IDealWALLET Specification - KASMIR DID Method Specification

Tobias Ehrlich (KAPRION Technologies GmbH)

tobias.ehrlich@kaprion.de

Petr Apletauer (KAPRION Technologies GmbH)

petr.apeltauer@kaprion.de

Ben Biedermann (formerly KAPRION Technologies GmbH)

bb@dvctvs.wtf

Draft

Version

1.0.0

Date

2023-09-21

Archived Version

Table of Contents

Most Decentralized Identifier (DIDs) target Distributed Ledgers Technologies (DLT) or require to be online to achieve a sufficient level of trust. This document serves the specification for the generation and interpretation of an offline capable DID method branded KASMIR DID.

1. Conformance

In this document the key words *MAY*, *MUST*, *MUST NOT*, and *SHOULD* are used and to be interpreted as described in [RFC2119] and when they appear, they are capitalized.

2. Terminology

ASM

Hardware Secure Module with installed Secure Applet

DID

Decentralized Identifier [DID-CORE]

DLT

Distributed Ledgers Technologies

EC

Elliptic Curves

Holder

the subject who holds one or more credentials and generating presentations from them

HSM

Hardware Secure Module

ID

identifier is a name that identifies/labels a unique class of objects

Issuer

creating and issuing a credential

IRI

Internationalized Resource Identifiers

JSON

JavaScript Object Notation

JWK

JSON Web Key as specified in [RFC7517]

KAPRION

Cooperative and Adaptive Process Interoperability Online Network (Kooperatives und Adaptives Prozess-Interoperabilitäts-Online-Netzwerk)

KASMIR

KAPRION ASM Mír

SSI

Self-Sovereign Identity, a paradigm of digital identity control that returns control of an entity's own digital being to that entity itself.

URI

Uniform Resource Identifier

URN

Uniform Resource Name

UUID

Universally Unique Identifier

Verifier

is receiving one or more credentials, optionally inside a presentation, for processing

W3C

World Wide Web Consortium

Witness

is an entity which validates or anchors the key event log with key states.

3. Introduction

In den meisten Fällen kann keine flächendeckende Garantie ausgesprochen werden, dass Geräte, die digitale Kommunikationsfähigkeiten besitzen, immer an einem aktiven Netzwerk hängen und somit nicht immer online auf Daten zugreifen können. Für die meisten web-basierende Angebote stellt das kein Problem dar, da sie ohnehin nur online verfügbar sind, dennoch sollten Daten ebenso leicht ohne Drittnetze (im folgendem als offline bezeichnet) austauschbar sein, ähnlich wie seit Jahrhunderten Zahlungsmittel ihren Besitzer wechseln. Eine offline fähige ID muss jedoch auch auch vertrauenswürdig sein, denn schließlich hat ein Gerät nicht immer die gleichen interfaces wie ein etwaiger menschlicher Konterfei, um zusätzliche attribute zu prüfen. Diese Anforderungen gelingen nur in zusammenspiel verschiedener Komponenten. Im folgendem wird die did:kasmir vorgestellt, verschiedene Technologien vereint, bzw. erweitert.

Zur Offlinefähigkeit wird als Anforderung eine ledger agnostic voraus gesetzt, die u.a. von [DID-KEY], wo der identifier den public key darstellt, [DID-WEB] [DID-PEER] [DID-KERI]

- Anforderungen HSM, offline, ohne DLT, Interop

- basiert auf einer modifizierten KERI
- merge peer / web

The DID method *MUST* be ledger agnostic

- **did:peer** Method 2 allows us to resolve the DID offline ⇒ we should have something similar for **did:keri** if possible
- if DID cannot be resolved, DID resolver should look into the DIDComm message attachments to take DID Document from there
- if the DID Document cannot be resolved and is not in attachments → send ***adopted* problem-report message**
- Resolution by KERI docs:
- KERI DID Mehthod

The KASMIR DID specification conforms the requirements of the Decentralized Identifiers v1.0 Specification [DID-CORE] and will be prefixed with `did:kasmir`.

4. Concept

4.1. Keys

- seems we need only **Assertion** and **Key Agreement** key pairs (see)
- well perhaps also **Authentication** key pair to authenticate 2 HW chips…
- **symmetric session key** will be created from the **Key Agreement** key pair

4.2. Events

4.3. Key Event Log

tbd.

4.4. Key Event Receipt Log

tbd.

4.5. Key State

tbd.

The did:peer section for Method 2 describes how all keys can be compressed as one long identifier. In KERI it is not forbidden to generate something similar but other information also need to be added. Also there is a big disadvantage because it is not realy efficient to handle such long identifier in the Applet. Therefore I (@Tobias) recommend to generate a URL like string as it is described in the W3C Decentralized Identifiers (DIDs) v1.0 specification. As

described above the *key state* object is part of the DID document so it might be useful to attach it as part of a base64url encrypted DID fragment.

`did:keri:prefix#base64KeyStateObject`

Alternatively a query could be constructed:

`did:keri:prefix?keyState=base64KeyStateObject`

If we assume the ICP event above is the last key state event, then we can create

did:keri:EZAoTNZH3ULvaU6Z-i0d8JJR2nmwyYAfSVPzhzS6b5CM#ImtleVN0YXRlIjp7InYiOiJLRVJJMTBKU09OMDAwMTFjXyIsImkiOiJFWkFvVE5aSDNVTHZhVTZaLWkwZDhKSlIybm13eVlBZlNWUHpoelM2YjVDTSIsInNiOiIwIiwidCI6ImljcCIsImt0IjoiMSIsImsiOlsiRGFVNkpSMm5td3laLWkwZDhKWkFvVE5aSDNVTHZZQWZTVlB6aHpTNmI1Q00iXSwibiI6IkVaLWkwZDhKWkFvVE5aSDNVTHZhVTZKUjJubXd5WUFmU1ZQemh6UzZiNUNNIiwid3QiOiIxIiwidyI6WyJEVE5aSDNVTHZhVTZKUjJubXd5WUFmU1ZQemh6UzZiWi1pMGQ4SlpBbzVDTSJdLCJjIjpbIkVPII19

or

did:keri:EZAoTNZH3ULvaU6Z-i0d8JJR2nmwyYAfSVPzhzS6b5CM?keyState=eyJ2IjoiS0VSSTEwSlNPTjAwMDExY18iLCJpIjoiRVpBb1ROWkgzVUx2YVU2Wi1pMGQ4SkpSMm5td3lZQWZTVlB6aHpTNmI1Q00iLCJzIjoiMCIsInQiOiJpY3AiLCJrdCI6IjEiLCJrIjpbIkRhVTZKUjJubXd5Wi1pMGQ4SlpBb1ROWkgzVUx2WUFmU1ZQemh6UzZiNUNNIl0sIm4iOiJFWi1pMGQ4SlpBb1ROWkgzVUx2YVU2SIlybm13eVlBZlNWUHpoelM2YjVDTSIsInd0IjoiMSIsInciOlsiRFROWkgzVUx2YVU2SIlybm13eVlBZlNWUHpoelM2YlotaTBkOEpaQW81Q00iXSwiYyI6WyJFTyJdfQ

Note: The JSON was minified before base64url encoding

A simple key state event from [KERI] whitepaper

{

"v" : "KERI10JSON00011c_",

"i" : "EZAoTNZH3ULvaU6Z-i0d8JJR2nmwyYAfSVPzhzS6b5CM", //DID of owner = controller

"s" : "0",

"t" : "icp",

"kt": "1",

"k" : [

"CF5pxRJP6THrUtlDdhh07hJEDKrJxkcR9m5u1xs33bhp", //C ~ X25519KeyAgreementKey2019

"DaU6JR2nmwyZ-i0d8JZAoTNZH3ULvYAfSVPzhzS6b5CM", //D ~ Ed25519VerificationKey2018 or Ed25519VerificationKey2020 (just choose one)

"1AABAsL0-AEWBfl876zt4XcTWpGcA4V248OF-n-8gou45OZA" //1AAB ~ EcdsaSecp256k1VerificationKey2019

],

"n" : "EZ-i0d8JZAoTNZH3ULvaU6JR2nmwyYAfSVPzhzS6b5CM",

"wt": "1",

"w" : ["DTNZH3ULvaU6JR2nmwyYAfSVPzhzS6bZ-i0d8JZAo5CM"],

"c" : ["EO"]

}

4.6. Resolver Metadata

tbd.

Like did:peer KERI keys also can have a *purposecode* and *transform* information but it is not encoded in constant two bytes, depending on purpose and/or key type the code length can go up to 12 bytes. At the moment only a length up to 4 bytes is documented.

4.7. The DID Document

tbd.

- DID Doc used in DIDComm
- Key State compact but needs to be resolved to DID Doc
- Key Log a set of Key states

see issue 20 IDeal wallet

A simple DID document taken from example 1 of [DID-CORE]

{

"@context": [

"https://www.w3.org/ns/did/v1",

"https://w3id.org/security/suites/ed25519-2020/v1"

]

"id": "did:example:123456789abcdefghi",

"authentication": [{

"id": "did:example:123456789abcdefghi#keys-1",

"type": "Ed25519VerificationKey2020",

"controller": "did:example:123456789abcdefghi",

"publicKeyMultibase": "zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"

}]

}

5. DID Specification

5.1. Method Name

The method name that identifies this DID method *MUST* be: `kasmir`.

When `did:kasmir` is used a ASM which fulfills KAPRIONs KASMIR Applet specification *MUST* be used, an additional registry, such as DLT, *MAY* be optionally. If connected to an additional register the KASMIR method name *MUST* be pressent to identify the additional capabilities of the KASMIR applet.

Possible DID strings:

- `did:kasmir:method-specific-id`
- `did:peer:3method-specific-id`; Note the numalgo *3* is not yet defined Peer DID Method.
- `did:indy:kasmir:method-specific-id`
- `did:indy:sov:kasmir:method-specific-id`
- `did:…:kasmir:method-specific-id`

5.2. Method Specific Identifier

Like in KERI the method specific identifier for the `kasmir` method is a self-addressing identifier which is fully generated and maintained within the ASM and protected by hardware.

The KASMIR self-addressing identifier is cryptographically bound to the inception keys used to create it.

6. Protocols

6.1. Create

wip.

1. Required Applet calls
2. init Key Object
3. add n key(s)
4. add m witness(es)/anchors
5. gen method-specific-id
6. Generate Key State
7. add public keys
8. add info about next keys
9. add witness(es) and anchors (otional)
10. sign the ICP
11. store the ICP in a key event log (KEL)
12. generate DID document (App)
13. convert key event message to JSON
14. extract prefix and gen did-string
15. extract keys and add them to verificationMethod object
16. add additional info
17. add JSON key event message to DID document

6.1.1. Prefix generation

Symbols

- NNN … prefix derivation code letter
- HHH … SHA-256
- DDD … Base64URL derivation
- QQQ … current Public Key
- Q'Q'Q' … next Public Key
- QQ^\hat{Q} … witness Public Key
- pabpabpab … prefix agreement byte

Generation process

1. generate the full KERI object (add keys & witness & prefix agreement byte aka configuration mode)
2. generate a SHA-256 hash over all pre-rotaded public keys

$next$=H(Q'0,⋯,$Q'n$),n∈ℕnext=H(Q0′,⋯,Qn′),n∈N

next = H(Q'_0, \cdots, Q'_n), n \in \mathbb{N}

3. generate a SHA-256 hash over all current public keys

$current = H(Q0, \cdots, Qn), n \in \mathbb{N}$current=H(Q0,⋯,Qn),n∈N

current = H(Q_0, \cdots, Q_n), n \in \mathbb{N}

4. generate a SHA-256 hash over all witnesses

$witnesses = H(Q0, \cdots, Qm), m \in \mathbb{N}$witnesses=H(Q^0,⋯,Q^m),m∈N

witnesses = H(\hat{Q}_0, \cdots, \hat{Q}_m), m \in \mathbb{N}

This step can provide a hen-egg-problem if the witnesses isn't already established and the event is not a delegated inception event (DIP), therefore this is at the moment a not available for public.

5. concatenate prefix agreement byte, current, next and witnesses and generate a SHA-256 hash

$data = H(pab \Vert current \Vert next~[\Vert witnesses])$data=H(pab‖current‖next [‖witnesses])

data = H(pab \mathbin\Vert current \mathbin\Vert next~[\mathbin\Vert witnesses])

6. generate a Base64URL string of the resulting hash

$digest = D(data)$digest=D(data)

digest = D(data)

7. add derivation code for KAPRION Version

$method-specific-id = \text{'N'} \Vert digest$method−specific−id='N'‖digest

method-specific-id = \text{'N'} \mathbin\Vert digest

6.2. Read

tbd.

sequenceDiagram

Alice->>John: Hello John, how are you?

John-->>Alice: Great!

Alice-)John: See you later!

6.3. Rotate

tbd.

6.4. Delete

tbd.

6.5. Deactivate

tbd.

6.6. Resolving (offline)

tbd

How to resolve KERI DID into DID document (meaning JSON object `didDocument` as per DIDComm v2 docs) - not just KERI DID is needed for this, but also KEL (Key Event Log) or its current Key State object

1. get KEL (Key Event Log) from `keyState` query parameter and service (if available) from `service` parameter

- `did:keri:prefix?keyState=base64KeyStateObject&didDocService=base64ServiceObject`
- `service` block is base64 encoded after whitespace removal and common word substitution as in Peer DID docs

2. extract prefix string from KEL (attribute `i`) and generate did as did:keri:prefix
3. extract keys (attr. `k`) from KEL (ICP or last ROT event) - *those are actually only x-coordinates of the pub. key*
4. build DID document -for each key:
5. convert x-coordinate to Base64URL format by removing derivation code (e.g. `D`) and adding `=`

- e.g. `DaU6JR2nmwyZ-i0d8JZAoTNZH3ULvYAfSVPzhzS6b5CM` ⇒ `aU6JR2nmwyZ-i0d8JZAoTNZH3ULvYAfSVPzhzS6b5CM=`

6. Use proper crypto algorithm to create full pub. key from the x-coordinate
7. encode the pub.key in Base52 format
8. Get a key `type` by the derivation code (1st. char or first 4 chars)

- `C` is for X25519 public encryption key ⇒ might be X25519KeyAgreementKey2019 or X25519KeyAgreementKey2020 - *we can choose which depending on key representation in the DID document*
- `D` is for Ed25519 public signing verification key ⇒ Ed25519VerificationKey2018 or Ed25519VerificationKey2020 - *we can choose which depending on key representation in the DID document*
- `1AAB` is for ECDSA secp256k1 public key ⇒ EcdsaSecp256k1VerificationKey2019

9. Get key `controller` from attr. `i` - *we assume the controller is the DID owner*
10. Get key rights (verification relationship - `authentication`, `assertionMethod`, `keyAgreement`, `capabilityInvocation`…)

- keyAgreement is for key with derivation code `C`
- all other is for key with derivation code `1AAB` (or `D`)

11. add last Key State (ICP or ROT) as DID Document Metadata as defined in 2.5 Resolver Metadata

7. Security Considerations

This section is non-normative.

7.1. Key State Verification

tbd.

7.2. Confidentiality Violations, Password Sniffing

tbd.

7.3. Replay Attacks

tbd.

7.4. Message Insertion, Deletion, Modification

tbd.

7.5. Man-In-The-Middle Attacks

tbd.

References

- [DID-CORE] Decentralized Identifiers (DIDs) v1.0. M. Sporny; A. Guy; M. Sabadello; D. Reed. W3C. 19. July 2022. W3C Recommendation. URL: https://www.w3.org/TR/did-core/
- [DID-KERI] The did:keri Method v0.1. S. Smith; C. Cunningham; P. Feairheller. Identity Foundation. 10. November 2021. Unofficial Draft. URL: https://identity.foundation/keri/did_methods/#security-considerations

- [DID-KEY] The did:key Method v0.7. M. Sporny et al. W3C. September 2022. URL: https://w3c-ccg.github.io/did-method-key/
- [DID-PEER] Peer DID Method Specification. Oskar Deventer et al. Identity Foundation. 28. June 2023. W3C Document. URL: https://identity.foundation/peer-did-method-spec/
- [DID-WEB] did:web Method Specification. Christian Gribneau et al. W3C. 6. May 2023. URL: https://w3c-ccg.github.io/did-method-web/
- [KERI] Key Event Receipt Infrastructure (KERI) Design. S. Smith. GitHub. 7. May 2021. v2.60. URL: https://github.com/SmithSamuelM/Papers/blob/master/whitepapers/KERI_WP_2.x.web.pdf
- [RFC2119] Key words for use in RFCs to Indicate Requirement Levels. S. Bradner. IETF. March 1997. URL: https://www.rfc-editor.org/rfc/rfc2119
- [RFC7517] JSON Web Key (JWK). M. Jones. IETF. May 2015. URL: https://tools.ietf.org/html/rfc7517

Version 1.0.0

Last updated 2023-09-21 12:28:55 +0200