DIDn’t

did:un = KERI => DID

Presentation at IIW#31

by Charles Cunningham, Jolocom

October 21, 2020

KERI != DIDs

DIDs are a general interface for attaching some types of metadata to an Identifier (most commonly Public Keys).

KERI is a specific method for attaching some (slightly different) metadata to an Identifier.

did:un = transforming KERI Identifier metadata into appropriate DID Document contents (+ potential combinations of availability mechanisms) using minimal extensions for each

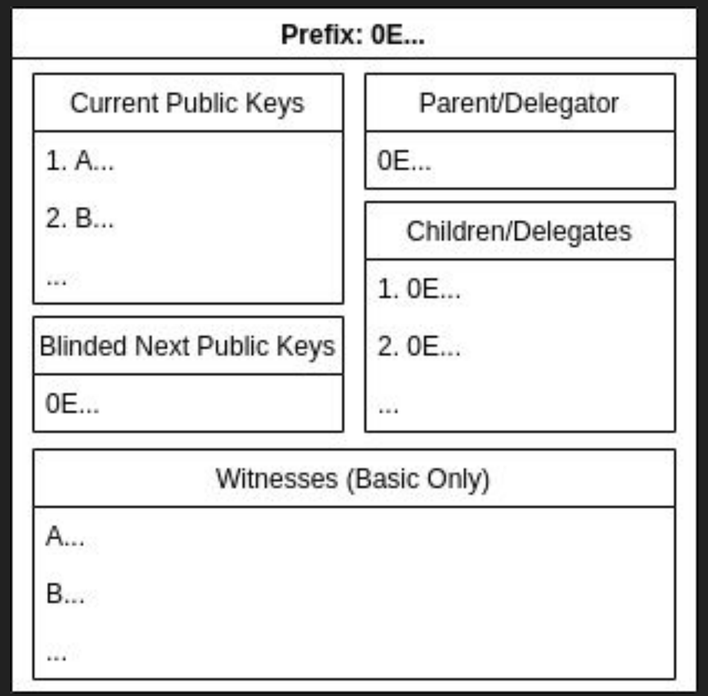
Context: Prefixes

- Prefix = Qualified Material = Self describing cryptographic material
- 3 kinds:
  - Basic: just a Public Key
  - Self Addressing: a cryptographic hash/digest of some data
  - Self Signing: a cryptographic signature
- Prefixes can be used as identifiers, but also are used to represent all crypto material in KERI
- Identified by the prepended “derivation code”
  - E.g. B => Ed25519 public key (Basic derivation), E => BLAKE3 digest (Self Addressing derivation), 1AAA => secp256k1 public key (Basic derivation)
- Format: concat(derivation_code, base64(material))

Context: State

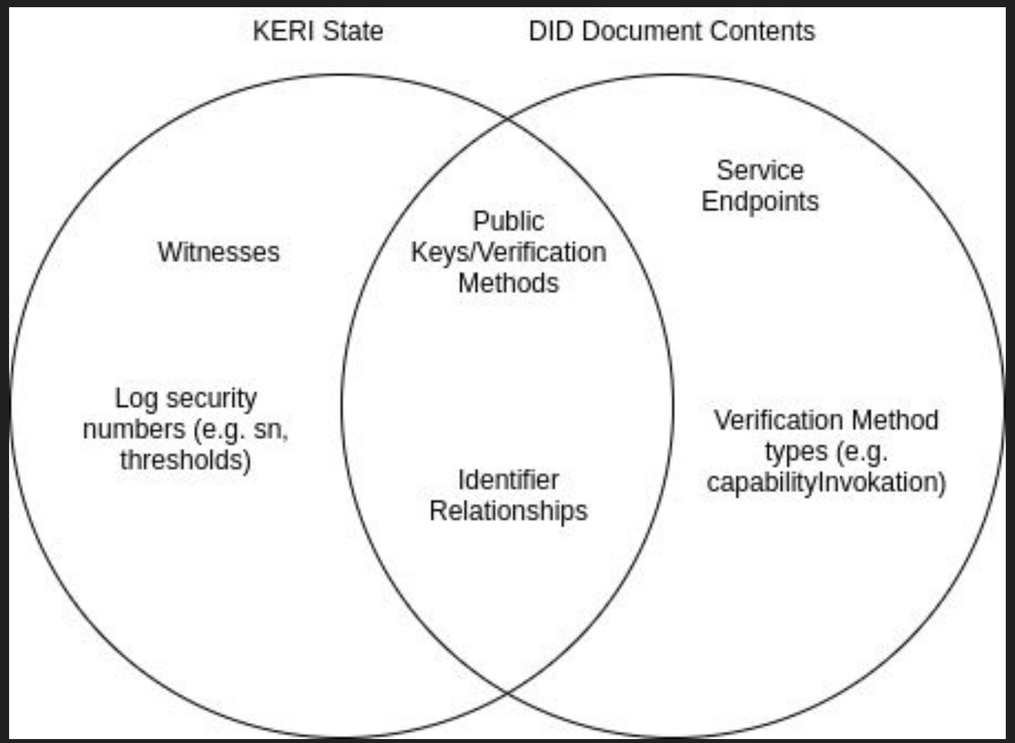
State Associated with an Identifier Prefix:

- Keys
  - Ordered List of Current Public Keys (Basic Prefixes)
  - Hash of Next Public Keys (Self Addressing Prefix)
  - Threshold for event signatures
- Identifier Tree
  - Delegator (optional)
  - Ordered List of Delegates
- Witness Set
  - Witnesses (Basic Prefixes Only)
  - Witness Threshold for event receipts



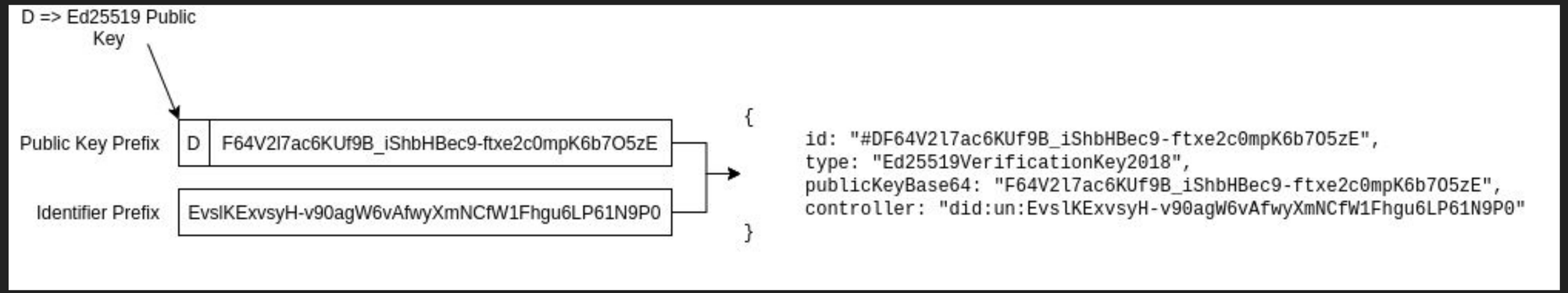
State and DID Documents

- Useful state:
  - Public Keys
  - Delegators and Delegates
- Not Useful state:
  - Event signature threshold
  - Witness receipt threshold
  - Witnesses



Keys and Verification Methods

Each key in a list of Current Public Keys for prefix P can be translated into a set of Verification Methods, as found on DID Documents:

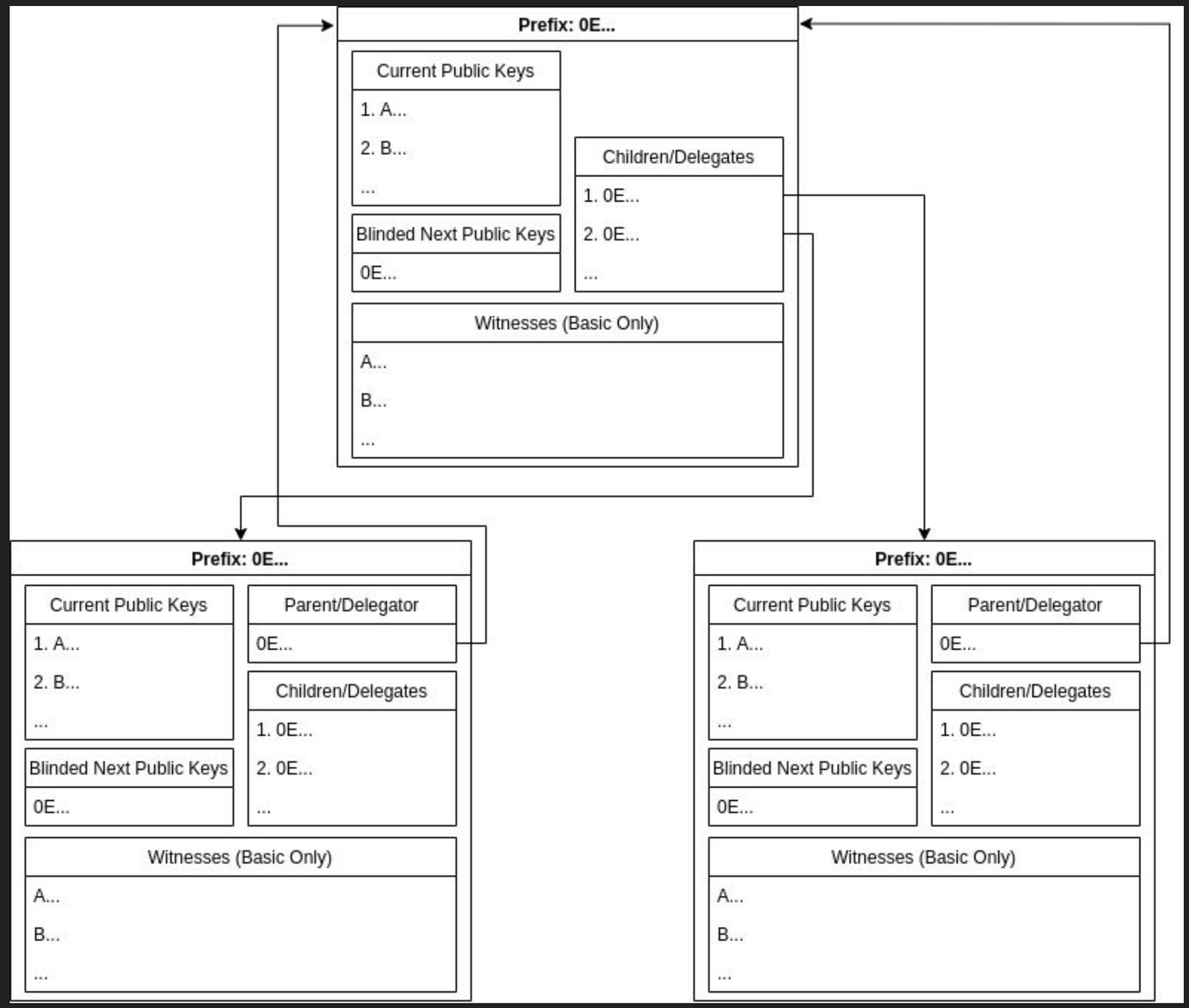


Keys and Verification Methods

```
fn pref_to_vm(
    pref: &BasicPrefix,
    controller: &IdentifierPrefix,
    method_prefix: &str,
) -> Result<VerificationMethod, String> {
    Ok(VerificationMethod {
        id: ["#".to_string(), pref.to_str()].join(""),
        key_type: match pref.derivation {
            Basic::Ed25519NT | Basic::Ed25519 => KeyTypes::Ed25519VerificationKey2018,
            Basic::ECDSAsecp256k1NT | Basic::ECDSAsecp256k1 => {
                KeyTypes::EcdsaSecp256k1VerificationKey2019
            }
            Basic::X25519 => KeyTypes::X25519KeyAgreementKey2019,
            _ => return Err("bad key type".to_string()),
        },
        controller: ["did", method_prefix, &controller.to_str()].join(":"),
        key: VerificationMethodProperties::Base64(base64::encode_config(
            pref.derivative(),
            base64::URL_SAFE,
        )),
    })
}
```

Delegation Tree

Each Prefix may recursively delegate to additional Prefixes in a tree-like fashion.



Charles Cunningham, Jolocom

Delegation: “controller” and “alsoKnownAs”

Delegates may contain >1 Public Key, and so cannot be represented by a simple Verification Method. They must be sub-Identifiers, but expressed how?

- controller: provides a reasonable approximation of the delegator relationship
  - “A DID controller is authorized to make changes to the respective DID document.”
  - “The corresponding DID document(s) SHOULD contain verification relationships that explicitly permit the use of certain verification methods for specific purposes.”
- alsoKnownAs: expresses an identity of subject
  - “This relationship is a statement that the subject of this identifier is also identified by one or more other identifiers.”
  - May not be exactly the same meaning as the delegation relationship

Resolution Example:

driverId = id, direct mode

Many different methods of indirect mode resolution/discovery are possible


```
{
       '@context': 'https://www.w3.org/ns/did-resolution/v1',
       didDocument: {
          '@context': 'https://www.w3.org/ns/did/v1',
          id: 'did:un:E85F-Se093ppwnuXqxhrsjO10TgCycQHq2dDUqIFAyD8',
          publicKey: [ {
              controller: 'did:un:EUxJ4LeINs5vzH9J1RO87ogEYwC2rfnJ6yqEzEq5n16w',
              id: '#DcNeAligZq5ne84HpVig03ns-qR39HrPgnmRMTM_u3a8',
              type: 'Ed25519VerificationKey2018',
              publicKeyHex: '70d780962819ab99def381e9562834de7b3ea91dfd1eb3e09e644c4ccfeeddaf'
           }, {
              controller: 'did:un:EUxJ4LeINs5vzH9J1RO87ogEYwC2rfnJ6yqEzEq5n16w',
              id: '#CtobRc5yTv7vopnlANEeOd-k8lGyhNCqbKnkGtM5aCzM',
              type: 'X25519KeyAgreementKey2019',
              publicKeyHex: 'b686d1739c93bfbbe8a6794034478e77e93c946ca1342a9b2a7906b4ce5a0b33'
       } ],
       resolverMetadata: {
          driverId: 'did:un:E85F-Se093ppwnuXqxhrsjO10TgCycQHq2dDUqIFAyD8',
          driver: 'jolocom/peer-resolution/0.1',
          retrieved: 1603291574934
       },
       methodMetadata: {
          stateProof:
'{"vs":"KERI10JSON00012a_","pre":"E85F-Se093ppwnuXqxhrsjO10TgCycQHq2dDUqIFAyD8","sn":"0","ilk":"icp","sith":"1","keys":["D4J_qIixMUJfRM
F6Wf-47lZpt8XgOIbo--7-lZVVoQkg","CR9dgxWZmj7WfVfGecHiariZJPzjtDbH1TgULa2pnWCc"],"nxt":"EXVPa540FXY1MY7maSPlVQSlaeBLzkxQoq_XYzifPQHA","t
oad":"0","wits":[],"cnfg":[]}-AABAAuXQHUUcCFCMwHwNHJzQgUWUFBDfUNvaYSEER0dtS-OzXSbK8qD9ZMv2PAGggrUXJgNo5v2NtmdhUMEim55rcCQ'
       }
}
```

Additional Transforms

- DID parameters can provide arbitrary additions to a DID Document (e.g. signed-ietf-json-patch)
- As long as control is established, patches can be verified and applied

Availability (the hard part)

KERI places no requirement or restriction on availability of data, only data format and integrity. Options Include:

- Something like Sidetree or combinations of 2nd layer solutions
- Kademlia DHT with witnesses as nodes (requires witnesses to stay online)
- Closed networks of KEL/KERL databases for private use (e.g. IoT)
- Cache

KERI could be a basis for a variety of actual method implementations with a unified Identifier space with 3 modes:

- Ephemeral: e.g. did:key (Identical in all methods)
- Direct: e.g. did:peer (Identical(?) in all methods)
- Indirect: e.g. did:ion, did:jolo, did:ethr etc. (may differ)

Ephemeral Mode

- Identifier is a Non-Transferable Basic Prefix
- Equivalent to did:key
- Create: for practical purposes, generation or knowledge of a private key (ICP event could be used as proof of control (being signed) but may not be considered necessary)
- Read: transform the Prefix into a DDO containing the key
- Update: none
- Delete: none

Direct Mode

- Equivalent to did:peer
- Create: broadcast ICP event to peer
- Read: process received or cached KEL
- Update: broadcast additional Key Events
- Delete: rotate to a null key set (termination)

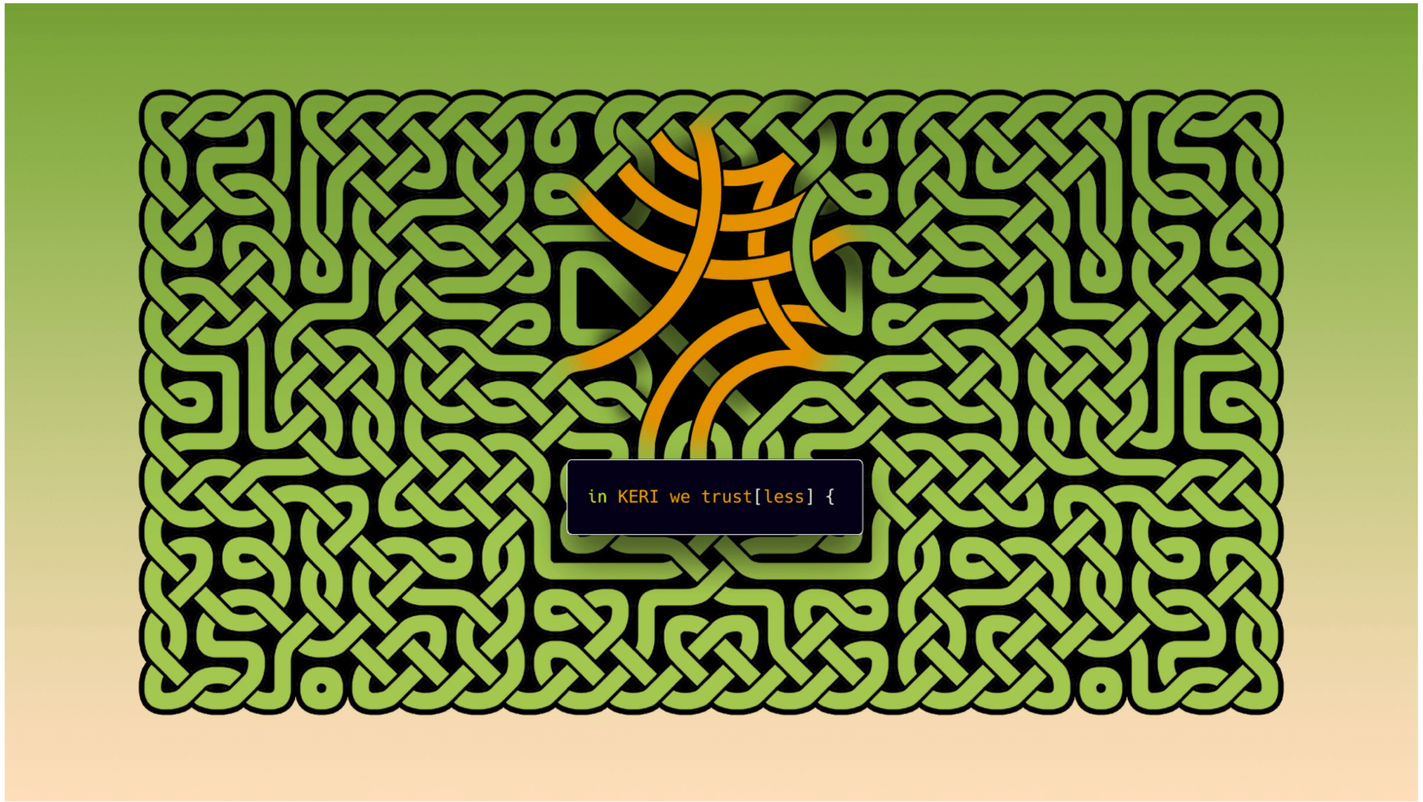
Indirect Mode

- Similar to many “global” DID Methods
- Discovery and Availability are not defined, or may be defined in a variety of ways
- Indirect:Direct :: Sidetree:peerDids
- Data model is still unified
- Create: broadcast an ICP event to your chosen replication mechanism (with witness receipts)
- Read: read from chosen replication mechanism (or cache)
- Update: broadcast new Key Events to chosen replication mechanism (with witness receipts)
- Delete: Update to null key set

Read on the Jolocom Logbook:

How KERI tackles the problem of trust

Tech dive • Oct 15, 2020 • logged_by: Charles



Rust Implementation of the KERI Core Library
by the Decentralized Identity Foundation

JOLOCOM

Charles Cunningham
charles@jolocom.com

jolocom.io