/did/v1",
  "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5",
  "created":"2019-07-11T10:27:27.326Z",
  "publicKey": [
 {
   "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-1",
   "type": "Ed25519VerificationKey2018",
   "controller": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5",
   "publicKeyBase58": "GfHq2tTVk9z4eXgyNRg7ikjUaaP1fuE4Ted3d6eBaYSTxq9iokAwcd16hu8v"
 }
  ]
}'
```

3.3.1.2 Response

- The API will respond with status `204` for a request with a well-formed instruction *and* a well-formed document *and* valid signature(s) and an unused ID.
- The API will respond with status `400` for a request with a deformed instruction *or* a deformed document or at least one invalid signature.
- The API will respond with status `409` for a request with an ID that has already been taken.

3.3.2 Read DID

3.3.2.1 Request

A simple `GET` request (specifying the ID as a fragment) is used to retrieve a DID document. The DID document contains a list of public keys, the type of public key, information about encodings used in these public keys and the controller of each public key.

Request Example (cURL)

```
curl -X GET http://example.org/did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5
```

3.3.2.2 Response

```
{
  "@context": "https://w3id.org/did/v1",
  "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5",
  "created": "2019-07-11T10:27:27.326Z",
  "publicKey": [
    {
      "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-1",
      "type": "Ed25519VerificationKey2018",
      "controller": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5",
      "publicKeyBase58": "GfHq2tTVk9z4eXgyNRg7ikjUaaP1fuE4Ted3d6eBaYSTxq9iokAwcd16hu
8v"
    }
  ]
}
```

- The API will respond with status `200` for a request with a known ID.
- The API will respond with status `404` for a request with an unknown ID.
- The API will respond with status `400` for a request with an ID in the incorrect format.

3.3.3 Update DID

3.3.3.1 Request

The instruction to create a new DID is issued via a `POST` HTTP request [rfc4122].

Updates use the optional created and updated concepts to mitigate replay attacks. This means an update will only be successful if the `updated` field *in the DID document* is set to an instant that is later than the instant previously saved in that field. Should no previous update be recorded, the update will only be successful if the `created` field *in the document* is set to an instant that is later than the instant provided with the update.

The current time is calculated by the DID owner, without verification of its accuracy by the consortium. This is appropriate, however, since this field is only used to determine a before/after relationship. Consumers of the DID document need to take into account that this value is potentially inaccurate.

The payload includes:

- The document, consisting of the new encoded public key, the type of public key and controller of the public key;
- The instruction, consisting of action to be taken (update), encoded signature(s) in the document using all private keys (including the one being added) associated with public keys in the document and the type of signature.

Request Example (cURL)

```
curl -X POST \
http://example.org/did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5 \
  -H 'content-type: multipart/form-data' \
  -F instruction='{
  "action": "update",
  "signatures": [
    {
        "id":"did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-1",
        "type":"Ed25519Signature2018",
        "signatureBase58":"57HQXkem7pXpfHnP3DPTyLqSQB9NuZNj7V4hS61kbkQA28hCuYtSmFQCABj
8HBX2AmDss13iDkNY2H3zqRZsYnD4"
    },
    {
        "id":"did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-2",
        "type":"Ed25519Signature2018",
        "signatureBase58":"26kkhZbQLSNvEKbPvx18GRfSoVMu2bDXutvnWcQQyrGxqz5VKijkFV2Goh
bkbafPa2WqVad7wnyLwx1zxjvVfvSa"
    }
  ]
}' \
  -F document'={
  "@context": "https://w3id.org/did/v1",
  "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5",
  "created":"2019-07-11T10:27:27.326Z",
  "updated":"2019-07-11T10:29:15.116Z",
  "publicKey": [
 {
   "id": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-2",
   "type": "Ed25519VerificationKey2018",
```

```
    "controller": "did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5",
    "publicKeyBase58": "GfHq2tTVk9z4eXgyHhSTmTRf4NFuTv7afqFroA8QQFXKm9fJcBtMRctowK33"
  }
  ]
}'
```

3.3.3.2 Response

- The API will respond with status `204` if the update is successful.
- The API will respond with status `404` for a request with an unknown ID.
- The API will respond with status `400` for other cases of incorrect payload (mismatched signatures, a malformed document, malformed instructions, etc.).

3.3.4 Delete DID

3.3.4.1 Request

The instruction to create a new DID is issued via a `DELETE` HTTP request.

This method is used to disable the identity in the ledger. Once deleted, the identity cannot be used again. The delete function only accepts an instruction as a payload. The instruction contains signature(s) for public key(s) for the latest DID document in the ledger.

The payload includes the instruction, consisting of action to be taken (delete), encoded signature(s) in the latest DID document in the ledger using all private keys associated with public keys present in the document and the type of signature.

To validate a delete request, the user must provide signature(s) in the instruction. The signature(s) must be over the latest DID document present in the ledger, signed with corresponding private keys for all public keys present in the document.

Request Example (cURL)

```
curl -X DELETE \
http://example.org/did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5 \
  -H 'content-type: multipart/form-data' \
  -F instruction='{
  "action": "delete",
  "signatures": [
    {
        "id":"did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-1",
        "type":"Ed25519Signature2018",
        "signatureBase58":"57HQXkem7pXpfHnP3DPTyLqSQB9NuZNj7V4hS61kbkQA28hCuYtSmFQCABj
8HBX2AmDss13iDkNY2H3zqRZsYnD4"
    },
    {
        "id":"did:corda:tcn:a609bcc0-a3a8-11e9-b949-fb002eb572a5#keys-2",
        "type":"Ed25519Signature2018",
        "signatureBase58":"26kkhZbQLSNvEKbPvx18GRfSoVMu2bDXutvnWcQQyrGxqz5VKijkFV2Goh
bkbafPa2WqVad7wnyLwx1zxjvVfvSa"
    }
  ]
}'
```

4. Implementation Considerations

This section is non-normative.

This section outlines 'on-ledger' implementation of the protocol, which has only marginal use.

A reference implementation of this method is available publicly.

DID Flows is the CorDapp component to be deployed by consortium-member nodes. It provides method-specific flows to create, read, update or delete DID documents. The DID flows can be invoked from RPC clients or from other flows.

4.1 Create DID

This is used to create a new DID. Proof of ownership of the document has to be presented in the envelope, as outlined in the API format.

- Invoke CreateDidFlow via RPC: `rpc.startFlowDynamic(CreateDidFlow::class.java, envelope)`.
- Invoke CreateDidFlow from another flow: `subFlow(CreateDidFlow(envelope))`,

where 'envelope' is an instance of the `DidEnvelope` type.

4.2 Read DID

This is used to fetch a DID document from a node's local vault. It returns an instance of the `DidDocument` type.

- Invoke FetchDidDocumentFlow via RPC: `rpc.startFlowDynamic(FetchDidDocumentFlow::class.java, linearId)`.
- Invoke FetchDidDocumentFlow from another flow: `subFlow(FetchDidDocumentFlow(linearId))`,

where 'linearId' is an instance of the `UniqueIdentifier` type and is the UUID part of the DID.

There might be a case where a node which is not part of the DID Business Network may request a DID document from one of the DID consortium nodes. In such situations, nodes can invoke `FetchDidDocumentFromRegistryNodeFlow`.

- Invoke FetchDidFetchDidDocumentFromRegistryNodeFlowDocumentFlow via RPC: `rpc.startFlowDynamic(FetchDidDocumentFromRegistryNodeFlow::class.java, didRegistryNode, linearId)`.
- Invoke FetchDidDocumentFromRegistryNodeFlow from another flow: `subFlow(FetchDidDocumentFromRegistryNodeFlow(didRegistryNode, linearId))`,

where 'linearId' is an instance of the `UniqueIdentifier` type and is the UUID part of the DID, and 'didRegistryNode' is an instance of the Party type representing the DID consortium node.

4.3 Update DID

This is used to update an existing DID.

- Invoke UpdateDidFlow via RPC: `rpc.startFlowDynamic(UpdateDidFlow::class.java, envelope)`.
- Invoke UpdateDidFlow from another flow: `subFlow(UpdateDidFlow(envelope))`,

where 'envelope' is an instance of the `DidEnvelope` type.

4.4 Delete DID

This is used to disable an existing DID. The delete operation introduces no changes to the `DidDocument`. It expires the `DidState` and is marked as `Consumed` in the ledger. To validate a delete request, the user must provide signature(s) in the instruction. The signature(s) are to be created using the key pair referenced in the latest DID document present in the ledger, signed with corresponding private keys for all public keys present in the document.

- Invoke DeleteDidFlow via RPC: `rpc.startFlowDynamic(DeleteDidFlow::class.java, instruction, did)`.
- Invoke DeleteDidFlow from another flow: `subFlow(DeleteDidFlow(instruction, did))`,

where 'instruction' is the instruction JSON object (in string form), containing DID-owner signatures in the DID-document to be deactivated; `did` is the DID to be deleted.

A. References

Corda Networking and Messaging. URL: https://docs.corda.net/messaging.html

Corda Network Foundation. URL: https://corda.network/

Corda Testnet. URL: https://docs.corda.net/head/corda-testnet-intro.html

Corda Technical White Paper. Mike Hearn; Richard Gendal Brown. URL: https://www.r3.com/reports/corda-technical-whitepaper/

Distributed denial of service (DDoS) resilience in cloud: Review and conceptual cloud DDoS mitigation framework. Opeyemi Osanaiye; Kim-Kwang Raymond Choo; Mqhele Dlodloa. URL: https://doi.org/10.1016/j.jnca.2016.01.001

Decentralized Identifiers (DIDs) v1.0. Drummond Reed; Manu Sporny; Markus Sabadello; Dave Longley; Christopher Allen. W3C. 25 February 2020. W3C Working Draft. URL: https://www.w3.org/TR/did-core/

The Multibase Data Format. J. Benet; M. Sporny. URL: https://tools.ietf.org/id/draft-multiformats-multibase-00.html

A Universally Unique IDentifier (UUID) URN Namespace. P. Leach; M. Mealling; R. Salz. IETF. July 2005. Proposed Standard. URL: https://tools.ietf.org/html/rfc4122

The Base16, Base32, and Base64 Data Encodings. S. Josefsson. IETF. October 2006. Proposed Standard. URL: https://tools.ietf.org/html/rfc4648

Returning Values from Forms: multipart/form-data. L. Masinter. IETF. July 2015. Proposed Standard. URL: https://tools.ietf.org/html/rfc7578