How DIDs, Keys, Credentials, and Agents Work in Sovrin

sovrin
identity for all

This document shows how low-level building blocks of the Sovrin ecosystem function in a practical real-world scenario where key management concerns matter deeply. It also introduces some notation and terminology. It makes heavy use of concepts described in the formal sequence diagrams and exposition from **DKMS Design and Architecture V2**.

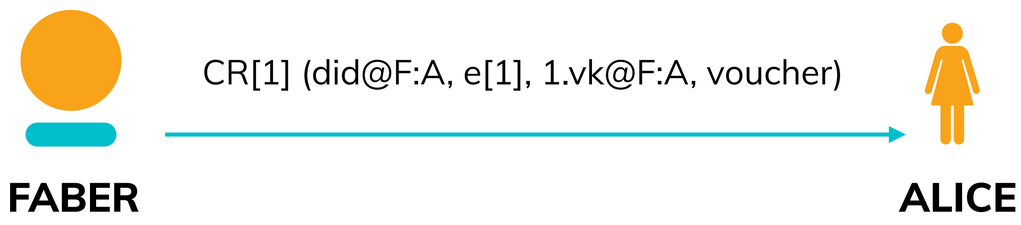
Daniel Hardman
April 2018

Prerequisites

Please make sure you are familiar with the terminology in **A Brief Overview of SSI Core Concepts**, plus the notation in **SSI Notation Conventions**, before proceeding. Also, please review the primer [**recording**, **slides**] on secure agent-to-agent communication in the Sovrin ecosystem.

Scenario

Imagine Alice, a private citizen, is invited to join the Sovrin ecosystem by Faber College (where she is an alumna). She accepts the invitation and is onboarded. Then she receives a digital diploma (a Sovrin-style ***credential***) from Faber. Later she receives another credential--a digital driving license issued by her government. Alice owns a mobile device and a laptop, and she establishes a connection and proves things to a potential employer, Consoto. Eventually, her phone will be stolen by Mallory, and Alice will have to recover.



Onboarding

Alice (denoted by A) begins her participation in the Sovrin ecosystem when she receives a ***connection request*** (denoted by `CR[1]`) from Faber College (denoted by `F`). The request conveys the following information to Alice:

- The fact that a party purporting to be Faber is requesting a connection to Alice. Alice does not need to trust that the remote party is truly Faber, but we will assume this is the case in order to simplify the discussion.
- A new DID (`did@F:A`) which Faber has allocated for itself, and by which it proposes to be known in the Faber:Alice relationship.
- The URL of an endpoint, `e[1]`, at which Alice may contact the sender if she chooses to accept the connection request. This endpoint is serviced by Faber's **edge agent** (`1@F`).
- A public Ed25519 verification key (`1.vk@F:A`) that will be used by `1@F` in the context of the `F:A` relationship to authenticate all messages as originating with `1@F`. (Faber does NOT share the private signing key complement, `1.sk@F:A`.)
- Optionally, a voucher for cloud agent services at an **agency** (identified by the URL of the agency's general message delivery endpoint `e[2]`) that Alice can use if she likes.

How this connection request is transmitted to Alice will influence both the security of this process, and the sort of experience that Alice has when the request arrives. We assume here that Alice has no special knowledge of Sovrin prior to this communication, but that she and Faber have an existing communication channel that both consider acceptably trustworthy. For example, perhaps Alice receives SMS updates from Faber at a phone number she has previously shared, and recognizes the sending phone number as reliably associated with Faber--or perhaps she gets email from the alumni office--or perhaps Alice logs in to Faber's secure alumni web portal, and receives the request there.

Let's explore the latter case. Alice is browsing the alumni portal on her phone after login, and she sees an offer for a digital, cryptographically verifiable copy of her college transcript. She is intrigued, so she clicks the link. The portal sends her to an onboarding handler (any piece of code that facilitates the core mechanics of onboarding). The handler detects no Sovrin-capable app on her phone, so it constructs a custom app install link for Alice and displays a message saying, "To use your digital diploma, you'll need a free app. Click here to get the app and your diploma." Alice clicks the link. Her phone downloads, installs, and starts the app.

The app welcomes Alice and explains that it will help manage digital credentials and connections in a secure, private way. It creates an empty **wallet** on her phone, so it can store secrets, and ensures that she has Touch ID or Face ID enabled so this wallet will be protected.

This app (which is also an edge agent) is the first entity in Alice's **self-sovereign domain,** which encompasses all the devices, software, and data that she controls with Sovrin technologies. Control over a domain is exercised via cryptographic keys, and each entity in the domain needs its own agent (device) keys. Therefore the app creates an Ed25519 key pair for itself, and stores the two parts in Alice's wallet under the names `2.vk` (the public verification key belonging to Alice's edge agent, numbered 2 because it's the second agent introduced in the present discussion) and `2.sk` (the private signing key belonging to Alice's edge agent).

The app knows that Alice will be receiving digital credentials, and that these credentials will need to contain a blinded **link secret** to prove that all of them belong to the same person--so it generates a link secret, `ls@A`, and stores it in Alice's wallet as well.

The app also anticipates the need for other parties to know whether they are dealing with an Alice-approved agent as they interact with her domain, so it creates an **agent policy registry** for her. This policy initially says that agent 2 is authorized to represent Alice in various ways, such as proving things with her credentials, and making subsequent updates to the policy registry itself. The agent policy registry is associated with a **proving secret** that Alice's agent also has to allocate. More details on this later.

The app writes this policy to the Sovrin ledger, signing the write transaction using `2.sk`[1]. The ledger records this transaction--and atomically, it updates a ledger-based, global public oracle called the policy oracle. The design of the **policy oracle** is outside the scope of this doc, but it's important to understand its semantic properties: it can be used to generate cryptographic evidence that an agent presenting credential-based proofs is authorized by the owner of a specific self-sovereign domain to do that proving — *without disclosing the owner's domain or the agent in a correlatable way.* In other words, the oracle tells a relying party whether it should accept proofs, based on information originally expressed in an agent policy registry, without disclosing which agent policy registry was the basis for its decision. Neither the oracle nor the ledger knows the basis for the decision at the time of proving, either.

More initialization could be done at this point to prepare Alice for recovery, but such steps are better explained later on, so we'll assume the app skips them.

[1] Since this is a new agent policy registry, the ledger doesn't particularly care who signed the transaction. However, for subsequent writes of the same policy, the ledger would check the signature on the transaction against the existing version of the registry to make sure that the signing agent is authorized to do the update.

Connecting

With basic setup complete, Alice has been onboarded to Sovrin, but she is not yet connected to any party. The app has been configured to return to the onboarding handler on first run; it does so. The handler now sends the connection request as a Sovrin message, and Alice's new app pops up a new message: "A party calling itself 'Faber College' proposes to connect with you. Accept?"

Alice says yes.

The app must now build up Alice's side of the Alice:Faber relationship. First it creates a new DID, `did@A:F`, and a new key pair, `2.vk@A:F` and `2.sk@A:F`[2]. It stores these three values in its wallet, along with Faber's corresponding DID, `did@F:A`.

A Detour for a Cloud Agent

Before sending a response to Faber, Alice probably needs a secure, private way for Faber to contact her even when she is not signed in to the Faber alumni portal. Ideally, this mechanism would be available even if Alice turns her phone off. Such a channel would be enabled by a **cloud agent** that runs 24x7--but since Alice has barely been onboarded, she doesn't yet have one. Happily, her connection request includes a voucher that she can redeem for a free cloud agent sponsored by Faber. Let's assume this cloud agent will be hosted at a third party provider of agents-as-a-service (an **agency**). We will denote this agency with G. Alice's app (her edge agent, 2) contacts the agency and establishes a relationship between A and G, as follows:

First, 2 generates a new DID+keypair **triplet** for the `A:G` relationship, and stores `did@A:G, 2.vk@A:G` and `2.sk@A:G` in its wallet. It then creates a connection request, `CR-2`, and contacts the agency URL, `e[2]`, that was given in `CR[1]`. The agency is configured to accept all connection requests from unknown parties that are backed by a voucher, so when it sees the voucher and request from 2, it allocates a new triplet for the `G:A` relationship and stores `did@G:A, 2.vk@G:A, 2.sk@G:A` -- plus Alice's DID, `did@A:G`, in its own agent's wallet. It then spins up a cloud agent for Alice, 4, assigns it the familiar endpoint e[2][4], and tells this agent to initialize and to prepare itself to service `did@A:G`, verified by `2.vk@A:G`. 4 now needs a wallet and keys of its own. It creates 4.wallet and a new keypair: `4.vk+4.sk`. It will use this keypair in its relationships with the ledger and Alice's other agents. In addition, it needs a non-correlating keypair that can be used in the context of the relationship representing its first serviced DID, `did@A:G`. So it creates and stores an extra key pair: `4.vk@A:F+4.sk@A:F`. It then sends a response to `2@A`, announcing its appointment as Alice's cloud agent, plus its verkey 4.vk, and the verkey it proposes to use for the A:F relationship `(4.vk@A:F)`.

At this point 2 must empower 4 to act to some degree on A's behalf. Let us assume that the cloud agent is trusted to route messages, but not to give legal consent or update Alice's agent policy registry. 2 constructs a new ledger transaction that updates A's agent policy registry, placing 4 in some authorization lists but not others, and causing the policy oracle to be updated. This transaction is accepted because it is signed by `2.sk`, which verifies using `2.vk`, which was already in the admin list of Alice's agent policy registry.

---

[2] In this notation, we know that 1 refers to Alice's first agent, rather than Faber's, because it looks at the Alice:Faber relationship from Alice's perspective `(1@A:F)` instead of from Faber's perspective `(1@F:A)`.

[3] It's likely that the agency will have terms of service and other disclosures that it needs to make to Alice, so this workflow could cause messages from the agency to display on Alice's device during the process. We are ignoring that detail for simplicity.

[4] One common misconception is that every agent should have its own endpoint. In fact, doing so would create a new correlation point. Rather, all agents at a given agency share a single endpoint which is basically a message dispatcher. Details about this message dispatcher are covered in the **primer on secure agent-to-agent communications.**

Alice's two agents now know about each other, and they are each appropriately empowered. Alice's cloud agent has a stable URL at which Faber can reach her. Alice also has a relationship with her new agency, G. When Alice receives subsequent connection requests, this detour to set up a cloud agent will be unnecessary. Although this detour may feel "heavy", it can all take place with in a second or two, with a click for user consent. Alice doesn't need to understand the details.

Back to Original Connection

Alice now uses her phone to construct a **connection response** that informs F about her acceptance of the connection request. This response includes:

- The DID and verkey she will use for the new connection: `did@A:F, 2.vk@A:F.`
- The endpoint at which Faber may contact her. This is generally `e[2]` (see footnote 4).
- Any other keys that Alice wants Faber to accept as empowered to participate in their relationship. In this case, the only key that qualifies is `4.vk@A:F.`

Alice (via her phone) sends this connection response to Faber at `e[1].` Faber records `did@A:F` and `2.vk@A:F` in its wallet next to `did@F:A, 1.vk@F:A` and `1.sk@F:A,` and the connection is established.

Issuing a Credential

Faber can now take advantage of the secure, private channel it has with Alice to negotiate the issuance of a credential representing Alice's college transcript.

Faber creates a **credential offer**. This offer identifies the definition of the proposed credential, including its schema (a "US Division 1 College Transcript v 2.0" as recorded in ledger transaction 293487), its issuer (Faber College), and its revocation mechanism (a particular accumulator with identifier XYZ, published on the ledger--plus a tails file with SHA256 hash 0xA1B2C...DEF downloadable at faber.com/transcripts/tails). It may also include a preview of the data that Faber proposes to include, a mention of a price, and terms and conditions that Alice is agreeing to if she accepts the offer.

Faber sends this credential offer to Alice (specifically encrypted for 2) via the message delivery endpoint `e[2]`; the message is addressed to `did@A:F`, which the agency knows to route to 4. Faber knows this is the correct place to drop a message for Alice, since her connection response said so.

The message is forwarded from 4 to 2. Alice's phone buzzes to notify her that Faber is offering her a transcript with such-and-such a description. Does she want to accept?

Alice says yes.

Alice now has to provide two pieces of data to Faber: a blinded version of her link secret, and blinded version of her proving secret. 2 embeds these pieces in its response, which is a message of type credential request that references the previous offer.

---

[5] Optionally, Alice could request a receipt from Faber, so she knows that Faber knows she has accepted..

[6] A "tails file" is a construct from Sovrin's anoncreds protocol. It is basically a giant list of random numbers that are used to construct a cryptographic witness; each issuer publicly publishes (at least) one such file for each credential type (but not each individual credential) it creates.

Faber responds in turn, with a message of type **credential**. This message contains Alice's transcript as a flat series of fields (e.g., student.first_name, student.last_name, issue_date, student.major, etc). Each field is represented in an intuitive format appropriate to its data type; fields that support zero-knowledge proving are also represented numerically. The credential is digitally signed by Faber (thus proving Faber as its issuer) and contains the two pieces of blinded information provided by Alice. It also contains a reference to its definition (template, not specific data) on the ledger, and thus information about its revocation mechanism.

Later, Alice will use this credential to generate proofs. Proofs will have the following characteristics:

- They may disclose some attributes, prove attributes about others, and ignore other attributes entirely.
- They are unique and context-specific; they cannot be reused in other channels besides the one between prover and relying party.
- They may prove attributes from multiple credentials, using the blinded value of the link secret (but not disclosing that value or its unblinded basis) to guarantee that each source credential belongs to the same owner.
- They include zero-knowledge proof that they have not been revoked.
- They include zero-knowledge proof the presenter of the proof is in the set of authorized proving devices pertaining to the identity owner.

At issuance, Faber must guarantee that the accumulator tracking revocation of its transcripts reports correct state for Alice's new credential. It could update the accumulator as it issues, but it is more efficient to pre-populate the accumulator with all foreseen issuances, and then only update the accumulator for revocations.

Faber must also communicate, as it issues its credential, a way for Alice to calculate the delta between her own contribution to the accumulator, and the contribution of all other issued credentials. This delta is called a **witness**. Over time, as more credentials are issued and revoked, the witness accumulates additional deltas. Alice must track (or be able to calculate) an updated witness for use in future proving. (For more detail about revocation mechanisms, **see this discussion**.)

Planning for Recovery

Alice is functional as a Sovrin participant, but our discussion up to this point has left Alice susceptible to some important problems:

- She might lose her phone.
- Her cloud agent might be hacked.

These contingencies can be handled by further configuration of Alice's sovereign domain.

First, Alice can add a second edge agent. She has a laptop. She can install Sovrin-capable software there; we can call this new edge agent 5. It will create a wallet and keys. Alice can add its keys to her agent policy registry. If she loses her phone, her laptop keeps her functional--and vice versa.

Second, Alice can create a paper agent. This is an agent that stores state (wallet: keys, DIDs) on printed paper (perhaps using QR codes). Alice can print this paper out and store it in a safe place. She can add the keys of this paper agent to her agent policy registry, such that the paper agent is empowered to act on her behalf in all respects if other agents are unavailable.

Third, Alice can introduce ratio-based **policy qualifiers.** These are rules such as, "Allow my agent policy registry to be updated by any 2 of the following 3 keys". This can prevent any one hacked agent from having too much power.

Fourth, Alice can use Shamir secret sharing to shard additional agent keys that are also part of her agent policy registry. She can distribute shards to trusted friends who she trusts to use good judgment when asked to reconstitute the key.

Second Credential

Alice establishes a second Sovrin relationship with the Driver's License authority (denoted by D) of her government. The process of connecting is as before, except that she doesn't need a new cloud agent. Alice's DID and key material for this relationship, `A:D`, is entirely independent of the values she put in her agent wallets for the `A:F` and `A:G` relationships. She ends up with keypairs on `2 (2.vk@A:D, 2.sk@A:D)` and on `4 (4.vk@A:D, 4.sk@A:D)`. `2` and 4 know each other's verkeys, and D knows both verkeys plus `did@A:D`.

Alice does not need to make any updates to her agent policy registry as she establishes this new relationship.

Her new credential has a different revocation registry (new tails file, new accumulator, new witness), but it contains the same link secret and the same proving secret, blinded differently.

First Proof

Alice wants to apply for a job at Consoto (denoted by `C`). She establishes the `A:C` relationship in the same way she's built others. Once she has a secure, private channel, Consoto sends her a ***proof request***. This message asks her for cryptographically robust evidence that she is a college graduate, over 21, and a resident of the city where Contoso is based. It also asks her to disclose her legal name. It requires that the legal name come from her driver's license, and that the birthdate on which her age proof is based appear on her driver's license.

Alice's agent, `2`, generates this proof using her two credentials. The proof contains all of the following:

- Disclosed attributes: her legal name as embedded in her driving license.
- Predicates: college grad (from credential 1--transcript), over 21 based on birthdate from driving license (credential 2), resident of Consoto's city (postal code on driving license = postal code of Consoto).
- Links: special predicate showing that the link secret in credential 1, if unblinded, would be the same as the link secret in credential 2, if unblinded. No unblinding actually happens, but the proof occurs.
- Non-revocation: Alice proves that neither of her credentials has yet been revoked. This is a proof of set membership (that a special number issued to Alice with her credential, and known only to her and the issuer) is "in" a cryptographic accumulator published on the ledger.
- Authorization to prove on Alice's behalf: In zero knowledge, the agent/device generating this proof must demonstrate to the verifier that it possesses a key which is named in Alice's agent policy registry as being a valid prover on Alice's behalf. This is done by providing information that allows the verifier to consult the policy oracle.

Another Agent

Alice now needs to onboard her laptop. Suppose she does so, creating agent `5`. The details of how this is done are mostly left as an exercise for the reader, but we note that `5` gets its own keys, just like `2` and `4`, and that `5` must be empowered to act in Alice's behalf just as `2` and `4` are, by updating the agent policy registry.

sovrin
identity for all

Theft

Now imagine that Alice takes a ride in a taxi. As she exits, she inadvertently leaves her phone on the seat, and it falls into the hands of Mallory. Assume that Mallory disassembles the phone and pulls Alice's wallet off of raw storage, thus short-circuiting FaceID or TouchID. Further assume that after some fiddling, Mallory is able to unlock Alice's wallet. Mallory now has access to all of the secrets that agent 2 had. This means that Mallory can use Alice's credentials to generate proofs in the same way that Alice did when she possessed the phone. These proofs can be used in existing relationships; theoretically they could also be used to establish new relationships where Alice is impersonated.

Recovery

First, Alice gets on her laptop and uses it to update her agent policy registry. She removes the keys for 2 and writes a transaction to the ledger that causes the global policy to be updated. This prevents Mallory from generating any proofs that impersonate Alice, because such proofs would lack the authorization to prove on Alice's behalf (the policy oracle would no longer confirm authorization). It also prevents Mallory from updating Alice's policy registry instead. (If Alice has previously used policy qualifiers to require that updates to the policy registry use an M of N scheme, this de-authorization of 2 may not be particularly urgent.)

Now Mallory cannot use Alice's credentials anywhere, in existing or new relationships, and cannot lock Alice out of her sovereign domain. This is a fast change (seconds), and takes effect globally. Most of the damage from the theft is already blocked.

Second, Alice rotates the keys that she uses in all her existing relationships. This prevents Mallory from continuing message exchange over secure channels that Alice has built with any of her existing relationships.

Third, Alice generates a new master secret, and sets about getting credentials re-issued.

What is the Sovrin Foundation?

The **Sovrin Foundation** is a nonprofit organization established to administer the Trust Framework governing the Sovrin Network, a public service utility enabling self-sovereign identity on the internet. The Sovrin Network is operated by independent Stewards and uses the power of a distributed ledger for giving every person, organization and thing the ability to own and control their own permanent digital identity.

Contact Info:

Sovrin Foundation

Email: info@Sovrin.org

Phone: +1 801 701-1848