Decentralized Identifiers: Implications for Your Data, Payments and Communications

Impervious Technologies Inc. (impervious.ai)

3.21.2022

Abstract

Decentralized Identifiers (DIDs) are user-controlled, can be globally persistent and are generated or registered cryptographically. A DID consists of a Uniform Resource Identifier specified in and used to resolve a DID document. An entity with write-privileges called a controller inputs the specifications describing a DID subject into a DID document. The DID Specification enables a controller to improve DID resiliency by listing multiple service endpoints and transport pathways. These specifications and their resulting resilience form the basis of DIDCommunications (DIDComm).

DIDComm offers communication options regardless of protocol or transport layer, enabling users to communicate without hosting their own listening servers online. DIDComm mediators act as relays that receive an onion-encrypted message from a sender (or another previous mediator in a chain) and pass this message on to another party. Relays are transport agnostic and can be used with any desired transport layer, including Lightning, HTTP, TOR, webRTC or WebSockets. DIDs make reliable direct payments possible by abstracting an identity layer from the public address of Lightning nodes.

Through the DID Specification, service endpoints and DIDComm, Impervious has interlaced DIDs with Bitcoin Lightning, IPFS, WebRTC and resilient relays to introduce a new peer-to-peer internet standard with practical applications for mitigating censorship and surveillance risk.

Introduction to Decentralized Identifiers

Decentralized Identifiers (DIDs) are cryptographic and verifiable, do not require centralized registries or certificate authorities and can be globally unique and persistent. DIDs can be tailored for different contexts, used to reinforce pseudonymity or leveraged to port attributes between identities. This technology aims to solve many ills on the internet today, from intermediary control of identity to surveillance at successive layers of the technology stack.

A DID can identify any subject (e.g., person, organization, data model, abstract entity, etc.) based on its controller, which has DID document write-privileges. A DID document specifies the syntax, common data mode, core properties, serialized representations, operations and an explanation of the DID resolution process. A DID is represented by a Uniform Resource Identifier (URI) for trustable interactions between a DID subject and a DID document. A DID is verified by associating a DID document and URI and can be universally resolvable with DID methods such as the Microsoft ION *Sidetree protocol <https://github.com/decentralized-identity/ion>*.

DIDs consist of a URI string used to resolve a DID document with information that describes the identity's subject.

DID Document

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
     "https://w3id.org/security/suites/ed25519-2020/
v1"
  ]
  "id": "did:example:123456789abcdefghi",
  "authentication": [{
    // used to authenticate as did:...fghi
    "id": "did:example:123456789abcdefghi#keys-1",
    "type": "Ed25519VerificationKey2020",
    "controller": "did:example:123456789abcdefghi",
                                "publicKeyMultibase":
"zH3C2AVvLMv6gmMNam3uVAjZpfkcJCwDwnZn6z3wXmqPV"
  }]
}
```

A DID document may include authentication keys, service endpoints, a short-name alias, free-form text and other attributes as specified by the controller. Use cases can also be implemented without a protocol modification by updating a DID document.

The URI resolves to a DID document based on the implementation of the specific DID method. This infrastructure could be a web server (similar threat model to current URL API calls), peer-to-peer discovery, anchor to the Bitcoin blockchain, IPFS or custom implementation devised by the controller. The DID Specification enables a controller to add or update information within a DID document, improving the utility of a DID and enabling DIDCommunications (DIDComm).

The DID Specification

The DID Specification enables a controller to maximize decentralization by inputting additional information into a DID document. It meets the core attributes of an optimal decentralized payment and communications network. The DID Specification is updatable and open-standard and is redundant and resilient with fallback nodes, fallback on-chain addresses and alternative pathways to receive data or payments through a custodian. A DID is a static string that can function as a QR code for ease of use.

A controller may specify service endpoints in a DID document by listing the applications, features and functions a DID can support. The DID Specification for service endpoints allows for choosing the transport path, message type and method used to perform payment actions. A network of users may also select a subnet configured for their particular needs.

DID Service Endpoints

Service endpoints can be application-specific data stores (e.g., Tweets), IPNS links or any other protocol. Moreover, a DID can support several specified service endpoints. A DID document specifies the broadcast method and claims ownership over a personal domain, as shown in the following code example:

DID Document Specifies a Service Endpoint

```
{
"service": [
    {
      "id":"did:example:123#free",
      "type": "LinkedDomains",
      "serviceEndpoint": {
          "origins": ["https://free.example.com", “https://
identity.foundation"]
      }
    },
    {
      “id":"did:example:123#dom",
      "type": "LinkedDomains",
      "serviceEndpoint": "https://dom.example.com"
    }
  ]
```

A DID document must include all metadata necessary for the respective service endpoint type. Service endpoints can be duplicated and hosted across multiple providers (URLs) for enhanced resilience. A controller can specify which service endpoint receives message information, either through code or configuration. See the W3C DID Specification Registries (working) for a standardized list of service endpoints and supporting appropriate documentation:
https://www.w3.org/TR/did-spec-registries/#service-types.

Service endpoints are not only an example of the utility and flexibility that DID documents provide, but also a critical characteristic of higher-order decentralized applications that rely on DIDComm.

DIDComm

DIDComm is a secure, private communications layer built on top of the decentralized properties of a DID. DIDComm Messaging accrues the decentralization benefits to a network of connected identifiers with a particular set of specifications. While a range of secure messaging protocols exists, most require trusting a centralized authority to issue identities, register keys and delegate certificate authorities. Furthermore, most legacy messaging protocols rely on a single transport mechanism. Centralized systems often sacrifice security to gain convenience and are limited by a lack of interoperability. DIDComm Messaging relegates centralized messaging protocols and dependent systems.

DIDComm uses service endpoints and public keys located in the DID document to enable communication between DIDs. Signing, authentication and authorization are derived from a DID and are therefore inherently decentralized.[1] DIDComm optimizes for resilience and utility while providing individual discretion of the preferential communication protocol or transport layer. DIDComm messages are easy to understand as they utilize the same message formatting and parsing as a DID document. The receiver can set priorities to manage optimal message delivery.

In the following example, three target services are listed:

1. The lightning node pubkey, specified as LNPubkey to support Keysend and AMP.
2. A DIDComm Messaging service, listening for POST requests sent to the http://example.com/path.[2]
3. A DIDComm Messaging service over Lightning payments.[3]

[1] Decentralized Identity Foundation, https://identity.foundation/didcomm-messaging/spec/.

[2] The receiver can and should configure message protocols to eliminate spam.

[3] For Lightning based DIDComm, they may set a minimum Satoshi amount for message processing.

DIDComm

```
{
"service": [
    {
      "id":"did:example:123#LNPubkey",
      "type": "LNPubkey",
      "serviceEndpoint": "lightning:abc123..."
    },
    {
      "id": "did:example:123#didcomm-1",
      "type": "DIDCommMessaging",
      "serviceEndpoint": "http://example.com/path",
    },
    {
      "id": "did:example:123#didcomm-2",
      "type": "DIDCommMessaging",
      "serviceEndpoint": "did:example:123#LNPubkey",
    }
  ]
}
```

The specific DIDComm service endpoint above allows for receiving messages despite the sender not having access to a Lightning node. In addition, the receiver can set a priority as to which other service endpoint or communications method they prefer. This functional utility inherent in DIDs can be improved by implementing mediators or relays to ensure offline communication and traffic resilience.

DIDComm Mediators: Introduction to Relays

DIDComm allows users to communicate without hosting their servers online. Mediators[4] receive an onion-encrypted message from a sender (or a previous mediator in the chain) and relay this message to another party. A mediator cannot assess whether the recipient is another relay or the intended recipient, and they cannot access message content since it is encrypted with the recipient's public key. Delivery methods are transport agnostic and should be coordinated between the receiver and the designated mediator. In addition, a mediator can push mobile or email notifications or store the encrypted messages and wait for the receiver to fetch them directly when they are offline.

[4] https://identity.foundation/didcomm-messaging/guide/#mediators-and-relays

```
{

"service": [
    {
      "id": "did:example:123#didcomm-1",
      "type": "DIDCommMessaging",
      "serviceEndpoint": "did:example:somemediator",
    },
  ]
}
```

DIDComm mediator properties enable evolutionary advances in secure and decentralized communications and in Bitcoin Lightning payments by securely routing messages and payments to recipients, even if they are temporarily offline.

DIDComm as a Lightning Transport Method

Lightning transports data via IP or TOR TCP sockets, and payments are routed by pubkeys, which are gossiped across the Lightning network. DIDs and DIDComm-based messaging *can* be added as alternative ways to communicate with nodes. Additionally, WebSockets, mediators, pubkey swaps, fallback nodes or communication servers can be implemented native to the network. As a transport option for Lightning payments, DIDComm improves adaptability and future-proofs the protocol. The following sections discuss how IP/TOR/Pubkeys can be referenced inside a DID to replicate Lightning features and make these resilient.

Keysend/AMP

Keysend and AMP allow for ad hoc payments by sending them directly to the destination node's public key. This payment method does not require an invoice. In the example below, the serviceEndpoint URI is the lightning prefix followed by the node pubkey.

```
{
"service": [
    {
   "id":"did:example:123#LNPubkey",
      "type": "LNPubkey",
      "serviceEndpoint": "lightning:abc123..."
    }
  ]
}
```

LNURL

Lightning URL (LNURL) is a service endpoint inside the DID document. The benefit of abstracting LNURL—also an abstraction layer—is that it does not need to solely depend on IP addresses or domain names. The LNURL location can be updated as needed without affecting payers. In the example above, there could be a list of multiple origins for LNURL.

In the next example, the serviceEndpoint URI is the lightning prefix followed by the LNURL encoded string <https://github.com/fiatjaf/lnurl-rfc/blob/luds/06.md>.

```
{
"service": [
    {
"id":"did:example:123#lnurl-pay",
      "type": "LNURL-Pay",
      "serviceEndpoint": "lightning:LNURL..."
    }
  ]
}
```

Offers

Offers are improved static invoices that enable ad hoc payments through a generic, non-amount-dependent bolt12 payment string <https://github.com/lightning/bolts/pull/798>. To support offers, recipients can simply create a bolt12 payment string and then add it as a service endpoint. The serviceEndpoint URI consists of the lightning prefix followed by a bolt12 encoded string.

```
{
"service": [
    {
      "id":"did:example:123#LNOffers",
      "type": "LNOffers",
      "serviceEndpoint": "lightning:lno1qcp...."
    }
  ]
}
```

Custodians

Custodians add convenience to the secure and decentralized attributes of DID mediators or relays. DID mediator properties enable a standard for custodians to accept Lightning deposits on behalf of a recipient. This protocol is similar to a banking wire transfer protocol, with recipient information attached via Type-Length-Variable (TLV). The front-end of applications that understand specific custodians can open the profile in another browser tab. Back-ends that understand the API for specific custodians can make the payment through these URLs.

```
{
"service": [
    {
      "id":"did:example:123#Strike",
      "type": "Strike",
      "serviceEndpoint": "https://strike.me/username"
    }
  ]
}
```

The serviceEndpoint URI links to the component of the custodian's code that provides a specific service to the recipient (e.g., message forwarding, Lightning deposit, other service endpoint). A controller specifies a particular custodian using the "type" argument in the "service" function. The type argument can be a URL, DID, or Lightning pubkey that has been delegated as the user's custodian. This can be particularly useful when a Lightning node is not always online but a friend or custodian is.

BIP47 and *Paynyms*

Paynyms allow on-chain payments to a pseudonymous username (i.e., "*paynym")*. A new address is generated using BIP47 each time a payment is made to preserve on-chain privacy and transaction security.

There are two options for interacting with paynyms. The first displays a specific BIP47 payment code directly to the payer, while the second stores the paynym URL in a public directory.

For BIP47, the serviceEndpoint URI is the prefix BTC followed by the BIP47 payment code. The serviceEndpoint URI is the paynym URL for a given user's registered paynym. Paynyms are a resilient and secure way to conduct payments, but another approach is to use a designated on-chain address.

```
{
"service": [
    {
      "id":"did:example:123#BIP47",
      "type": "BIP47",
      "serviceEndpoint": "BTC:PM8T..."
    },
    {
      "id":"did:example:123#Paynym",
      "type": "Paynym",
      "serviceEndpoint": "https://paynym.is/+username"
    }
  ]
}
```

On-Chain Address

A static Bitcoin address can also be provided as payment information. In this case, the serviceEndpoint is BTC followed by the address. When advanced functionality such as running a Lightning node becomes too complex, another choice is to use a normal Bitcoin address. There is wide support for sending and receiving bitcoin to a normal on-chain address. However, due to privacy implications when reusing addresses for multiple purposes, on-chain addresses should mainly be a fallback payment option.

```
 {
"service": [
    {
      "id":"did:example:123#Bitcoin",
      "type": "Bitcoin",
      "serviceEndpoint": "BTC:bc1..."
    }
  ]
}
```

Bitcoin Payment Metadata

For each service endpoint payment type, optional metadata can be added to specify a receiver's preferences. This can include the priority and the minimum/maximum number of Satoshis (sats).

Priority

For payment options, a proposal for how the recipient prefers to get paid is as simple as setting the parameter "priority": 1.

```
{
"service": [
    {
      "id":"did:example:123456789abcdefghi#LNPubkey",
      "type": "LNPubkey",
      "priority": 1,
      "serviceEndpoint": "lightning:abc123..."
    },
    {
      "id":"did:example:123456789abcdefghi#BIP47",
      "type": "BIP47",
      "priority": 2,
      "serviceEndpoint": "BTC:PM8T..."
    },
    {
      "id":"did:example:123456789abcdefghi#Bitcoin",
      "type": "Bitcoin",
      "priority": 3,
      "serviceEndpoint": "BTC:bc1..."
    }
  ]
}
```

Min/Max

A minimum/maximum sats amount can be specified by setting the parameter “minAmountSat": 1 and “maxAmountSat": 100000. Msat and BTC could be supported as well.

```
{
"service": [
    {
      "id":"did:example:123456789abcdefghi#LNPubkey",
      "type": "LNPubkey",
      "maxAmountSat": 1000000,
      "serviceEndpoint": "lightning:abc123..."
    },
    {
      "id":"did:example:123456789abcdefghi#BIP47",
      "type": "BIP47",
      "minAmountSat": 1000000,
      "serviceEndpoint": "BTC:PM8T..."
    },
    {
      "id":"did:example:123456789abcdefghi#Bitcoin",
      "type": "Bitcoin",
      "minAmountSat": 1000000,
      "maxAmountSat": 5000000,
      "serviceEndpoint": "BTC:bc1..."
    }
  ]
}
```

Lightning Communication

Several Lightning based communications applications rely on Keysend/AMP to attach messages to the payment TLV records. One existing issue for any decentralized communications method involves determining whether a specific Lightning node is online and listening. Service endpoints can be used to signal the Lightning node status. By resolving a recipient’s DID, a caller may learn the status, payment and messaging information for their desired Lighting node.

In the example below, the recipient has their desired Lightning node pubkey specified as LNPubkey and is listening to Sphinx and Impervious messages.

```
{
"service": [
    {
      "id":"did:example:123#LNPubkey",
      "type": "LNPubkey",
      "serviceEndpoint": "lightning:abc123..."
    },
    {
      "id":"did:example:123#Sphinx",
      "type": "Sphinx",
      "serviceEndpoint": "did:example:123#LNPubkey"
    },
    {
      "id":"did:example:123#Impervious",
      "type": "Impervious",
      "serviceEndpoint": "did:example:123#LNPubkey"
    }
  ]
}
```

The service endpoint is the method used to communicate with destinations by referencing the pubkey directly from within the DID document. The type identifies the application listening to these messages, each of which may require their own message protocol or TLV records according to that application's standard.

Lightning Mailboxes

DIDComm mediators provide an avenue for two parties to coordinate payments, even if one is temporarily offline by delegating a fallback:

> If a DID sender wants to pay via Lightning to a DID recipient who is currently offline but has a delegated mediator as a fallback, the sender can send a message to their mediator. This message would contain payment details such as "DID:xyz tried to pay you 10000 sats." Once they are online and pull this message from the mediator, the sender can contact DID:xyz to request payment again.

This method informs the recipient that a Lightning payment was missed. In addition, DIDComm mediators offer advanced functionality such that upon Lightning payment transmission to an offline node, this payment is routed to a receiver-designated mediator. Next, a secondary message is sent out of band (txt, email, etc.) alerting the recipient of a missed payment. The user is then able to bring their Lightning node online, which alerts the designated mediator to receive payment. DIDComm can use the Lightning Network as a transport layer for a pure Bitcoin Layer 2 experience.

Well Known DID and DID DNS

The LNURL protocol (https://lightningaddress.com) has been popularized as a method for human-readable email addresses for making payments. LNURLs utilize the well-known txt record of a domain in order to return a LNURL-pay compatible payload specific to a particular username under the /.well-known/ URI. A similar protocol exists for DIDs called the "Well Known DID" (https://identity.foundation/.well-known/resources/did-configuration/). Well Known DIDs provide human-readable addresses for payments and messages.

> The "alsoKnownAs" field (https://www.w3.org/TR/did-core/#also-known-as), which can be specified in a receiving party's DID document, can be used in conjunction with Well Known DIDs to prove the common identity of a DID and URL.

In addition, the IETF has proposed a DNS for DIDs (https://www.ietf.org/archive/id/draft-mayrhofer-did-dns-05.txt). This proposal is designed to achieve the same effect as Well Known DIDs and represents another competitor for a DID URI standard as well as a redundant service endpoint that the controller may include to make a DID document more robust.

> **Example:** _did.example.net. IN URI 100 10 "did:example:1234abcd"

The Impervious Browser: A Revolutionary Application of Decentralized Identity

DID is the cornerstone of the peer-to-peer web, and the Impervious Browser makes peer-DIDs accessible for everyone. These enable a range of functions, from messaging to video conferencing, live document editing and content monetization. The following two scenarios involve Impervious DIDComm between two hypothetical users (Alice and Bob).

In each scenario, Alice opens a chat box to message Bob. The Impervious Browser prepares a plaintext JSON message. The Impervious Daemon checks to see if Bob's DID document is saved on the Daemon itself, checks a public DID-resolver URL or as a fallback checks Microsoft ION. The Impervious Daemon accesses two pieces of information and then returns to Alice: a service endpoint for messages to be delivered to Bob and the public key that Bob's agent is using in the Alice-to-Bob relationship. Next, Alice's browser uses Bob's public key to encrypt the plaintext messages so that only Bob's browser with that particular DID can read it. Alice's browser also adds authentication with its own private key.

Scenario 1: Alice and Bob are online

> The browser arranges delivery to Bob directly, using either the Lightning Network, HTTP or TOR.[5] Bob's browser receives and decrypts the message and authenticates its origin using Alice's public key. Bob's browser searches for Alice's key and an appropriate service endpoint in her DID document. Bob's browser then routes a response by encrypting the plaintext message, adding authentication with his private key and coordinating delivery via the available transport method and service endpoint.

Scenario 2: Alice or Bob are offline

> Alice attempts to contact Bob based on the prioritized communication channel specified in Bob's DID document, but this fails because his server is offline. Therefore, the browser arranges delivery to Bob via a DID mediator (also known as a "relay"), using either the Lightning Network, HTTP or TOR. When Bob comes back online, his Impervious Daemon connects to the mediator and retrieves messages. The relay does not have access to message content, as it is encrypted to Bob's public key. In this case, the mediator acts as a mailbox for Bob. If Bob has a pre-negotiated service endpoint (i.e., email address, Lightning Address, etc.), the mediator network acts as a forwarding service.

At launch, the Impervious Daemon will employ DID Specification service endpoints, DIDComm and mediators to provide the core peer-to-peer and encrypted functionality of the Impervious Browser. In future Impervious Browser versions, DID Specification will improve the usability and resilience of the Lightning network. The novel use of DIDs covered in this paper addresses the potential of this technology to weave the fabric of a new peer-to-peer internet standard, enabling everyone to mitigate the risk of censorship and surveillance.

Complete Code Example

A DID document includes:

- Personal domain ownership
- Owning two nodes: the user's preferred method of payment under 1M sats
- An LNURL endpoint supporting pay requests
- Supporting bolt12 offers with a generic donation invoice
- BIP47/paynym as a privacy preserving fallback payment method
- A static bitcoin address as a fallback payment method after paynyms
- A custodial service as a last resort payment method
- Impervious and Sphinx running off both LN nodes

[5] TOR pending v.1.02 update. Expanding to WebRTC and WebSockets.

- Supporting DIDComm on a user-owned HTTP server, user-owned LN node and mediator HTTP relay when a user is offline

The following example shows a DID document utilizing all payment and communication service endpoints covered throughout this paper.

```
{
  "@context": [
    "https://www.w3.org/ns/did/v1",
  ],
  "id": "did:example:123456789abcdefghi",
  "service": [
    {
      "id": "did:example:123456789abcdefghi#blog",
      "type": "LinkedDomains",
      "serviceEndpoint": "https://myserver.com"
    },
    {
      "id":"did:example:123456789abcdefghi#LNPubkey",
      "type": "LNPubkey",
      "priority": 1,
      "maxAmountSat": 1000000,
      "serviceEndpoint": ["lightning:abc123...",
"lightning:xyz987..."]
    },
    {
      "id":"did:example:123456789abcdefghi#lnurl-pay",
      "type": "LNURL-Pay",
      "serviceEndpoint": "lightning:LNURL..."
    },
    {
      "id":"did:example:123456789abcdefghi#LNOffers",
      "type": "LNOffers",
      "serviceEndpoint": "lightning:lno1qcp...."
    },
    {
      "id":"did:example:123456789abcdefghi#BIP47",
      "type": "BIP47",
      "priority": 2,
      "serviceEndpoint": "BTC:PM8T..."
```

```
},
{
  "id":"did:example:123456789abcdefghi#Paynym",
  "type": "Paynym",
  "priority": 2,
  "serviceEndpoint": "https://paynym.is/+username"
},
{
  "id":"did:example:123456789abcdefghi#Bitcoin",
  "type": "Bitcoin",
  "priority": 3,
  "serviceEndpoint": "BTC:bc1..."
},
{
  "id":"did:example:123456789abcdefghi#Strike",
  "type": "Strike",
  "priority": 4,
  "serviceEndpoint": "https://strike.me/username"
},
{
  "id":"did:example:123456789abcdefghi#Sphinx",
  "type": "Sphinx",
  "serviceEndpoint": "did:example:123#LNPubkey"
},
{
  "id":"did:example:123456789abcdefghi#Impervious",
  "type": "Impervious",
  "serviceEndpoint": "did:example:123#LNPubkey"
},
{
  "id": "did:example:123456789abcdefghi#didcomm-1",
  "type": "DIDCommMessaging",
  "serviceEndpoint": "http://example.com/path",
  "priority": 1,
  "accept": [
     "didcomm/v2",
     "didcomm/aip2;env=rfc587"
   ]
},
{
      "id": "did:example:123456789abcdefghi#didcomm-2",
      "type": "DIDCommMessaging",
      "serviceEndpoint": "did:example:123#LNPubkey",
      "priority": 2,
      "minAmountSat": 100,
      "accept": [
         "didcomm/v2",
         "didcomm/aip2;env=rfc587"
       ]
    },
    {
      "id": "did:example:123456789abcdefghi#didcomm-3",
      "type": "DIDCommMessaging",
      "serviceEndpoint": "http://somemediator.com/path",
      "priority": 3,
      "accept": [
         "didcomm/v2",
         "didcomm/aip2;env=rfc587"
       ],
       "routingKeys": ["did:example:somemediator#somekey"]
    }
  ]
}
```

See Impervious Technologies Inc.'s Github to reference example code snippets:

Impervious Technologies: Decentralized Identifiers ("DID") Spec

https://github.com/imperviousai/specs/blob/main/did-usage.md