This doc: https://bit.ly/ppred-doc; see also https://bit.ly/3SlsB1P

Privacy-Preserving Discovery (PPreD)

Daniel Hardman, first described in early 2017, updated March 2020

Background

The decentralized identity systems we want to build need privacy, but they also need discoverability. These two goals are sometimes in conflict. If we record everybody's PII in a giant registry in the sky, it's easy to find, but not very private. If we operate in stealth mode, nobody can find us.

This paper is about a possible, partial resolution of that tension. It is based on the insight that normal discovery is passive (you get discovered without any effort on your part) — but that it could be built to require your active participation instead.

We'll first explore the discovery problem from the perspective of *endpoint* discovery — finding a way to talk to someone without knowing the full route that gets to them. Later, we'll explore how to apply the technique to other discovery problems, and look at the consequences for regulations like GDPR.

Discovering a route

A simple way to talk to participants in a decentralized identity ecosystem would be to access them at a well known URI: https://identityhosting.infrastructureprovider.com/identity123. This is the purpose of endpoints in DID docs. It works well when the privacy of the owner behind a host is not vital (e.g., for a public institution). However, it presents a serious problem in some cases. Imagine what would happen if a pseudonymous political dissident's endpoint could be traced back to the infrastructure that hosted it, and if the infrastructure's billing records could then be hacked to determine the billing address of the person who uses it.

What we would like is a protocol that is similar to onion routing, that allows an arbitrary party to initiate a conversation with another arbitrary party such that:

- Each party knows the other's identifier, but nothing else. Specifically, neither party knows the route or endpoint for the other.
- No other parties along the route between the two endpoints know the full route, either.

Onion routing in mix networks doesn't do this, because it assumes a publicly routable target as input. It then interposes indirection to anonymize only the *sender*.

Privacy-Preserving Discovery (PPreD, pronounced /PEE pred/ to rhyme with “seabed”) is a solution to this problem. With PPreD, it is possible to use a URI like ppred://identity123 to talk to an identity owner in a way that preserves both sides’ privacy from each other and from all observers. This mechanism is somewhat expensive to use, relative to the cost of communicating with a known party over a stable onion route--but it is scalable and relatively practical to implement.

PPreD Address Space

The type of identifier that PPreD can use to derive a private route is called a decentralized identifier (DID). In this discussion, I’ll assume that DIDs are 16-byte, world-unique values with the same size and entropy as randomly generated UUIDs. Other sizes of DIDs could use this same mechanism; I’ll leave adaptation of the mechanism to those sizes as an exercise for the reader.

Given a DID size of 16 bytes, the DIDs can be represented in 32 hex digits, or 8 hex digit quads:

3293	31d1	7d2d	4dd1	9427	eb3f	aac9	3769

Each quad in a DID can have 2^16 or 65536 possible values. The set of all possible values `0000` through `FFFF` is called the quad address space.

PPreD Rendezvous Nodes

Posit a large network (thousands) of nodes that facilitate PPreD. These could be nodes that participate in a blockchain, since decentralized identity talks about blockchain a lot--but there’s no particular relationship to a blockchain. These “rendezvous” nodes are not considered especially trustworthy and are not entrusted with secrets, but they do donate their services to the larger community. Their function is analogous to a neutral meeting place that human messengers might use as a rendezvous point in the physical world.

When a rendezvous node initializes, it claims at random a range from the quad address space that begins with any number between 0 and 65535, and that represents 1/64th of the quad address space--for example, 18BF through 1CBE, inclusive. It then registers its availability[1] and its range so it can be used by the community.

[1] The PPreD service node registry could be hierarchical, like DNS, but it would have to be dynamic as well. Machines that participate in the registry are assumed to be relatively long-lived--but the registry needs to support some amount of turnover. The details of this registry are not discussed here.

Advertising

When Alice wants to communicate with Bob, she starts with Bob's PPreD URI (his DID with a little decoration). Suppose it is `ppred://329331d17d2d4dd19427eb3faac93769`. She breaks this URI into 8 quads. She then looks up[2] rendezvous nodes in the rendezvous registry that have claimed ranges of the address space that includes each quad. For example, she would need to find a node that services the quad `3293`, and another node that services the quad `31d1`, and so forth. Lookup might return multiple nodes that service the range, in which case Alice can select one at random. At the end of this lookup process, Alice has chosen 8 rendezvous nodes that she wants to use.

Alice then contacts each of these 8 rendezvous nodes with a message that has a formal structure that we'll ignore for the moment. The message essentially says: "I am interested in talking to an ID with `3293` as one of its quads. Here is an encrypted version of my ID, unlockable by the correct recipient, and here are several pre-computed onion routes that get from you (this rendezvous node) back to me [where the routes are encrypted by the target ID's public key so they are opaque to the rendezvous node]. Here is an encrypted nonce X' that, when decrypted by the proper recipient, will produce the value X. The TTL for my request is Y."[3] The nonce in this message is encrypted using the public key for the target DID, as declared in a DID doc. The overall message is encrypted using the public key of the rendezvous node, so that only the rendezvous node can read it.

The rendezvous server records this connection request and keeps track of it until it expires.

Each rendezvous node is running a gossip protocol that shares this request with other rendezvous nodes having a compatible range, re-encrypting the message using each peer node's public key as appropriate. This makes knowledge of the request propagate, and it makes the final location of a successful rendezvous unpredictable. The amount of gossip needs to be tuned.

Polling

Meanwhile, Bob is polling some randomly selected rendezvous servers that match certain quads in his identifier. He may poll at whatever interval he chooses. For example, he may poll once per minute, or he may set rules about polling for himself such as "I only check for new contacts on Thursdays between noon and 1 pm." He doesn't need to tell anybody his schedule.

---

[2] Lookup needs to include a proof-of-patience task or similar, so it can't be DDoS'ed. And so do other parts of the algorithm.

[3] Perhaps the message may contain other constraints on the communication request, such as a start time ("I don't want to talk until date X"), a proposed topic for the conversation, etc? Does this introduce security risk?

Bob's polling operation essentially asks each rendezvous server that interests him, "Is there anybody trying to talk to quad X?" (where X is whatever quad is compatible with the server he's asking, e.g., `3293`). If a rendezvous server knows about a pending request that matches, it asks Bob to prove that the message is meant for him. It might not be; lots of people might have registered for this quad because it was part of their legitimate DID. Bob doesn't have to respond to this proof request. Before he does, he might want to poll some other servers that are monitoring other quads relevant to his DID. Only if he sees that there's a pending request for *all* of his quads is it worth his time to engage further. (Even then, it might not be worth his time, because Alice might be trying to reach a DID that uses his quads in a different order…)

Assuming Bob detects a match for his DID, he distinguishes himself from malicious listeners or from innocent listeners that weren't Alice's real target by presenting a proof that he can decrypt the nonce X' and produce X. Only the possessor of Alice's target DID can do that. If Bob is successful, the rendezvous server asks Bob for one or more onion routes that get from it to Bob, where the routes should be communicated in a blob encrypted by Alice's public key. At this point the rendezvous server knows that two parties should connect, and it holds at least one halfway route between it and each party, but it does not know either party's identity, and the routes it is holding are opaque to the rendezvous server (since the were encrypted by Bob for Alice, and by Alice for Bob).

Connecting

The rendezvous server now builds payloads announcing the connection to both parties. Bob receives a message announcing that Alice wants to connect, with an onion route that gets from Bob to Alice (combining info provided by both parties); Alice gets the opposite. This message is sent along all the routes that either party provides, to account for the possibility that some of the routes have been invalidated in the meantime. (Or, each route is tested by the rendezvous server until one works, and then that route is the one that it selects?)

Once Alice and Bob connect, they can negotiate a persistent rendezvous point (which might be hosted on a rendezvous server), or they can maintain a session as long as they want, then tear it down and build a new one next time they wish to speak. At some point, if Bob and Alice trust one another, they can use a non-ephemeral endpoint for communication. Such endpoints can be rotated using a scheme that Bob and Alice agree upon during their first secure and private session, allowing efficiency without the overhead of repeated discovery.

A malicious rendezvous server cannot substitute routes, because the communication between Alice and Bob is signed by public keys that are known to belong to the identities in question. It never knows the identity of either party, or the full route to either party.

Likewise, malicious parties cannot spy on a rendezvous server and learn anything particularly useful.

Net effect

The overall consequence of this protocol is that a party can be contacted at a public identifier without anybody in the world knowing how to route to the party directly. PPreD is like an anonymous rendezvous in a public place, where both parties guarantee that they are not followed by a tail on the way there or on the way back. And it goes one step further, because any interested party can show up at the rendezvous site unilaterally, not pre-announced--and if the other party is open to being contacted, communication can be bootstrapped.

PPreD is a relatively expensive protocol, too inefficient for routine use. However, there are cases where privacy is crucial, and it is worth the overhead of this protocol to make strong guarantees.

Beyond Endpoints

This same principle — requiring the active participation of a data holder to complete discovery — can be used to facilitate discovery of other things besides Bob's endpoint. One particularly interesting example is that it could be used to discover the registration of DID values themselves.

Before describing how this would work, let me comment about how this differs from the sort of passive enumeration that could be done against a phone book or traditional directory. If DIDs are publicly registered on a blockchain, then the blockchain probably offers some sort of transaction enumeration that makes discovery easy. That's passive discovery, and in immutable blockchains it has the GDPR problem that right-to-be-forgotten is really hard to support.

But if the DIDs are off-chain, or if their method doesn't allow querying for arbitrary values, then what's described here could be useful — and a value-add over and above what would be possible with the simple endpoint discovery that we first described. Given thousands of rendezvous nodes and millions of registered DIDs, it's likely impractical for Alice to enumerate all registered DIDs by querying rendezvous nodes for all their registered quads. Even if she could do that, the results wouldn't give her a definitive set of DIDs, since registration doesn't include information about quad ordering, and nothing stops people from doing spurious registrations to introduce noise. So the form of discovery we'd do is discovery of the answer to a very narrow question: *Does the owner of DID value X want to be discovered?* It's not discovery of whether the DID exists.

So here's how discovery of a single registered DID value would work. We posit registration of millions of Bobs as described earlier. However, we imagine that Alice starts with different information. Instead of beginning with a DID that she resolved to get its public key, so she could talk to it, we posit that Alice has a theory that DID value X might be registered. She has no key material for it.

She now crafts a different message to the rendezvous nodes. Instead of, "I want to be routed to X, and here's an encrypted nonce that X must prove it can decrypt properly, and here's an encrypted route back to me", Alice's message says, "I want to know if X is registered. Please have X contact me at DID Y, if the answer is yes." Besides sending this message, Alice may also register her DID Y with the rendezvous nodes, so she can be contacted via rendezvous by Bob. Alternatively, Alice may make DID Y a traditionally, publicly registered DID if she likes.

Information about this message now percolates through the system, and Bob learns about Alice's intentions. If he decides he wants Alice to know about his DID's registration, he can either turn around and run the endpoint discovery protocol on her DID (Y), or he can reach out to her public endpoint. The latter would degrade his privacy; he could go through a mix network if he liked. But either way, he has to share with Alice a proof that he controls the DID in question. This means his DID, X, has to become resolvable to Alice, and he must sign or encrypt his response to her using a key from DID X's DID doc such that she becomes confident she's talking with X's true controller.

Regulatory Compliance

What's interesting about this mechanism from a regulatory compliance perspective is that the registration is mutable. It doesn't need blockchain features, and it doesn't have to be supported by trustworthy nodes. Nobody is ever registering their full DID anywhere; rather, they are registering possibly true, possibly false interest in particular portions of a DID which, taken in the aggregate, allow only the holder of the real DID to confirm its existence, its routability, etc.