DIDComm and the Self-Sovereign Internet

Summary

DIDComm is the messaging protocol that provides utility for DID-based relationships. DIDComm is more than just a way to exchange credentials, it's a protocol layer capable of supporting specialized application protocols for specific workflows. Because of its general nature and inherent support for self-sovereign relationships, DIDComm provides a basis for a self-sovereign internet much more private, enabling, and flexible than the one we've built using Web 2.0 technologies.

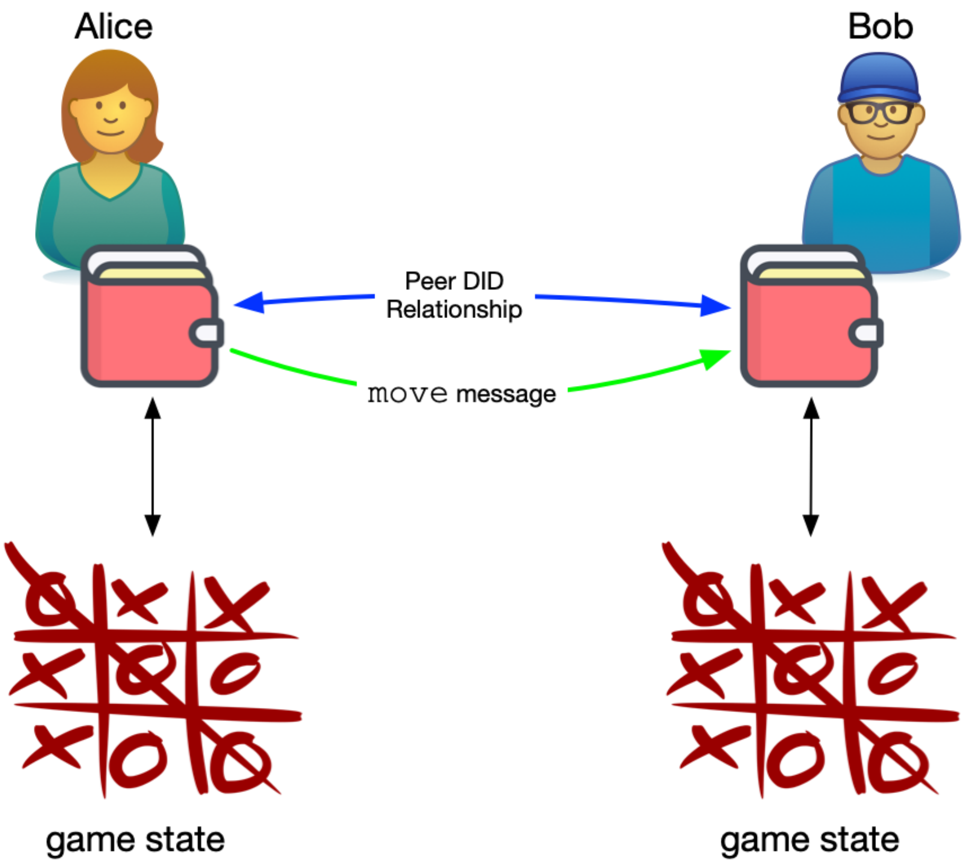
DID-based relationships are the foundation of self-sovereign identity (SSI). The exchange of DIDs to form a connection with another party gives both parties a relationship that is self-certifying and mutually authenticated. Further, the connection forms a secure messaging channel called DID Communication or DIDComm. DIDComm messaging is more important than most understand, providing a secure, interoperable, and flexible general messaging overlay for the entire internet.

Most people familiar with SSI equate DIDComm with verifiable credential exchange, but it's much more than that. Credential exchange is just one of an infinite variety of protocols that can ride on top of the general messaging protocol that DIDComm provides. Comparing DIDComm to the venerable TCP/IP protocol suite does not go too far. Just as numerous application protocols ride on top of TCP/IP, so too can various application protocols take advantage of DIDComm's secure messaging overlay network. The result is more than a secure messaging overlay for the internet, it is the foundation for a *self-sovereign internet with all that that implies*.

DID Communications Protocol

DIDComm messages are exchanged between software agents that act on behalf of the people or organizations that control them. I often use the term "wallet" to denote both the wallet and agent, but in this post we should distinguish between them. Agents are rule-executing software systems that exchange DIDComm messages. Wallets store DIDs, credentials, personally identifying information, cryptographic keys, and much more. Agents use wallets for some of the things they do, but not all agents need a wallet to function.

For example, imagine Alice and Bob wish to play a game of TicTacToe using game software that employs DIDComm. Alice's agent and Bob's agent will exchange a series of messages. Alice and Bob may be using a game UI and be unaware of the details but the agents are preparing plaintext JSON messages[1] for each move using a TicTacToe protocol that describes the format and appropriateness of a given message based on the current state of the game.



Alice and Bob play TicTacToe over DIDComm messaging (click to enlarge)

When Alice places an X in a square in the game interface, her agent looks up Bob's DID Document. She received this when she and Bob exchanged DIDs and it's kept up to date by Bob whenever he rotates the keys underlying the DID[2]. Alice's agent gets two key pieces of information from the DID Document: the endpoint where messages can be sent to Bob and the public key Bob's agent is using for the *Alice:Bob* relationship.

Alice's agent uses Bob's public key to encrypt the JSON message to ensure only Bob's agent can read it and adds authentication using the private key Alice uses in the *Alice:Bob* relationship. Alice's agent arranges to deliver the message to Bob's agent through whatever means are necessary given his choice of endpoint. DIDComm messages are often routed through other agents under Alice and Bob's control.

Once Bob's agent receives the message, it authenticates that it came from Alice and decrypts it. For a game of TicTacToe, it would ensure the message complies with the TicTacToe protocol given the current state of play. If it complies, the agent would present Alice's move to Bob through the game UI and await his response so that the process could continue. But different protocols could behave differently. For example, not all protocols need to take turns like the TicTacToe protocol does.

DIDComm Properties

The DIDComm Protocol is designed to be

1. Secure
2. Private
3. Interoperable
4. Transport-agnostic
5. Extensible

Secure and private follow from the protocol's support for heterarchical (peer-to-peer) connections and decentralized design along with its use of end-to-end encryption.

As an interoperable protocol, DIDComm is not dependent on a specific operating system, programming language, vendor, network, hardware platform, or ledger[3]. While DIDComm was originally developed within the Hyperledger Aries project, it aims to be the common language of any secure, private, self-sovereign interaction on, or off, the internet.

In addition to being interoperable, DIDComm should be able to make use of any transport mechanism including HTTP(S) 1.x and 2.0, WebSockets, IRC, Bluetooth, NFC, Signal, email, push notifications to mobile devices, Ham radio, multicast, snail mail, and more.

DIDComm is an asynchronous, simplex messaging protocol that is designed for extensibility by allowing for protocols to be run on top of it. By using asynchronous, simplex messaging as the lowest common denominator, almost any other interaction pattern can be built on top of DIDComm. Application-layer protocols running on top of DIDComm allow extensibility in a way that also supports interoperability.

DIDComm Protocols

Protocols describe the rules for a set of interactions, specifying the kinds of interactions that can happen without being overly prescriptive about their nature or content. Protocols formalize workflows for specific interactions like ordering food at a restaurant, playing a game, or applying for college. DIDComm and its application protocols are one of the cornerstones of the SSI metasystem, giving rise to a protocological culture within the metasystem that is open, agentic, inclusive, flexible, modular, and universal.

While we have come to think of SSI agents being strictly about exchanging peer DIDs to create a connection, request and issue a credential, or prove things using credentials, these are merely specific protocols defined to run over the DIDComm messaging protocol. Many others are possible. The follow specifications describe the protocols for these three core applications of DIDComm:

There's a protocol for agents to discover the protocols that another agent supports. And another for one agent to make an introduction[4] of one agent to another. The TicTacToe game Alice and Bob played above is enabled by a protocol for TicTacToe. Bruce Conrad who works on picos with me implemented the TicTacToe protocol for picos, which are DIDComm.

Daniel Hardman has provided a comprehensive tutorial on defining protocols on DIDComm. We can imagine a host of DIDComm protocols for all kinds of specialized interactions that people might want to undertake online including the following:

- Delegating
- Commenting
- Notifying
- Buying and selling
- Negotiating
- Enacting and enforcing contracts
- Putting things in escrow (and taking them out again)
- Transferring ownership
- Scheduling
- Auditing
- Reporting errors

As you can see from this partial list, DIDComm is not just a secure, private way to connect and exchange credentials. Rather DIDComm is a foundation protocol that provides a secure and private overlay to the internet for carrying out almost any online workflow. Consequently, agents are more than the name "wallet" would imply, although that's a convenient shorthand for the common uses of DIDComm today.

A Self-Sovereign Internet

Because of the self-sovereign nature of agents and the flexibility and interoperable characteristics they gain from DIDComm, they form the basis for new, more empowering internet. While self-sovereign identity is the current focus of DIDComm, its capabilities exceed what many think of as "identity." When you combine the vast landscape of potential verifiable credentials with DIDComm's ability to create custom message-based workflows to support very specific interactions, it's easy to imagine that the DIDComm protocol and the heterarchical network of agents it enables will have an impact as large as the web, perhaps the internet itself.

Notes

1. DIDComm messages do not strictly have to be formatted as JSON.
2. Alice's agent can verify that it has the right DID Document and the most recent key by requesting a copy of Bob's key event log (called delta's for peer DIDs) and validating it. This is the basis for saying peer DIDs are self certifying.
3. I'm using "ledger" as a generic term for any algorithmically controlled distributed consensus-based datastore including public blockchains, private blockchains, distributed file systems, and others.
4. Fuse with Two Owners shows an introduction protocol for picos I used in building Fuse in 2014 before picos used DIDComm. I'd like to revisit this as a way of building introductions into picos using a DIDComm protocol.

Photo Credit: Mesh from Adam R (Pixabay)

Please leave comments using the Hypothes.is sidebar.
Last modified: Thu Jul 1 10:14:55 2021.

Share This Article

2

61