Getting Started with Sovrin

A Developer Guide from the Sovrin Foundation

image alt text

What Sovrin is, and Why it Matters

Sovrin is a software ecosystem for private, secure, and powerful identity. It puts people — not the organizations that traditionally centralize identity — in charge of decisions about their own privacy and disclosure. This enables all kinds of rich innovation: link contracts, revocation, novel payment workflows, asset and document management features, creative forms of escrow, curated reputation, integrations with other cool technologies, and so on.

Sovrin uses open-source, distributed ledger technology. These ledgers are a form of database that is provided cooperatively by a pool of participants, instead of by a giant database with a central admin. Data lives redundantly in many places, and it accrues in transactions orchestrated by many machines. Strong, industry-standard cryptography protects it. Best practices in key management and cybersecurity pervade its design. The result is a reliable, public source of truth under no single entity's control, robust to system failure, resilient to hacking, and highly immune to subversion by hostile entities.

If the cryptography and blockchain details feel mysterious, fear not:

getting traction to develop on Sovrin is easy and fast. You’re starting in the right place.

What We’ll Cover

Our goal is to introduce you to many of the concepts of Sovrin, and give you some idea of what happens behind the scenes to make it all work.

We are going to frame the exploration with a story. Alice, a graduate of the fictional Faber College, wants to apply for a job at Acme Corp. As soon as she has the job, she wants to apply for a loan so she can buy a car. She would like to use her college transcript as proof of her education on the job application; once hired, Alice would like to use the fact of employment as evidence of her creditworthiness for the loan. The sorts of identity and trust interactions required to pull this off are messy in the world today; they are slow, they violate privacy, and they are susceptible to fraud. We’ll show you how Sovrin is a quantum leap forward.

Ready?

Alice Gets a Transcript

As a graduate of Faber College, Alice receives an alumni newsletter where she learns that her alma mater is offering digital transcripts. She logs in to the college alumni website and requests her transcript by clicking a *Get Transcript* button. (Other ways to initiate this request might include scanning a QR code, downloading a transcript package from a published URL, etc.)

Faber College has done some prep work to offer this service to Alice. It has the role of **trust anchor** on Sovrin. A trust anchor is a person or organization that Sovrin already knows about, that is able to help bootstrap others. (It is *not* the same as what cybersecurity experts call a “trusted third party”; think of it more like a facilitator). Becoming a trust anchor is beyond the scope of this guide; for now, we’ll just assume that Faber College has jumped through some hoops and has the status.

Alice doesn’t realize it, yet, but in order to use this digital transcript she will need a new type of identity - - not the traditional identity that Faber College has built for her in its on-campus database, but a new and portable one that belongs to her, independent of all past and future relationships, that nobody can revoke or co-opt or correlate without her permission. This is a ***self-sovereign identity***, and it is the core feature for which Sovrin was named.

In normal contexts, managing a self - sovereign identity will require a tool such as a desktop or mobile application. It might be a standalone app, or it might leverage a third party service provider that Sovrin calls an ***agency***. The Sovrin Foundation publishes reference versions of such tools. Faber College will have studied these requirements and will recommend a ***Sovrin app*** to Alice if she doesn't already have one; this app will install as part of the workflow from the *Get Transcript* button.

When Alice clicks *Get Transcript*, she will download a file that holds a Sovrin ***link invitation***. This file, having a.sovrin extension and associated with her Sovrin app, will allow her to establish a secure channel of communication with another party in the Sovrin ecosystem–Faber College.

So when Alice clicks *Get Transcript*, she will normally end up installing an app (if needed), launching it, and then being asked by the app whether she wants to accept an invitation to connect with Faber.

For this guide, however, we'll be using a command - line interface instead of an app, so we can see what happens behind the scenes. We will pretend to be a particularly curious and technically adventurous Alice…

Install Sovrin

Coming soon: We are in the process of creating new vm images for Ubuntu and Windows for validator nodes and also client side docker image to make it easy for you to play with Sovrin.

Manually setup validator nodes (optional): If you prefer to install Sovrin-Node manually without using virtual machine images, please see the installation instructions for Linux or Windows.

```
$ pip install -U --no-cache-dir sovrin-client
```

If you get any error, check out the info about prerequisites; there are a few dominoes you might have to line up.

The install puts some python modules on your system. Most importantly, it gives you a command - line interface(CLI) to Sovrin. We are going to use that CLI to explore what Sovrin can do. (Sovrin also has a programmatic API, but it is not yet fully formalized, and this version of the guide doesn’t document it. See the Sovrin roadmap.)

Run the Sovrin CLI

Type this command:

```
$ sovrin
```

You should see an interactive prompt, like this:

```
Sovrin - CLI version 1.17(c) 2016 Evernym, Inc.
Type 'help' for more information.
sovrin>
```

We’re going to be playing the role of multiple ***identity owners*** (a person like Alice, an organization like Faber College, or an IoT - style thing; these are often called “principals” in security circles) before the guide is done.To do this we’ll use multiple shells. To make it easy to keep track of which identity owner we’re representing in a given window, let’s change the prompt:

```
sovrin> prompt ALICE
ALICE>
```

The status command gives general information about the state of the CLI. Alice tries it:

```
ALICE> status
Not connected to Sovrin network. Please connect first.


Usage:
    connect(test | live)
```

Alice might also try the help command to explore what’s available.

Evaluate the Invitation

To make this guide more convenient, the sovrin CLI package installs a sample Faber College invitation to CLI ROOT/scripts/sample/faber-invitation.sovrin. We’re going to use this file as if we had downloaded it from Faber. (Remember, in normal usage, Alice’s Sovrin app would be doing a lot of these steps automatically.)

```
ALICE> show sample/faber-invitation.sovrin
{
  "link-invitation": {
    "name": "Faber College",
    "identifier": "FuN98eH2eZybECWkofW6A9BKJxxnTatBCopfUiNxo6ZB",
    "nonce": "b1134a647eb818069c089e7694f63e6d"
  },
  "sig":
"4QKqkwv9gXmc3Sw7YFkGm2vdF6ViZz9FKZcNJGh6pjnjgBXRqZ17Sk8bUDSb6hsXHc

}



Try Next:
    load sample/faber-invitation.sovrin
```

Alice sees a bunch of data that looks interesting but mysterious. She wants Sovrin to tell her if the link invitation file is well formed and has something useful in it, so she uses the load command:

```
ALICE> load sample/faber-invitation.sovrin
New keyring Default created
Active keyring set to "Default"
1 link invitation found for Faber College.
Creating Link for Faber College.

Try Next:
    show link "Faber College"
    accept invitation from "Faber College"
```

This causes Sovrin to parse and validate the file. Alice would now like to know what’s entailed in accepting the invitation. She types:

```
ALICE> show link Faber
```

Unlike the show command for files, this one asks Sovrin to show a link. More details are exposed:

```
Expanding Faber to "Faber College"
Link (not yet accepted)
    Name: Faber College
    Identifier: not yet assigned
    Trust anchor: Faber College (not yet written to Sovrin)
    Verification key: <empty>
    Signing key: <hidden>
    Target: FuN98eH2eZybECWkofW6A9BKJxxnTatBCopfUiNxo6ZB
    Target Verification key: <unknown, waiting for sync>
    Target endpoint: <unknown, waiting for sync>
    Invitation nonce: b1134a647eb818069c089e7694f63e6d
    Invitation status: not verified, target verkey unknown
    Last synced: <this link has not yet been synchronized>

Try Next:
    sync "Faber College"
    accept invitation from "Faber College"
```

You'll see the link contains several pieces of information. Let's examine them one at a time.

```
Name: Faber College
```

This is a friendly name for the link that Alice has been invited to accept. The name is stored locally and not shared. Alice can always rename a link; its initial value is just provided by Faber for convenience.

```
Identifier: not yet assigned
```

Identifier is a unique value that gets generated when user tries to accept the invitation, and that identifier will be sent to Faber College, and used by Faber College to reference Alice in secure interactions. Each link invitation on Sovrin establishes a pairwise relationship when accepted, and each pairwise relationship uses different identifiers. Alice won’t use this identifier with other relationships. By having independent pairwise relationships, Alice reduces the ability for others to correlate her activities across multiple interactions.

```
Trust anchor: Faber College(not yet written to Sovrin)
```

This gives Alice a friendly name for the entity that is bootstrapping the new pairwise relationship onto the Sovrin ecosystem. Trust anchors provide a way for identifiers to be added to Sovrin. They are generally organizations but can be persons as well. Faber College is a trust anchor, and if its invitation is accepted, will write Alice’s identifier to Sovrin.

It is important to understand that this identifier for Alice is not, in and of itself, the same thing as Alice’s self-sovereign identity. Rather, Alice’s identity will– for her–be the sum total of all the pairwise relationships she has, and all the attributes knowable about those manifestations of her identity, across the full network. If Alice accepts this invitation, she will have a self-sovereign identity by virtue of the fact that she is accessible on the network through at least one relationship, and Faber College will be creating the first relationship participating in Alice’s identity–but Alice’s identity will not be captive to Faber College in any way.

```
Verification key: <empty>
```

Alice’s ***verification key*** allows Sovrin and Faber College to trust, in cryptographic operations, that interactions with Alice are authentically bound to her as sender or receiver. It is an asymmetric public key, in cryptographic terms, and the Sovrin CLI generated this value randomly when it loaded the invitation.

Verification key is a 32 byte Ed25519 verification key. Ed25519 is a particular elliptic curve, and is the default signature scheme for Sovrin.

The Verification key has a subtle relationship with the Identifier value a couple lines above it in the CLI output. Identifiers in Sovrin are called ***DIDs*** (***distributed identifiers***). These are opaque, unique sequences of bits, like UUIDs or GUIDs.

There are three options possible for ***verification key*** associated with a DID: - Empty. There are no verkey associated with a DID, and DID is NCID (non-cryptographic identifier). In this case, the creator of the Sovrin identity record (called a trust anchor) controls the identifier, and no independent proof-of-existence is possible until either Abbreviated or Full verkey is created. - Abbreviated. In this case, there is a verkey starting with a tilde '~' followed by 22 or 23 characters. The tilde indicates that the DID itself represents the first 16 bytes of the verkey, and the string following the tilde represents the second 16 bytes of the verkey, both using base58Check encoding. - Full. In this case, there is a full 44 character verkey, representing a base58Check encoding of all 32 bytes of a Ed25519 verification key

In the current guide Abbreviated key will be created and used by Alice (you will notice `~` prefix for verification key in the guide).

The key that Alice uses to interact with Faber can change if she revokes or rotates it, so accepting this invitation and activating this link doesn't lock Alice in to permanent use of this key. Key management events are discoverable in the Sovrin ledger by parties such as Faber College; we'll touch on that later in the guide.

```
Signing key: < hidden >
```

A different ***signing key*** is used by Alice to interact with each party on Sovrin (Faber College in this case). A signing key is an asymmetric private key, in cryptographic terms, and the Sovrin CLI also generated this value when it loaded the invitation. Alice will sign her messages to Faber College with this key, but she will never transmit the signing key anywhere. Because she signs with this key, Faber College can use the associated verification key and know it's really dealing with Alice. It's important that this signing key is kept secret, as someone with this key can impersonate Alice. If this key is ever compromised, Alice can replace it with a new one using several methods not covered here.

```
Target: FuN98eH2eZybECWkofW6A9BKJxxnTatBCopfUiNxo6ZB
```

Target is the unique identifier Alice uses to reference Faber College. Faber College provided this value in the invitation. Alice can use it to look up Faber College’s verification key in the Sovrin Ledger to ensure interactions with Faber College are authentic.

```
Target Verification key: < unknown, waiting for sync >
```

Communication from the target can’t be confirmed unless we know its verification key. We know the target is a CID(that’s what the Target line just above told us)–but since key revocations and rotations might happen at any time, we cannot assume that a CID has not updated its verification key. To know the true verification key of an identifier, we have to query Sovrin. Different use cases require different levels of assurance as to how recently we’ve queried Sovrin for any key replacements. In this case we might be comfortable if we know that the key was synchronized in the last hour. But we can see that we’ve never synchronized this link, so we don’t know what the verification key is at all. Until Alice connects to Sovrin, she won’t be able to trust communication from Faber College.

```
Target endpoint: < unknown, waiting for sync >
```

Targets can have endpoints –locations(IRIs / URIs / URLs) on the network where others can contact them. These endpoints can be static, or they can be ephemeral pseudonymous endpoints facilitated by a third party agency. To keep things simple, we'll just use static endpoints for now.

```
Invitation nonce: b1134a647eb818069c089e7694f63e6d
```

This is just a big random number that Faber College generated to track the unique invitation. When an invitation is accepted, the invitee digitally signs the nonce such that the inviter can match the acceptance with a prior invitation.

```
Invitation status: not verified, target verification key unknown
```

Invitations are signed by the target. We have a signature, but we don't yet know Faber College's verification key, so the signature can't be proved authentic.We might have an invitation from someone masquerading as Faber College. We'll resolve that uncertainty when we sync. `Last synced: < this link has not yet been synchronized >` A link stores when it was last synchronized with the Sovrin network, so we can tell how stale some of the information might be. Ultimately, values will be proved current when a transaction is committed to the ledger, so staleness isn't dangerous–but it makes Sovrin more efficient when identity owners work with up-to-date data.

Accept the Invitation

Alice attempts to accept the invitation from Faber College.

```
ALICE> accept invitation from Faber
Expanding Faber to "Faber College"
Invitation not yet verified.
Link not yet synchronized.
Invitation acceptance aborted.
Cannot sync because not connected. Please connect first.

Usage:
    connect <test|live>
```

In order to accept an invitation, its origin must be proved. Just because an invitation says the sender is “Faber College” doesn’t make it so; the ease of forging email headers is a reminder of why we can’t just trust what a sender says. Syncing the link with Sovrin will allow us to prove the association between Faber College’s identity and public key, but the CLI must be connected to the Sovrin network to sync– and we haven’t connected yet.

There are two Sovrin networks we might connect to. One is a test network, and the other is live(production). We’ll use the test network for the demo.

```
ALICE> connect test
Connected to test.
```

Alice tries again to accept the invitation from Faber College. This time she succeeds.

```
ALICE> accept invitation from Faber
Expanding Faber to "Faber College"
Invitation not yet verified.
Link not yet synchronized.
Attempting to sync...

Synchronizing...
    Link Faber College synced

Pinging target endpoint: ('54.70.102.199', 5555)
    Ping sent.
    Pong received.

Signature accepted.

Response from Faber College (835.18 ms):
    Trust established.
    Identifier created in Sovrin.
    Available Claim(s): Transcript

Synchronizing...
    Confirmed identifier written to Sovrin.

Try Next:
    show claim "Transcript"
    request claim "Transcript"
```

Accepting an invitation takes the nonce that Faber College provided, and signs it with the Alice’s signing key. It then securely transmits the signed data along with the identifier and verification key to Faber College’s endpoint, which is discovered when the link is synchronized. Faber College matches the provided nonce to the record of the nonce it sent to Alice, verifies the signature, then records Alice’s new pairwise identifier in the Sovrin ledger.

Once the link is accepted and synchronized, Alice inspects it again.

```
ALICE> show link Faber
Expanding Faber to "Faber College"
Link
    Name: Faber College
    Identifier: LZ46KqKd1VrNFjXuVFUSY9
    Trust anchor: Faber College (confirmed)
    Verification key: ~CoEeFmQtaCRMrTy5SCfLLx
    Signing key: <hidden>
    Target: FuN98eH2eZybECWkofW6A9BKJxxnTatBCopfUiNxo6ZB
    Target Verification key: <same as target>
    Target endpoint: 54.70.102.199:5555
    Invitation nonce: b1134a647eb818069c089e7694f63e6d
    Invitation status: Accepted
    Available Claim(s): Transcript
    Last synced: 14 seconds ago

Try Next:
    show claim "Transcript"
    request claim "Transcript"
```

Notice now that the Last synced line is updated.

Alice can see now that the target verification key and target endpoint, as well as identifier and verification key are updated, which allows her to communicate with Faber College. She can also see that the identity of the trust anchor was confirmed (from the Sovrin network), and that her invitation has been accepted.

Test Secure Interaction

At this point Alice is connected to Faber College, and can interact in a secure way. The Sovrin CLI supports a ping command to test secure pairwise interactions. (This command is partly implemented today, and partly still a stub.)

```
ALICE> ping Faber
Expanding Faber to "Faber College"

Pinging target endpoint: ('54.70.102.199', 5555)
    Ping sent.
    Pong received.
```

Alice receives a successful response from Faber College. Here’s what happens behind the scenes:

1. The ping she sends contains a random challenge.
2. The ping also includes Alice’s pairwise identifier and a signature.

3. Faber College verifies Alice's signature.

4. Faber College digitally signs that challenge and sends it back.

5. Alice verifies that the response contained the same random challenge she sent.

6. Alice uses the verification key in the Faber College Link to verify the Faber College digital signature.

She can trust the response from Faber College

because (1) she connects to the current endpoint, (2) no replay - attack is possible, due to her random challenge, (3) she knows the verification key used to verify Faber College's digital signature is the correct one because she just confirmed it on Sovrin.

Inspect the Claim

Notice that when Alice last showed the Faber link, there was a new line: `Available Claim(s): Transcript`. A ***claim*** is a piece of information about an identity - -a name, an age, a credit score… It is information claimed to be true. In this case, the claim is named "Transcript."

Claims are offered by an ***issuer***. An issuer may be any identity owner known to Sovrin, and any issuer may issue a claim about any identity owner it can identify. The usefulness and reliability of a claim are tied to the reputation of the issuer, with respect to the claim at hand. For Alice to self-issue a claim that she likes chocolate ice cream may be perfectly reasonable, but for her to self-issue a claim that she graduated from Faber College should not impress anyone. The value of this transcript is that it is provably issued by Faber College. Alice wants to use that claim. She asks for more information:

```
ALICE> show claim Transcript
Found claim Transcript in link Faber College
Status: available (not yet issued)
Name: Transcript
Version: 1.2
Attributes:
    student_name
    ssn
    degree
    year
    status

Try Next:
    request claim "Transcript"
```

Alice sees the attributes the transcript contains. These attributes are known because a schema for Transcript has been written to the ledger (see Appendix). However, the “not yet issued” note means that the transcript has not been delivered to Alice in a usable form. To get the transcript, Alice needs to request it.

```
ALICE> request claim Transcript
Found claim Transcript in link Faber College
Getting Keys for the Schema from Sovrin
Requesting claim Transcript from Faber College...

Signature accepted.

Response from Faber College (65.17 ms):
    Received claim "Transcript".
```

Now the transcript has been issued; Alice has it in her possession, in much the same way that she would hold a physical transcript that had been mailed to her. When she inspects it again, she sees more details:

```
ALICE> show claim Transcript
Found claim Transcript in link Faber College
Status: 2016-10-18 14:44:53.368163
Name: Transcript
Version: 1.2
Attributes:
    ssn: 123-45-6789
    student_name: Alice Garcia
    year: 2015
    status: graduated
    degree: Bachelor of Science, Marketing
```

Apply for a Job

Alice would like to work for Acme Corp. Normally she would browse to acmecorp.com, where she would click on a hyperlink to apply for a job. Her browser would download a link invitation which her Sovrin app would open; this would trigger a prompt to Alice, asking her to accept the link with Acme Corp. Because we’re using a CLI, the interface is different, but the steps are the same. We do approximately the same things that we did when Alice was accepting Faber College’s link invitation:

```
ALICE> show sample/acme-job-application.sovrin
{
  "link-invitation": {
    "name": "Acme Corp",
    "identifier": "7YD5NKn3P4wVJLesAmA1rr7sLPqW9mR1nhFdKD518k21",
    "nonce": "57fbf9dc8c8e6acde33de98c6d747b28c",
    "endpoint": "54.70.102.199:6666"
  },
  "proof-requests": [{
    "name": "Job-Application",
    "version": "0.2",
    "attributes": {
        "first_name": "string",
        "last_name": "string",
        "phone_number": "string",
        "degree": "string",
        "status": "string",
        "ssn": "string"
    },
    "verifiableAttributes": ["degree", "status", "ssn"]
  }],
  "sig":
"oeGeMdt5HRjRsbaXybGpRmkkijhHrGT82syxofEJQbkjTCLW63tM3jMn1boaf62vCS

}

Try Next:
    load sample/acme-job-application.sovrin
```

Notice that this link invitation contains a ***proof request***. ACME Corp is requesting that Alice provide a Job Application. The Job Application is a rich document type that has a schema defined on the Sovrin ledger; its particulars are outside the scope of this guide, but it will require a name, SSN, and degree, so it overlaps with the transcript we've already looked at. This becomes important below.

Notice that the invitation also identifies an endpoint. This is different from our previous case, where an identity owner's endpoint was discovered through lookup on the Sovrin ledger. Here, Acme Corp.has decided to short - circuit Sovrin and just directly publish its job application acceptor endpoint with each request. Sovrin supports this. Alice quickly works through the sequence of commands that establishes a new pairwise connection with Acme: ``` ALICE> load sample/acme-job-application.sovrin 1 link invitation found for Acme Corp. Creating Link for Acme Corp.

Try Next: show link "Acme Corp" accept invitation from "Acme Corp"

ALICE> show link Acme Expanding Acme to "Acme Corp" Link (not yet accepted) Name: Acme Corp Identifier: not yet assigned Trust anchor: Acme Corp (not yet written to Sovrin) Verification key: Signing key: Target: 7YD5NKn3P4wVJLesAmA1rr7sLPqW9mR1nhFdKD518k21 Target Verification key: <unknown, waiting for sync> Target endpoint: 54.70.102.199:6666 Invitation nonce: 57fbf9dc8c8e6acde33de98c6d747b28c Invitation status: not verified, target verkey unknown Proof Request(s): Job-Application Last synced:

Try Next: sync “Acme Corp” accept invitation from “Acme Corp”

ALICE> accept invitation from Acme

Expanding Acme to “Acme Corp” Invitation not yet verified. Link not yet synchronized. Attempting to sync…

Synchronizing… Link Acme Corp synced

Pinging target endpoint: (‘54.70.102.199’, 6666) Ping sent. Pong received.

Signature accepted.

Response from Acme Corp (841.46 ms): Trust established. Identifier created in Sovrin.

Synchronizing… Confirmed identifier written to Sovrin.

Try Next: show proof request “Job-Application” send proof “Job-Application” to “Acme Corp” ```

Notice what the proof request looks like now. Although the application is not submitted, it has various claims filled in:

```
ALICE> show proof request Job-Application
Found proof request "Job-Application" in link "Acme Corp"
Status: Requested
Name: Job-Application
Version: 0.2
Attributes:
    first_name: string
    last_name: string
    phone_number: string
    degree: Bachelor of Science, Marketing
    status: graduated
    ssn: 123-45-6789
Verifiable Attributes:
    degree
    status
    ssn


The Proof is constructed from the following claims:

    Claim (Transcript v1.2 from Faber College)
        ssn: 123-45-6789 (verifiable)
        status: graduated (verifiable)
        year: 2015 (verifiable)
        student_name: Alice Garcia (verifiable)
        degree: Bachelor of Science, Marketing (verifiable)

Try Next:
    set <attr-name> to <attr-value>
    send proof "Job-Application" to "Acme Corp"
```

Alice only has one claim that meets proof requirements for this Job Application, so it is associated automatically with the request; this is how some of her attributes are pre-populated.

The pre - population doesn't create data leakage, though; the request is still pending. Alice can edit what she is willing to supply for each requested attribute.

Notice that some attributes are verifiable, and some are not. The proof request schema says that ssn and degree(and others) in the transcript must be formally asserted by an issuer other than Alice. Notice also that the first occurrence of first_name and last_name, plus the only occurrence of phone_number, are empty, and are not required to be verifiable. By not tagging these claims with a verifiable status, Acme's claim request is saying it will accept Alice's own claim about her names and phone numbers. (This might be done to allow Alice to provide a first name that's a nickname, for example.) Alice therefore adds the extra attributes now:

```
ALICE> set first_name to Alice
ALICE> set last_name to Garcia
ALICE> set phone_number to 123-45-6789
```

Alice checks to see what the proof request looks like now.

```
ALICE> show proof request Job-Application
Found proof request "Job-Application" in link "Acme Corp"
Status: Requested
Name: Job-Application
Version: 0.2
Attributes:
    first_name: Alice
    last_name: Garcia
    phone_number: 123-45-6789
    degree: Bachelor of Science, Marketing
    status: graduated
    ssn: 123-45-6789
Verifiable Attributes:
    degree
    status
    ssn


The Proof is constructed from the following claims:

    Claim (Transcript v1.2 from Faber College)
        ssn: 123-45-6789 (verifiable)
        status: graduated (verifiable)
        year: 2015 (verifiable)
        student_name: Alice Garcia (verifiable)
        degree: Bachelor of Science, Marketing (verifiable)

Try Next:
    set <attr-name> to <attr-value>
    send proof "Job-Application" to "Acme Corp"
```

She decides to submit.

```
ALICE> send proof Job-Application to Acme
Signature accepted.

Response from Acme Corp (451.9 ms):
    Your claim Job-Application 0.2 has been received and is
verified

Response from Acme Corp (311.2 ms):
    Available Claim(s): Job-Certificate
```

It will be interesting to see whether Acme accepts this application with the informal first_name not matching the one on her transcript. If Acme is concerned about this discrepancy, it could reach out to Alice and ask about it, using the secure channel that's now established. Alice could send a photo showing her college ID that lists her name as "Alice (Sally) Gonzales".

Here, we'll assume the application is accepted, and Alice ends up getting the job. When Alice inspects her link with Acme a week later, she sees that a new claim is available:

```
ALICE> show link Acme
Expanding Acme to "Acme Corp"
Link
    Name: Acme Corp
    Identifier: cid-
2:EigjZrDwLREcX2T7P5zNZwZ6R7wv3GcdaKxAeN7cbqYX
    Trust anchor: Acme Corp (confirmed)
    Verification key: <same as local identifier>
    Signing key: <hidden>
    Target: 7YD5NKn3P4wVJLesAmA1rr7sLPqW9mR1nhFdKD518k21
    Target Verification key: <same as target>
    Target endpoint: 54.70.102.199:6666
    Invitation nonce: 57fbf9dc8c8e6acde33de98c6d747b28c
    Invitation status: Accepted
    Proof Request(s): Job-Application
    Available Claim(s): Job-Certificate
    Last synced: a minute ago

Try Next:
    show claim "Job-Certificate"
    request claim "Job-Certificate"
    show proof request "Job-Application"
    send proof "Job-Application" to "Acme Corp"
```

Apply for a Loan

Now that Alice has a job, she'd like to apply for a loan. That will require proof of employment. She can get this from the Job-Certificate claim offered by Acme. Alice goes through a familiar sequence of interactions. First she inspects the claim:

```
ALICE> show claim Job-Certificate
Found claim Job-Certificate in link Acme Corp.
Status: available(not yet issued)
Name: Job-Certificate
Version: 0.2
Attributes:
    first_name
    last_name
    employement_status
    experience
    salary_bracket

Try Next:
    request claim "Job-Certificate"
```

Next, she requests it:

```
ALICE> request claim Job-Certificate
Found claim Job-Certificate in link Acme Corp
Getting Keys for the Schema from Sovrin
Requesting claim Job-Certificate from Acme Corp...

Signature accepted.

Response from Acme Corp (55.17 ms):
    Received claim "Job-Certificate".
```

The Job-Certificate has been issued, and she now has it in her possession.

```
ALICE> show claim Job-Certificate
Found claim Job-Certificate in link Acme Corp.
Status: issued 2016-08-15
Name: Job-Certificate
Version: 0.1
Attributes:
    first_name: Alice
    last_name: Garcia
    employement_status: Permanent
    experience: 3 years
    salary_bracket: between $50,000 to $100,000
```

She can use it when she applies for her loan, in much the same way that she used her transcript when applying for a job.

There is a disadvantage in this approach to data sharing, though–it may disclose more data than what is strictly necessary. If all Alice needs to do is provide proof of employment, this can be done with an anonymous credential instead. Anonymous credentials may prove certain predicates without disclosing actual values (e.g., Alice is employed full-time, with a salary greater than X–but how much her salary is, and what her hire date is, remain hidden).

Support for anonymous credentials is at a late alpha stage on Sovrin right now. We’ll circle back and update this guide when we reach beta.

Alice now loads Thrift Bank’s loan application link: ``` ALICE@test> load sample/thrift-loan-application.sovrin 1 link invitation found for Thrift Bank. Creating Link for Thrift Bank.

Try Next: show link “Thrift Bank” accept invitation from “Thrift Bank” ```

Alice accepts the loan application link:

```
ALICE@test> accept invitation from thrift
Expanding thrift to "Thrift Bank"
Invitation not yet verified.
Link not yet synchronized.
Attempting to sync...

Synchronizing...
    Link Thrift Bank synced

Pinging target endpoint: ('54.70.102.199', 7777)
    Ping sent.
    Pong received.

Signature accepted.

Response from Thrift Bank (842.49 ms):
    Trust established.
    Identifier created in Sovrin.

Synchronizing...
    Confirmed identifier written to Sovrin.

Try Next:
    show proof request "Loan-Application-Basic"
    send proof "Loan-Application-Basic" to "Thrift Bank"
    show proof request "Loan-Application-KYC"
    send proof "Loan-Application-KYC" to "Thrift Bank"
```

Alice checks to see what the proof request “Loan-Application-Basic” looks like: ``` ALICE@test> show proof request Loan-Application-

Basic Found proof request “Loan-Application-Basic” in link “Thrift Bank” Status: Requested Name: Loan-Application-Basic Version: 0.1 Attributes: salary_bracket: between $50,000 to $100,000 employee_status: Permanent Verifiable Attributes: salary_bracket employee_status

The Proof is constructed from the following claims:

```
Claim (Job-Certificate v0.2 from Acme Corp)
    last_name: Garcia (verifiable)
    salary_bracket: between $50,000 to $100,000 (verifiable)
    employee_status: Permanent (verifiable)
    experience: 3 years (verifiable)
    first_name: Alice (verifiable)
```

Try Next: set to send proof Loan-Application-Basic to Thrift Bank ```

Alice sends just the “Loan-Application-Basic” proof to the bank. This allows her to minimize the PII that she has to share when all she’s trying to do right now is prove basic eligibility.

```
ALICE@test> send proof Loan-Application-Basic to Thrift Bank

Signature accepted.

Response from Thrift Bank (479.17 ms):
    Your Proof Loan-Application-Basic 0.1 has been received and
verified

    Loan eligibility criteria satisfied, please send another
proof 'Loan-Application-KYC'
```

Alice now checks the second proof request where she needs to share her personal information with bank.

```
ALICE@test> show proof request Loan-Application-KYC
Found proof request "Loan-Application-KYC" in link "Thrift Bank"
Status: Requested
Name: Loan-Application-KYC
Version: 0.1
Attributes:
    first_name: Alice
    last_name: Garcia
    ssn: 123-45-6789
Verifiable Attributes:
    first_name
    last_name
    ssn


The Proof is constructed from the following claims:
    Claim (Transcript v1.2 from Faber College)
        degree: Bachelor of Science, Marketing (verifiable)
        student_name: Alice Garcia (verifiable)
        year: 2015 (verifiable)
        ssn: 123-45-6789 (verifiable)
        status: graduated (verifiable)

    Claim (Job-Certificate v0.2 from Acme Corp)
        last_name: Garcia (verifiable)
        salary_bracket: between $50,000 to $100,000 (verifiable)
        employee_status: Permanent (verifiable)
        experience: 3 years (verifiable)
        first_name: Alice (verifiable)

Try Next:
    set <attr-name> to <attr-value>
    send proof Loan-Application-KYC to Thrift Bank
```

Alice now sends “Loan-Application-KYC” proof to the bank: ``` ALICE@test> send proof Loan-Application-KYC to Thrift Bank

Signature accepted.

Response from Thrift Bank (69.9 ms): Your Proof Loan-Application-KYC 0.1 has been received and verified ```

Appendix

Faber College Configures Transcripts

The following operations show how Transcripts are defined on the ledger, such that they can later be issued with reference to a known schema. ``` faber> use Schema-Keyring

faber> new schema name = “Transcript” version = “1.2” attributes = { “student_name”: “string”, “ssn”: “int”, “degree”: “string”, “year”: “int”, “status”: “string” }

Schema Transcript v1 .2 added to Schema-Keyring

faber> add keys of type CL for schema Transcript version 1.2 Keys added

faber> show pending 2 pending … … faber> submit pending Submitting 2 transactions… …. …. Submitted. ```

Acme Corp Defines a Job-Application

A similar process is followed by Acme Corp. to define a Job Application.

```
$ sovrin
```

```
sovrin> prompt
acme
acme> new schema
name = "Job-Application"
version = "0.1"
attributes = {
    "first_name": "string",
    "last_name": "string",
    "phone_number": "int",
    "proofs": [{
        "name": "Transcript",
        "version": 1.2,
        "attributes": {
            "first_name": "string",
            "last_name": "string",
            "ssn": "string",
            "degree": "string",
            "status": "string"
        }
    }]
}

Schema Job-Application v0 .1 added


acme> show pending 1 pending
...
acme> submit pending Submitting 2 transactions...
....
....
Submitted.
```