LearnThings.Online

Subscribe

Introduction to Hyperledger Sovereign Identity Blockchain Solutions: Indy, Aries & Ursa

About this course

This course is from edX, scroll down & click “Read More” for more informations.

To the surprise of absolutely no one, trust is broken on the Internet. Wherever you go online, the advice is the same–make sure you understand what’s behind each button before you click it.

In this course, we’ll dive into three Hyperledger open source projects–Indy, Aries and Ursa–looking at the tools, libraries, and reusable components they provide for creating and using independent digital identities rooted on blockchains or other distributed ledgers. We will explore the possibilities they offer for building applications on a solid digital foundation of trust and examine how these technologies can make the Internet safe. It’s quite a challenge!

The course is addressed to a wide-ranging audience, walking the line between business and technology.

What you’ll learn

- The problems with existing Internet identity/trust mechanisms today.
- How a distributed ledger, such as Hyperledger Indy, can be used for identity.
- How the underlying blockchain technology makes it possible.

- Understand the purpose, scope and relationship between Aries, Indy and Ursa.
- The possibilities enabled by this new technology.

Welcome!

Introduction and Learning Objectives

Introduction

Welcome to LFS172x – Introduction to Hyperledger Sovereign Identity Blockchain Solutions: Indy, Aries and Ursa!

To the surprise of absolutely no one, trust is broken on the Internet. Wherever you go online, the advice is the same—make sure you understand what's behind each button before you click it.

In this course, we'll dive into three Hyperledger open source projects—**Indy**, **Aries** and **Ursa**—looking at the tools, libraries, and reusable components they provide for creating and using independent digital identities rooted on blockchains or other distributed ledgers. We will explore the possibilities they offer for building applications on a solid digital foundation of trust and examine how these technologies can make the Internet safe. It's quite a challenge!

What You Will Learn

Hyperledger Indy, Aries and Ursa are blockchain-based frameworks that enable an important missing layer of the Internet—the trust layer. While still rooted in blockchain technology, the use of blockchain in these projects is quite different from the other Hyperledger projects, such as Fabric or Sawtooth. Where the other projects are general purpose blockchain systems (they can be used in many situations), Indy, Aries and Ursa are used for a specific purpose—and it's big! They are about identity on the Internet and being able to trust the data passed to you. They are about being able to prove to others who you are and you being certain who they are.

What's the big deal about identity? Well, when the Internet was first created, all of the computers that were connecting to one another were "trusted". The number of systems was small and the people running those systems knew each other so they didn't need mechanisms to know who was sending what data between the systems. As the number of systems on the Internet grew (and grew and grew…), that trust quickly diminished and the first of many mechanisms was added to systems to try to identify who was contacting whom. Unfortunately, the problem has never really been solved. The most common (and universally hated) system, that of user IDs and passwords, is fraught with problems. The resulting lack of certainty of who is on the other keyboard has led to the loss of billions from hacks, data breaches, identity theft, scams and more. Further, that same lack of certainty has made many types of business transactions on the Internet impossible—the risk of being fooled is just too high.

Indy, Aries and Ursa have been created to add a trust layer to the Internet using mechanisms that are easy to use, enable online trust and enhance privacy. It's a big goal that is of vital importance to everyone on the Internet. And, it's a goal that has recently become realizable with the advent of blockchain.

Terminology

We use the terms blockchain, ledger, decentralized ledger technology (DLT) and decentralized ledger (DL) interchangeably in this course. While there are precise meanings of those terms in other contexts, in the context

of the material covered in this course the differences are not meaningful.

For more definitions, take a look at our course Glossary section.

Chapter 1: Something Is Missing

Chapter Overview

Trust is broken on the Internet.

That statement is intended to be controversial and surprising. But it's not. Everyone knows that trust is not a part of using the Internet. We are trained to not believe what is said on the Internet, to not click links that we are unsure of, and if we want to do something really important, well, we just don't use the Internet for that. As the famous 1993 New Yorker cartoon goes—"*No one knows you are a dog on the Internet*." It was true then, and sadly, it's still true today. We need to fix that.


Laptop and dog
Trust Is Broken on the Internet

To ensure that the person you're talking to online is who they say they are, that what they are saying is true, and whether or not you want to trust them for some business transaction, you really have to do a thorough background investigation. Today,

- Would you buy a house based solely on the electronic data sent to you by the seller?
- Would a university admit a student based solely on the grades and background delivered electronically directly from the student?
- Would you help a Nigerian prince transfer 30M$ for 30% commission based solely on unsolicited online messages?

The answer is "no way!!," right?

In this chapter, we will explore the reasons why it's difficult to establish trust about the identity of an online contact and what they say. We'll look at how this trust was established in the past—and why that approach no longer works. In future chapters, we'll look at how the Hyperledger Indy, Aries and Ursa projects can change that situation by adding the missing trust layer to the Internet.

Learning Objectives

By the end of this chapter you should:

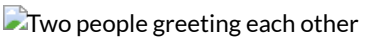
- Understand the basic mechanism (user ID and password) for proving your identity on the Internet and the problems it creates.
- Understand how Identity Providers (IdPs) can learn more about us than we want them to.
- Be familiar with other Internet identity challenges such as unwanted correlation, data breaches and central authority issues.
- Understand the paper credential model that we have used for thousands of years.

Interaction Creates Identifiers

When we interact in the real world, we often need to "prove" who we are. To do that, we present some evidence we have about ourselves. What we present varies based on the context of the relationship. When we meet someone socially, we introduce ourselves. When we want to open a bank account, we show documents (attributes) issued by others (driver's license, utility bill, government ID, etc.) to prove things about ourselves, such as our name, address and government ID number (e.g. SIN in Canada, SSN in the USA).

In turn, those with whom we interact create an identifier for us and check that identifier when we connect again. A person remembers our name and face, and "verifies" them on our subsequent meetings ("Hi Stephen! Good to

see you again!"). Our bank creates a card with an ID on it for us to use each time we go to a bank branch.


Two people greeting each other

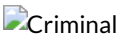
Interaction Creates Identifiers

The same pattern is used online, but there are a variety of problems that occur, some obvious, some a little more subtle. Let's go through them.

Identifiers: User IDs and Passwords

The basic mechanism for knowing who you are on the Internet is the user ID and password combination. You register at a site and get a user ID and set a secret password that only you know (right?), and each time you return you use them to access your account. We all know the problems with user IDs and passwords—we deal with them every day:

- We have too many to track.
- Because we have so many, we often use "easy to remember" passwords that are also easy for others to guess.
- Password recovery mechanisms (question/responses, support desks) provide avenues of attack by hackers. For example, if they know our ID and our recovery answers, they can reset our password to something only they know. This video from CNN shows an example of collecting data about a person and then using it against them. It's amazing—in a bad way.
- Data breaches occur that result in our IDs and passwords being exposed and used—either by hackers or anyone that buys them.
- We often use the same password on many sites, and if that password gets exposed on one site, our account on others sites is also exposed.


Criminal

Today's User ID and Password Model Has Problems!

A common approach to solving the "too many passwords" problem is the use of Identity Providers (IdPs) such as Facebook and Google. Smaller sites use an IdP for authentication (user ID and password verification) and to get basic identity attributes such as name and email address. A problem with that approach is that each time an IdP is used, the IdP learns things about us—our habits, our interests, what sites we use and so on. Our privacy is lost.

We also need to note that while users of websites have IDs for the site, the reverse is not the case—we are not provided an ID and password for the site that we can verify each time we connect. This has enabled "phishing" techniques to become common in recent years, where users are tricked to click a link that takes them to a website that appears to be real, has a similar name, but is actually fake (for example, goog1e.com). This fools us into revealing our user ID and password and sometimes even into revealing our two-factor authentication code for the real site.

Identity Attributes

On almost every site, we also share other identifiers to use a service—our email and name at minimum, and depending on the nature of the service, additional information (address, credit card info, etc.). With that, we can do the only common business transactions on the Internet—buy things. Buying things has a low enough risk because sellers can easily trace to whom the purchased item is delivered, which deters widespread abuse.

Conducting higher trust online transactions—such as opening a bank account—is much more difficult. We have to provide the same information about ourselves as we would in person. In theory, we should be able to just type those attributes in since they are private and (in theory) only we know them. However, much of that type of information is relatively widely known, either because it is routinely published (e.g. name and address), or because of the many data breaches that have occurred (e.g. government ID number). Even non-identifiers that are used for verification, so-called "shared secrets" such as the value of "Line 150 from your 2018 Tax Return," can be fraudulently acquired and used for targeting specific individuals.

An alternative is to do an online version of the in-person verification—scan and send the source documents. However, scans are easy to forge and as such are not trusted. We should also add that paper documents used in person are increasingly easy to forge as well. That issue is actually putting at risk the "real world" identity-proofing that we mentioned earlier. Verifiers, such as bank employees and lawyers, have to become experts at detecting the authenticity of documents used for identity proofing.

Additional Problems

We've talked about a few problems with the current Internet identity approaches already—user IDs and passwords are hackable, and supposedly private information is too well-known to be trusted. Let's discuss a few more problems with the current approaches.

Unwanted Correlation

The use of common identifiers on so many different sites creates what is known as a correlation problem. Correlation in this context means associating without consent information about a single identity across multiple systems. The proliferation of this kind of correlation on the Internet, driven primarily by advertising, has resulted in a massive loss of privacy for Internet users (basically, everyone). An excellent/horrible example of this was exposed by a data breach at the relatively unknown Florida company, Exactis. The breach was massive, covering almost every American and American company (340M records total). However, the content was equally shocking: more than 400 data elements per record collected from a number of sites correlating details about each person—their name, age, race, religion, size of family, etc. You can be sure that no one ever agreed to allow Exactis to collect that information. They correlated the data across many "partner" sites to collect a picture of each person that they in turn sold to anyone willing to pay.


Common Identifiers Across Multiple Sites Causes Correlation
Common Identifiers Across Multiple Sites Causes Correlation!

Correlation is made possible because of the common identifiers we use online daily. Our email address is the single largest factor since we share it on almost every site, but there are others. Each time we use the same account name on a different site we create the possibility of correlation. When we give other identifiers about ourselves (e.g. phone number, address, government IDs, etc.), firms can correlate that data across sites. Tracking cookies placed by websites and ads enable the linking of IDs across websites. Some of the new General Data Protection Regulations (GDPR) in Europe are designed to prevent these practices for companies that do business in Europe, but not so in other places. Even then, it is a legal, not technical solution, and so remains susceptible to bad actors. In fact, even with GDPR in place, the data collection continues, as this article from the Linux Journal shows in quite shocking detail, "*If Your Privacy Is in the Hands of Others Alone, You Don't Have Any*."

The Identity Providers (IdPs) model is also a correlation point—although in this case, one given with consent. The IdP approach trades convenience (fewer user IDs and passwords) for correlation. Since the IdP is used for each login, the IdP can track your use of other sites and thus correlate your online activities, increasing their knowledge of you.

Data Breaches

Identity-related data is currently of particularly high value and so large data repositories of identity data are favourite targets for hackers. This includes not only user ID and password data to enable unauthorized access to accounts, but also all of the other information we use to "prove" our identity online such as name, email address, government ID and so on. As noted, the availability of this supposedly private data makes the online use of such data impossible for high-value interactions because of the risk that the person typing the data is not its owner.

Centralized Identifiers

The vast majority of identifiers we use today are centralized—the identifiers are provided to us and maintained by a centralized entity. That might be the government in the case of our tax ID and driver's licence, or a company for an ID we use to log into a website. A major problem with this approach is that the central authority can choose to take away that identifier at any time. This is of particular concern for those in a minority situation— a critic of a central entity, be it a government suppressing its people, or a private company.

A second concern with central authorities controlling identifiers is that if they are compromised in some way, those identifiers can be used in malicious ways. For example, a hack of a Dutch Certificate Authority (manager of Secure Socket Layer Encryption Certificates) allowed supposedly secure encrypted data going across the Internet to be intercepted and accessed by hackers. Further, since the identifiers are held in a central place, if that repository is compromised, it impacts many people.

Paper Credentials

Let's talk about what have we done in the real world for identity. As far back as 450BC, we have used paper credentials to enable trusted identity. Legend has it that King Artixerxes of the Persian Empire signed and gave Nehemiah a paper "safe transit" authorization to travel across Persia. People have been using such documents ever since. However, starting around 1970, with the widespread use of corporate/personal printers and copiers, forging these paper documents became easy. And trying to use paper credentials on the Internet is even worse because they are even easier to forge—all you need is Photoshop—you don't even need a printer. So while paper credentials may not help us in the digital age, let's look at the attributes of paper credentials and what made them useful for so many years.

A **credential** is an attestation of qualification, competence, or authority issued to an entity (e.g. an individual or organization) by a third party with a relevant or de facto authority or assumed competence to do so. Examples of credentials issued to people include a driver's license, a passport, an academic degree, and so on. Credentials are also issued to companies, such as business registrations, building permits, and even health inspection certifications.


Examples of Paper Credentials
By Peter Stokyo

A typical paper credential, say a driver's license, is issued by a government authority (an issuer) after you prove to them who you are (usually in person using your passport or birth certificate) and that you are qualified to drive. You then hold this credential (usually in your wallet) and can use it elsewhere whenever you want—for example, when renting a car, in a bank to open up an account or in a bar to prove that you are old enough to drink. When you do that you've proven (or presented) the credential. That's the **paper credential model**.


The Paper Credential Model
By Peter Stokyo

The paper credential model (ideally) proves:

- Who issued the credential.
- Who holds the credential.
- The claims have not been altered.

The caveat "ideally" is included because of the possibility of forgery in the use of paper credentials. As many university students know, it's pretty easy to get a fake driver's license that can be used when needed to "prove" those same things.

Summary

So where does that leave us? These are important points and we'll come back to these throughout the rest of the course.

- User IDs/passwords are the norm, but they are a pain to use, and as a result, are susceptible to attack. They are the best we have right now, but not a solid basis for trust. Further, IDs work only one way—users don't get an ID from a service they are using.


User ID and Passwords Are a Pain to User

User ID and Passwords Are a Pain to Use!

- Other personal information and identifiers we have that we could otherwise use to prove our identity are not trusted because it's impossible to tell if the data was actually issued to the person entering it. The many breaches of private identifiers make them impossible to completely trust (even in person), and verifying that information adds (sometimes significant) costs.
- Since the identity attributes we could use are not trusted (they are not things only we know), we often have to resort to in-person delivery of paper documents to prove things about ourselves, adding costs for all participants.
- Reviewers of the paper documents we present must become experts in the state of the art in forging and falsifying paper documents, a difficult role in which to be placed.
- The identifiers we use are correlated across sites, allowing inferences to be made about us, and exposing information we don't intend to be widely shared.
- This is annoying at the least, and can have catastrophic results in the worst case. Centralized repositories of identifiers and data about the people associated with those identifiers are targeted by hackers because the data has high value. This exacerbates the problem of not being able to trust "personal" data presented online.
- Centralized identifiers can be abused by those who control those identifiers. For example, they can be taken away from a subject without due process.

In the next chapter, we'll look at how Hyperledger Indy, Aries and Ursa enable the same credential model as with paper, but one that is useful in person and online using an approach called the verifiable credentials model.

Chapter 2: Adding a Layer of Trust to the Internet

Chapter Overview

How do we address the issue of trust on the Internet? In the last chapter, we talked about a number of challenges with the current state of identity on the Internet. In this chapter, we'll talk about solutions that are becoming available that will add that missing layer of trust. To do that, we will move from paper credentials discussed in the last chapter to verifiable credentials—credentials we can use online. We will also introduce the important concepts of self-sovereign identity (SSI) and trust over IP (ToIP). Lastly, we will explore the technology that enables verifiable credentials (for example, decentralized identifiers (DIDs) and agents), and how all of this together makes the Internet a much more trusted place.

Learning Objectives

By the end of this chapter you should:

- Understand how the verifiable credentials model works.
- Be familiar with the core concepts of self-sovereign identity and the aspects of Indy, Aries and Ursa architecture the enable SSI.
- Know what the term "trust over IP" means.
- Know what a DID is and does.
- Be familiar with the term zero-knowledge proof and selective disclosure.
- Understand why blockchain is important for self-sovereign identity.

Verifiable Credentials

As we talked about in Chapter 1, a credential is (formally) an attestation of qualification, competence, or authority issued to an entity by a third party with a relevant or de facto authority or assumed competence to do so. Examples of paper versions of credentials are a driver's license, passport and university degree. What if we could get credentials from the same sources as our paper credentials and use them online in a fully trusted manner both for the holder of the credential and the person or organization verifying the credential? Something we can trust way more than just a scan of the paper credential.

The Hyperledger projects, Indy, Aries and Ursa, which we will talk about in upcoming chapters, are tools and libraries that allow for the development of independent digital identities rooted on blockchains or other distributed ledgers. They help bring about the possibility of building applications with a solid digital foundation of trust by enabling the **verifiable credentials model**.

The verifiable credentials model is the same as the paper credentials model. That is:

- An authority decides you are eligible to receive a credential and issues you one.
- You hold your credential in your (digital) wallet.

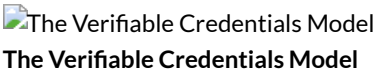
Note: We'll talk later in the course about how your wallet will be protected from being lost or stolen. It's a big deal!

- At some point, you are asked to prove the claims from the credential.
- You provide a verifiable presentation to the verifier, proving the same things as with a paper credential.
- You can prove that the issued credential has not been revoked.

As we'll see, verifiable credentials and presentations are not simple documents that anyone can create and use. They are cryptographically constructed so that a presentation proves the four key attributes of all credentials:

- Who issued the credential.
- The credential was issued to the entity presenting it.
- The claims were not tampered with.
- The credential has not been revoked.

Unlike a paper credential, those four attributes are evaluated not based on the judgment and expertise of the person looking at the credential, but rather online using cryptographic algorithms that are extremely difficult to forge. When a verifier receives a presentation from a holder, they use information from a blockchain (shown as the **verifiable data registry** in the image below) to perform the cryptographic calculations necessary to prove the four attributes. Forgeries become much (MUCH!) harder with verifiable credentials!



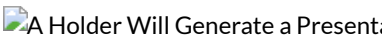
The Verifiable Credentials Model
By Peter Stokyo

(Continued on next page).

Verifiable Credentials (Cont.)

In essence, verifiable credentials are digital, cryptographically-protected data that you can use to prove you are you! With Indy, the data can be used by their holder to generate cryptographic zero-knowledge proofs (ZKPs—we will talk about these shortly) that can be checked by a verifier.

Note: For the more technically inclined, credentials are JSON docs, constructed and digitally signed by an issuer and countersigned by the holder.



A Holder Will Generate a Presentation That Will Be Checked by Verifier
By Peter Stokyo

Referring to the following illustration from the World Wide Web Consortium (W3C), there is an **issuer**, a **holder** and a **verifier**, just like in the paper credentials model. Unlike the paper credential model, which depends on the skill of the verifier to recognize an altered or forged credential, the verifiable credentials model includes a **verifiable data registry**—most importantly cryptographic keys and identifiers—that enables the proving of the data. That's the blockchain part of verifiable credentials.


The W3C Verifiable Credentials Model
The W3C Verifiable Credentials Model

Compared to the paper credentials model, verifiable credentials are far more secure. The verifier is not a person trying to identify a forged document. Instead, if the cryptographic verification succeeds, the verifier can be certain of the validity of the data—those four attributes stemming from verifying the presentation. However, there does remain a challenge with a verifiable credential—and it's actually the same thing with a paper credential: *Do you trust the issuer and the process by which the issuer decided to issue the credential*?

To trust the information in a credential, it is necessary that the verifier trust the issuer and that the issuer's processes are carried out with integrity. For an official government-issued document such as a passport, that may be fairly easy. But what about an educational transcript? For example, when you get a verifiable presentation from Faber College that says Alice has received a degree and you verify the presentation, you will know that Faber did indeed issue the credential to Alice, that Alice did not tamper with it and it's not been revoked. But is Faber College an authorized (and by whom?) degree-issuing entity or an unregulated diploma mill? Does it show that Alice legitimately earned her degree or did she pay $100 to be issued that degree? Later in the course we'll look at ways for a verifier to determine if they can trust the issuer.

Credentials and Claims/Proofs and Presentations

You probably noticed that in the previous sections we used the terms **credential** and **claim** seemingly interchangeably. Actually, we were quite intentional about it. Claims are assertions about a person (or business).

For example:

"The name of the holder is Bob. He was born on June 18, 2000"

or,

"Cloud Compass Computing was incorporated in 2011."

A credential is made up of a set of individual claims. For example, in the case of a driver's license, the claims would be a person's name, address, height, weight, hair color, driver's license number, and so on.


Claims versus Credential
Claims versus Credential
By Peter Stokyo

In all uses of a verifiable credential, what is issued is a credential. However, in some implementations, when a credential is presented the credential is proven, while in others (including in Hyperledger Indy), the claims within the credential are proven individually. As we'll see shortly when we talk about selective disclosure, this subtle difference can be quite important!

Another pair of terms that might seem to be used interchangeably are **proof** and **presentation**. A proof is evidence of the claim (for example, a birth certificate, passport or driver's license, or in the case of a business, incorporation papers). These terms are essentially equivalent. In the early days of Indy, the term proof was used when talking about a verifier requesting proven claims from a holder and the holder providing those claims to the verifier. Presentation was selected for essentially the same thing by the W3C Credentials Community Group. Further, the phrase "present a proof" sounds a little better than "present a presentation." As such we use both terms in this course. For developers, this is important because the underlying Indy and Aries code uses both terms as well.

Uses and Impact of Verifiable Credentials

So what does the availability of verifiable credentials mean? Well, in the past, you have probably shared identity attributes online as text (such as your name, address or government ID) or perhaps shared a paper credential you were given by scanning it and uploading it to an online service. Verifiable credentials are VERY different because

of the cryptography used to secure the verifiable credentials and presentations such that once verified, the information can be trusted to be authentic. That is really powerful!

For example:

- Instead of typing in your name, address and government ID, you provide a presentation of that information from verifiable credentials issued to you by an authority trusted by the verifier. The verifier can automatically accept the claims in the presentation without any further checking (saving them money).
- Anyone that knows your government ID (the string of numbers) would not be able to impersonate you on a site that checks verifiable credentials because the information they have does not come from a verifiable credential issued by the government.
- A doctor can be issued a professional accreditation credential from the relevant authority (e.g. the College of Physicians and Surgeons) and the claims verified (and trusted) by medical facilities in real-time. Should the doctor lose his or her accreditation, the credential can be revoked, which would be immediately in effect. This would hold true for any credentialled profession be it lawyers, engineers, nurses, tradespeople, real estate agents and so on.

The verifiable credential model has other important ramifications:

- Your verifiable credentials are issued to you, stored in your digital wallet, and you decide when and where you want to use them. That improves your privacy—you are in control of when and with whom you share your information.
- Verifiable presentation data is proven without needing to call back to the issuer. Just as when you use your driver's license to prove your age without a call back to the government, there is no need to call back to the government when you use the verifiable credential version of your driver's license.
  – Instead of getting the data directly from the issuer, the data from the issuer comes from the holder, and the cryptographic material to verify the authenticity of the data comes from the blockchain.
  – This reduces the number of integrations that have to be implemented between issuers and verifiers. Imagine if every company needed to integrate with every degree-issuing institution to see if a job applicant's claim of a degree was valid? That's a lot of interfaces to be built!
- Verifiable credentials go beyond identity to enable the digitization of almost any paper-based verification process. For example, regulatory processes that are currently implemented with paper submissions that are manually verified can be reworked to use verifiable credentials that provide authentic data that can be processed automatically.

What Is Self-Sovereign Identity?

Self-sovereign identity (SSI) is a term you may have heard in connection with the Hyperledger identity projects, Indy, Aries and Ursa. Together, these Hyperledger capabilities can be the basis for accomplishing SSI. So what is SSI—and what isn't?

There is No Central Authority with SSI
There is No Central Authority with SSI

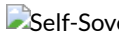
SSI is the idea that you control your own data, you control when and how it is provided to others, and when it is shared, it is done so in a trusted way. With SSI, there is no central authority holding your data that passes it on to others upon request. And because of the underlying cryptography and blockchain technology, SSI means that you can present claims about your identity and others can verify it with cryptographic certainty.

SSI is not about you self-asserting information about yourself and demanding that everyone else accept it as fact. Recall that the verifiable credentials you receive and present to verifiers come from others and their usefulness relies on the trust verifiers have in the credential issuers. You could issue credentials to yourself ("I am the Queen of England"), but verifiers might not trust you as an issuer.

Drummond Reed, identity guru and Founding Trustee of the Sovrin Foundation, describes self-sovereign identity as:

"*Lifetime portable identity for any person, organization, or thing that does not depend on any centralized authority and can never be taken away.*"

Sounds like something to strive for, doesn't it?


Self-Sovereign Identity Definition by Drummond Ree
Self-Sovereign Identity Definition by Drummond Reed

The Ten Principles of SSI

According to Christopher Allen, Co-chair of the W3C Credentials Community Group working on standards for decentralized identity and one of the first to use the term SSI, the ten principles of self-sovereign identity are:

1. Users must have an independent existence.
2. Users must **control their identities**.
3. Users must have access to their own data.
4. Systems and algorithms must be **transparent**.
5. Identities must be long-lived.
6. Information and services about identity must be **transportable**.
7. Identities should be as **widely used** as possible.
8. Users must agree to the use of their identity.
9. Disclosure of claims must be minimized.
10. The **rights of users** must be protected.

A number of these principles are technical in nature and are enabled by Indy, Aries and Ursa. Some are about how those technical projects are managed—transparency by (for example) using open source software based on well-defined standards. Others are about how the technology is used, and while Indy, Aries and Ursa focus on making it easy for entities to implement SSI solutions, governance—such as the European Union's General Data Protection Regulation (GDPR)—is still necessary. A lot of work has gone into creating the conditions necessary for achieving SSI that meet the ten principles outlined by Christopher Allen.

What Are Decentralized Identifiers?

While verifiable credentials are an important component of SSI, a thing called a **decentralized identifier** (DID) is a key enabler for verifiable credentials. DIDs are a new type of identifier that is in the process of becoming a World Wide Web Consortium (W3C) standard. As we discussed, the verifiable credential model requires a decentralized ecosystem in order to work. Such an ecosystem is brought about with DIDs and agents (discussed further in the chapter).

DIDs are a special kind of identifier that are created by their owner, independent of any central authority. Per the DID specification, a DID looks like the following and is similar to an HTTP address but used differently.


An Example of a DID
An Example of a DID

Note: This section gets a little technical, but don't worry, we'll go through this and then come back to what DIDs mean from a business perspective.

DIDs are:

- A new type of uniform resource location (URL).
- Created by anyone at anytime.
- Globally unique.
- Highly available.
- Cryptographically verifiable.

What does all that mean? Let's go through each of those attributes in turn:

- Like the URLs that we are familiar with (such as the one for this web page that is in the address field of your browser), DIDs can be resolved. When we pass a valid DID to a piece of software called a DID Resolver, it works like a browser given a URL, resolving the DID and returning a document. However, instead of returning a web page, a DID Resolver returns a DID Document (DIDDoc), a JSON document whose format is defined in the DID specification. We'll talk about the elements of a DIDDoc in a moment.
- A DID can be created by anyone, not just a central service. People, organizations and things can all create, publish and share DIDs.
- That DIDs are globally unique means that properly generated, each DID is unique across the globe. The "properly generated" caveat means that if you follow the rules for creating a specific type of DID, it will be globally unique. That's the type of detail that Hyperledger Indy takes care of for you.
- Highly available means that DIDs can be resolved even if some servers are down. Since DIDs are often resolved by reading from a blockchain, and there can be many copies of a given blockchain on servers across the world, DIDs are highly available. Put another way, DIDs are not susceptible to a central implementation with a single source of failure.
- Cryptographically verified means that the control of a DID can be verified by asking the DID controller to prove their control over the DID by having them prove control of a private key that is related to a public key.

This last point brings us to what is in the DIDDoc. Notably, a DIDDoc contains (usually) public keys whose private keys are held by the entity that controls the DID, and (usually) service endpoints that enable communication with that entity. This means that with a DID, you can:

- Resolve a DID to get a DIDDoc.
- Within the DIDDoc, find a public key and an endpoint for the entity that controls the DID.
- Use the endpoint to send a message to that entity.
- In the message, ask them for proof they have the private key related to the public key.
- Receive back that proof.
- Verify that proof.

DIDs from a Business Perspective

OK, we've seen technically what DIDs are, but why do we care? There are several characteristics of DIDs that are crucial from an SSI and verifiable credentials perspective.

User Control

DIDs are identifiers that you create, control and share under your control, which is a core tenant of SSI. Most identifiers that you use today are created or controlled by someone else:

- your government ID
- your phone number or email address
- your account at a particular website

If a central authority decides to take that identification away from you, they can, removing your ability to use it. Since you created a DID and hold the private key(s) for it, only you can "delete" the DID. A DID (and DIDDoc) on a blockchain can never be removed, so "deleting" a DID really means no longer responding to requests for proof of control of the DID. Technically, you can update the DIDDoc to delete the public key(s).

Provable

There are no easy, trusted ways for you to prove your control over commonly used identifiers today. From the list of identifiers that we mentioned, the only ones that you are commonly asked to prove control over are your email address and phone number, and those have become prime targets for hackers to hijack, often by attacking the central authority that gave you the identifier. The association of public/private keypairs with a DID makes ongoing proof of control trivial.

As we've talked about, one of the attributes proven with a verifiable credential presentation is the identity of the issuer. In fact, what is proven is just the public DID of the issuer and, because of the characteristics of DIDs, that's enough. From the DID, the verifier can get proof about who the issuer is, and further, (likely using verifiable credentials) their authority to issue credentials.

Non-Correlatable

One of the problems with the common identifiers that we use today is that it is often necessary to reuse them. On almost every site we visit, we provide an email address so that the site can send us notifications. That means that sites can share information about us just by correlating our email address. Since we create our own DIDs and choose how we want to share them, we needn't have just one DID, but rather one DID for every service to which we connect. The expectation is that you will have many DIDs—one for each relationship you have on the Internet. Note that both sides of a relationship provide a DID for the other to use to communicate with them, as shown below.

DID Relationships

Just as today, when you first register with a site you create an account, userID and password; with DIDs, you will create and share a new DID (and DIDDoc) with the site. When you return to the site, providing the DID and proving that you control the DID are sufficient for the service to know who you are. Hey—a way to eliminate user IDs and passwords!

DIDs from a Business Perspective (Cont.)

Secure Communications

As described earlier, a DIDDoc contains public keys and service endpoints for the entity that controls that DID. That enables a powerful capability: a secure, end-to-end encrypted messaging mechanism. Given a DID and its corresponding DIDDoc, an entity can encrypt a message for the controller of the DID (using the public key in the DIDDoc) and send it to the designated service endpoint. Upon receipt, the DID's controller can decrypt (using the corresponding private key) and process the message. We'll cover this capability a lot more in later chapters, including the fact that with this capability, there is no need to share an email address to receive notifications from a service. The service can just use the messaging capability inherent in DID communications.

Public Versus Private DIDs

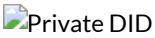
Let's look at the types of DIDs for a minute because there are important differences between public and private (pairwise) DIDs.

A **public DID** is one that is intended to be widely available—visible to anyone that is presented with the DID. Since anyone can both resolve the DID and contact the owner, if the DID is used many times, it is correlatable. In this case, correlation isn't a bad thing. A company or government, for example, will often use public DIDs. A prime example of a public DID is the DID of a verifiable credential issuer. Since the holder of the credential may present the credential to anyone, the identity (via the public DID) of the issuer must be part of what the verifier learns from the presentation. That way, the verifier can investigate (as necessary) the issuer to decide if they trust the issuer. Public DIDs are put on blockchains so that they can be globally resolved.

Public DID

On the other hand, an important use case of DIDs is that they enable two (or several) parties to connect with one another. In that case, it's not important that everyone in the world be able to resolve the DIDs for the parties—just the parties themselves. In that case, **private DIDs** (often called **pairwise DIDs**) are used. Instead of publishing the DIDs on the blockchain for anyone in the world to see, the entities create the DIDs and DIDDocs and then send both directly to the other party(ies) to hold. If they need to update the DID, such as to change the endpoint,

the update is sent directly to the other party(ies) to update their copy of the data. No one other than parties involved can see or resolve the DIDs.


Private DID

Private DID

Why are private DIDs important? There are two reasons. First, since the DIDs are only intended for use by a small, specific group (usually just two), the DIDs are not shared beyond that group. Second, writing to a blockchain is expensive and by NOT writing DIDs that don't have to be public to the blockchain, the costs (resources, time, transaction fees) are reduced. Happily, private DIDs far (far) outnumber the DIDs of verifiable credential issuers and as a result, the load on a public blockchain serving DIDs is massively reduced.

In the Aries section of the course we'll learn about the DID method called "did:peer," which defines the mechanism for sharing and using private, pairwise DIDs.

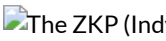
Review this demonstration to learn more about public and private DIDs.

Zero-Knowledge Proof

A **zero-knowledge proof** (ZKP) is about proving attributes about an entity (a person, organization or thing) without exposing a correlatable identifier about that entity. Formally, it's about presenting claims from verifiable credentials without exposing the key (and hence a unique identifier) of the proving party to the verifier. A ZKP still exposes the data asked (which could uniquely identify the prover), but does so in a way that proves the prover possesses the issued verifiable credential while preventing multiple verifiers from correlating the prover's identity. Indy, based on underlying Ursa cryptography, implements ZKP support.


ZKPs
ZKPs

There are other types of verifiable credentials that do not use a ZKP-based approach. In a non-ZKP proof, the holder/prover proves control over a DID to an issuer, and the issuer embeds that DID in the issued credential. Later, a verifier extracts the that DID from the credential and has the holder/prover prove they (still) control that DID. This proves the the credential was issued to the holder. Thus, a non-ZKP proof includes a public DID for the issuer (as with the Indy ZKP model) and a public DID for the holder/prover (unlike Indy). Usually in the non-ZKP model, the entire credential (with the embedded DIDs) is cryptographically signed by the issuer (proving the claims and the DIDs). Note the differences in this model and Indy's ZKP model. The DID of the holder/prover must be on a public ledger, and all verifiers of the credential know a common DID for the holder/prover—a point of correlation.


The ZKP (Indy) versus Non-ZKP Approach
The ZKP (Indy) versus Non-ZKP Approach

Selective Disclosure

The Indy ZKP model enables some additional capabilities beyond most non-ZKP implementations. Specifically, that claims from verifiable credentials can be selectively disclosed, meaning that just some data elements from credentials, even across credentials can (and should be) provided in a single presentation. By providing them in a single presentation, the verifier knows that all of the credentials were issued to the same entity. In addition, a piece of cryptography magic that is part of ZKP (that we won't detail here—but it's fascinating) allows proving pieces of information without presenting the underlying data.

A useful example of both selective disclosure and a ZKP supported in Indy is a proof that a person is older than a given age based on the date of birth in a verifiable credential driver's licence without disclosing any other information, including name or date of birth. The image below demonstrates how that might look in an SSI-enabled app at a pub.

An Example of Selective Disclosure
By Peter Stokyo

The verifier (a bartender) can confirm:

- The issuer is the appropriate authority.
- The picture shows the same person presenting the verifiable credential.
- The person is old enough to drink in the pub (the check mark).
- The credential has not been revoked.

That information is proven without the person having to reveal their personal information (address, eye color, etc.), or even their date of birth.

Combined, ZKPs and selective disclosure enable a massive reduction in the data exposed during an identifying process. By not revealing a correlating identifier, and using selective disclosure to only expose the data needed for a transaction, including being able to prove possession of a credential without disclosing any information from the credential at all, is an important step in being able to keep private data private.

Important to remember is that ZKPs are not a panacea for privacy. For any business transaction, the verifier must collect enough information from the prover as to mitigate their risk for the given transaction. That may mean requesting enough claim data to uniquely identify the prover. For example, the verifier may be required by government regulations to collect identifying information about the prover for auditing purposes. Selective disclosure enables them to be much more precise in collecting that data than existing technology, but does not eliminate the collection entirely. As well, once claims data is given to the verifier in a presentation, the verifier has that data, and can technically (if not legally) do anything they want with the data. There is no technical capability in this model that allows, for example, a prover to "revoke" a presentation and take the data back from the verifier.

Agents and Wallets

A third key component of SSI, along with verifiable credentials and DIDs, is the software that you use to process verifiable credentials and DIDs—what we've called your digital wallet to this point in the course. Indy, Aries and Ursa use the term **agent** to mean the software that interacts with other entities (via DIDs and more, as we'll find out). For example, a person might have a mobile agent app on their smart device, while an organization might have an enterprise agent running on an enterprise server, perhaps in the cloud.

All agents (with rare exceptions) have secure storage for securing identity-related data including DIDs, keys and verifiable credentials. As well, all agents are controlled to some degree by their owner, sometimes with direct control (and comparable high trust) and sometimes with minimal control, and far less trust. We'll cover different uses and types of agents in another chapter.

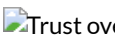
A Mobile Agent App
By Peter Stokyo

For those familiar with a password manager such as 1Password or LastPass, you'll find a personal agent is quite similar—there is a name for each relationship and associated data. However, unlike password managers that use things such as your copy/paste clipboard for user IDs and passwords and screen-scrape applications and websites, agents communicate directly with agents to accomplish identity-related tasks. As agents use messaging to communicate with one another, users will also see a similarity between agents and the messaging apps we use today, such as WhatsApp and Facebook Messenger. Unlike those apps, agents use messaging for far more than just sending text, images and links, including for transactions around the exchange of credentials and presentations. Further, we expect agents to be the cornerstone of self-sovereign identity, which is very different from the foundations of those apps. We dig into agents in Chapter 5 of this course.

Trust Over IP (ToIP)

Along with SSI, another term you will hear surrounding Indy, Aries and Ursa is **trust over IP** (ToIP). ToIP is a set of protocols being developed to enable a layer of trust on the Internet, protocols embodied in Indy, Aries and Ursa. It includes self-sovereign identity in that it covers identity, but goes beyond that to cover any type of authentic data. Authentic data in this context is data that is not necessarily a credential (attributes about an entity) but is managed as a verifiable credential and offers the same guarantees when proven.

ToIP is defined by the "Trust over IP Technology Stack," as represented in this image from Drummond Reed:


Trust over IP (ToIP) Technology Stack
Trust over IP (ToIP) Technology Stack

You should recognize most of the components of the ToIP technology stack from what we've already talked about in this course. At the bottom left, the foundation of the stack are public blockchains that store decentralized identifiers (DIDs) and (in some cases) other data necessary for the higher layers. Next are agents and the protocols that enable agents to establish connections and exchange messages (information). We'll talk about agents in Chapter 5. The next layer up is the verifiable credential layer that enables trusted information flow—data cryptographically signed by the participants during issuing and verified during presentation. Note the coloring of the layers on the left side: "technical trust" (shown in blue) is enabled by the cryptography and "human trust" (shown in pink) is enabled by the governance frameworks.

We'll talk about **governance frameworks** later in the course. That top layer, and the associated governance frameworks on the right, are the rules that the participants in the network agree to use to enable the system to work in practice. Example governance frameworks include the following topics we'll cover in this course:

- How the readers and writers of the blockchain operate.
- How the agents know meaning of messages that form a protocol.
- What the claims in credentials signify to the verifier.
- By what authority an issuer is able to issue a credential.

For the purposes of this course, we will talk largely about self-sovereign identity (SSI)—it's a big topic. Do remember though as you consider the capabilities of this technology that it can be applied beyond identity and into any domain where authentic data is important. The following are a few examples:

- Many supply chain use cases can be handled using authenticate data. As events occur to items being tracked (e.g. transferred from one carrier to another) verifiable credentials can be issued to capture the state of the item and event. Verifying the claims in the credential provide proof of that data later, demonstrating the provenance of the item. Food, shipping, diamonds, oil and gas and many, many more supply chains can be tracked this way.
- Regulatory processes that are based on paper trails can be replaced with authentic data. Rather than relying on paper for documenting and manually verifying compliance, verifiable credentials could be generated from the source and digitally verified. Since the verifiable credentials are generated at the source and signed (immutable), they provide a reliable audit trail of events and actions.
- The Government of British Columbia has demonstrated the use of authentic data in the handling of mine inspections in BC. Rather than a notebook being used for tracking mine inspections, the inspection data is digitally collected, recorded in a database and an inspection verifiable credential generated about the report that guarantees its authenticity. Should there be a need later to present the report in court (for example) the inspection credentials can be proven, verifying the accuracy of the inspection report.

Note that in all of these cases, even though the use case is not centered on identity, there is an element of identity in each one. Specifically, in call cases, the identity of the credential issuer must be known to determine if they have the authority and capability to issue the credential, and the holder/prover must be able to generate a presentation of the claims in the verifiable credential.

Summary

We've introduced a lot of new capabilities in this section and along the way alluded to how these capabilities help solve the Internet identity problems outlined in Chapter 1. Verifiable credentials, DIDs and agents are at the core of self-sovereign identity and although they get you a long way towards solving the problems we talked about in Chapter 1, we need to provide a little more background about the technical elements that make a new trust layer for the Internet possible. In the next chapter, we will learn about the software components necessary to receive, store and transmit both verifiable credentials and DIDs.

Let's go back to bullet points in the Chapter 1 summary and see how the capabilities introduced in this chapter address the problems we find on the Internet today:

Internet Challenges Today	How Verifiable Credentials, SSI and ToIP Address These Challenges
User IDs/passwords are the norm, but they are a pain to use, and as a result, are susceptible to attack. They are the best we have right now, but not a solid basis for trust. Further, IDs work only one way—users don't get an ID from a service they use.	Websites can use DIDs and, as needed, verifiable credentials to get enough information about users to establish sessions. We'll see later that using DIDs and verifiable credentials are a lot easier and safer for users than passwords.
Other personal information and identifiers we have that we could otherwise use to prove our identity are not trusted because it's impossible to tell if the data was actually issued to the person providing it. The many breaches of private identifiers make them impossible to completely trust, and verifying that information adds (sometimes significant) costs.	Presentations of claims from verifiable credentials, and knowing who issued the credentials mean that we can trust the claims to be correct. This is based on the four attributes we learn when verifying a presentation.
Since the identity attributes we could use are not trusted (they are not things only we know), we often have to resort to in-person delivery of paper documents to prove things about ourselves, a further cost for all participants.	We can use verifiable credentials online instead of having to use paper documents in-person.
Reviewers of the paper documents we present must become experts in the state of the art in forging and falsifying paper documents, a difficult role in which to be placed.	The trust of the claims is in cryptography and knowing about the issuer. People don't have to be experts at detecting forgeries.
The identifiers we use are correlated across sites, allowing inferences to be made about us, and exposing information we don't intend to be shared across sites. This is annoying at the least, and can have catastrophic results in the worst case.	By using private DIDs and verifiable credentials based on ZKPs, we can radically reduce the online correlation that is happening today.
Centralized repositories of identifiers and data about the people associated with those identifiers are targeted by hackers because the data has high value. This exacerbates the problem of not being able to trust "personal" data presented online (see above).	If high value data is only accepted when presented as a claim from a verifiable credential, there is no value in having data from breaches.
Centralized identifiers can be abused by those who control those identifiers. For example, they can be taken away from a subject without due process.	You create your own DIDs and those identifiers cannot be taken away by a centralized authority. You control your DIDs, no one else.

Chapter 3: SSI Using Indy, Aries and Ursa

Chapter Overview

In Chapter 2, we introduced the concept of self-sovereign identity and how the verifiable credentials model adds a needed layer of trust to the Internet. In this chapter, we will introduce Hyperledger Indy, Aries and Ursa and how the technologies from these projects (surprise!) enable SSI. Along the way we'll continue to dig into the verifiable credential model and learn more about decentralized identifiers, agents, zero-knowledge proofs and selective disclosure.

Learning Objectives

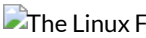
By the end of this chapter you should:

- Have an understanding of how each of the Hyperledger identity projects came about.
- Be familiar with the capabilities of each of the Hyperledger identity projects.
- Be familiar with how agents communicate and exchange credentials.
- Have some hands-on experience working with agents!

Hyperledger Indy, Aries and Ursa – History

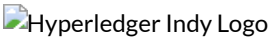
Here we are, into the third chapter and we've barely mentioned anything about the stars of the course: Hyperledger Indy, Aries and Ursa. We've talked about the problems with trust on the Internet in Chapter 1. In Chapter 2, we presented some new approaches enabled by blockchain technology that are the building blocks of decentralized identity—approaches that make SSI possible! And, as you might be able to guess, Hyperledger Indy, Aries and Ursa are (awesome) implementations of those building blocks. However, before we dig too deeply into these projects, let's introduce them by way of a brief history lesson on how they came about in the first place.


The Linux Foundation Frameworks and Tools


The Linux Foundation Frameworks and Tools

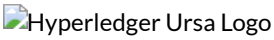
Hyperledger Indy Overview

Hyperledger Indy was Hyperledger's first "identity-focused" blockchain framework, joining Hyperledger in 2017. The code for Indy was contributed by the Sovrin Foundation, an organization that we'll cover a little later, after we've dug a bit deeper into Indy. Indy is a purpose-built distributed ledger for decentralized identity, supporting the concepts we talked about in Chapter 2. Indy includes verifiable credentials based on zero-knowledge proof (ZKP) technology, decentralized identifiers, a software development kit (SDK) for building agents and an implementation of a public, permissioned distributed ledger. We'll be talking more about all of these things in this and later chapters.



Hyperledger Ursa Overview

As Indy evolved within Hyperledger, there was a realization that the cryptography in Indy could be used in a number of Hyperledger projects, and even outside of Hyperledger. In 2018, the decision was made to migrate the **indy-crypto** code repository out of Indy and into its own project: Hyperledger Ursa. Like the Linux kernel, cryptographic algorithms and software are the domain of a few experts in the field and should not be coded by other than those experts. Ursa gives those experts a place to work together to build and package cryptographic primitives—combinations of low-level cryptography elements that only experts should build, test and make available to "ordinary developers". Just as important, it's an open source space for experts to review those primitives and find and eliminate flaws in the implementations. Ursa packages the primitives in a way that can be consumed by Indy, Aries and any other software that needs a solid, vetted cryptographic base. The initial transfer of code from Indy to Ursa was pretty straightforward, involving a lot of renaming and updating references, but little change. As we'll see, splitting up a project isn't always as straightforward.



Hyperledger Aries Overview

The **indy-sdk** repository is the Indy software that enables building components (called agents) that can interact with an Indy ledger and with each other. Until mid-2018, groups working on building on top of Indy did so in silos, making agents that talked only to other instances of their agent, not to agents built by others. The *Indy Agents Working Group* was established to change that, by defining standard protocols to enable agent interoperability. By early 2019, at a "Connect-a-thon" in Utah, USA, the fruits of that effort were realized as developers from a variety of organizations gathered to demonstrate interoperability across a set of independently developed agent implementations. At that time, a further idea developed that led to the creation of Hyperledger Aries. Although Indy is an excellent implementation of SSI capabilities, it is not the only implementation. In the long term, it's likely that there will be multiple implementations that are used by different people, organizations and communities. What if we had agents that could use DIDs and verifiable credentials from multiple ecosystems? Aries was proposed on that basis, and accepted as a Hyperledger project in March 2019.

Aries is a toolkit designed for initiatives and solutions focused on creating, transmitting, storing and using verifiable digital credentials. At its core are protocols enabling connectivity between agents using secure messaging to exchange information. Aries is all about peer-to-peer interactions between agents controlled by different entities—people, organizations and things. Using the standardized messaging channel, verifiable credentials can be exchanged based on DIDs rooted in different ledgers (based on Indy or other technology) using a range of verifiable credentials implementations.

Unlike when Ursa was separated from Indy, the separation of Aries from Indy is, well, complicated, and as this course is being written, the approach is still being planned. Some elements will clearly remain with Indy (for example, ledger interactions), others will clearly move to Aries (for example, secure storage). But there's a lot in the middle that might fit in both places, such as verifiable credentials exchange algorithms. Initial Aries agents have been built using Indy as a base (changing little from "Indy agents"), but the approach is evolving to meet the vision of Aries.

The Hyperledger Identity "Stack"

Together, Indy, Aries and Ursa make the Hyperledger Identity "Stack."

The Hyperledger Identity "Stack"

The logo sizes in the image are intentionally different, conveying the size of the customer base for each of the projects. The vast majority of those working with the Hyperledger identity stack will build on top of Aries and have relatively little interaction with Indy, and almost none with Ursa. Only those contributing code directly into Indy, Aries and Ursa will have much interaction with Indy and Ursa.

Demo Time

With that bit of history out of the way and a basic understanding of the three projects and their relationship to one another, let's go through an example of verifiable credentials being created, issued, held and then proven. We'll go step-by-step through a demonstration of the interactions and how the different components of Indy, Aries and Ursa come into play.

To see the demonstration and optionally run it yourself, click on this link. Please come back here when you are done!

Introduction

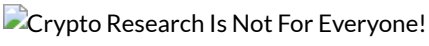
The Hyperledger Ursa project produces cryptographic packages that can be used to build higher level applications. The packages Ursa produces are used by Indy and Aries for all of the uses of cryptography by those projects, including:

- generation of public/private key pairs
- data encryption and decryption
- data signing and verifying
- data hash generation and verification
- zero-knowledge proof (ZKP) technology, including issuing ZKP credentials and generating and verifying ZKPs

These capabilities are used throughout Indy and Aries, as we will see when we get deeper into these projects. The need for and importance of Ursa is best understood by looking at how cryptography, an extremely complex technology, is able to be used in everyday applications. The following is a brief summary of how cryptographic algorithms are developed, implemented, standardized (in libraries such as Ursa) and deployed.

Basic Research

Core cryptographic techniques are the work of mathematical researchers. They think of and subsequently prove (or not) the viability of the algorithms using pure math. It is those researchers that are responsible for the cryptographic capabilities we use in everyday life. But for the average software engineer, their work is inaccessible—the concepts are too complex to convert to software requirements and products.


Crypto Research Is Not For Everyone!
Crypto Research Is Not For Everyone!

Over time, such researchers have created breakthrough algorithms, and have evolved and enhanced existing algorithms to improve their usefulness and effectiveness. For example, creating a new encryption algorithm that is more effective than existing techniques but too slow to be useful, and then evolving the algorithm to be practically useful. The well-known Diffie-Hellman algorithm was one such breakthrough, as was the use of elliptic-curve cryptography.

"Effective" in practical cryptography means that an algorithm is both useful and (relatively) safe from attack. An encryption algorithm is effective if the data cannot reasonably be decrypted except with authorized access by th intended recipient. For example, if the encryption algorithm uses a key, the key cannot easily be discovered, or th data cannot "easily" be decrypted without the key. Often that means a brute force attack (e.g. trying every possible key) cannot be completed in a reasonable time (e.g. at least 10s to 100s of years) using technology anticipated to be available some period in the future. There are many such attack vectors and both cryptographers and their adversaries (aka hackers) are constantly thinking about what new attack vectors are possible.

Implementation

The next step towards practical use of the research results is the implementation of the algorithm in code—converting the mathematical equations into software. Again, that is an exclusive domain of expert researchers/developers that know the math, the code and where weakness might show up in the algorithm or (more likely) the implementation. Implementations are packaged into well-known and well-tested open-source libraries (for example, libsodium). It is common for the algorithm to be named and evolutions of the algorithm to be given versioned names (for example, elliptic curve algorithms, with variations for different purposes (e.g. signatures, hashes, etc.), different parameters (e.g. key length), with versioned standards defined by (often) the year of implementation). An example is:

Ed25519VerificationKey2018

which is interpreted as "standard signature suite created in 2018 for the Ed25519 signature scheme using Curve25519, currently used by Hyperledger Indy."

You really have to be paying attention to follow and understand all the options! What non-crypto developer has time for that? Further, having an algorithm and using it properly, without introducing vulnerabilities in the security of an application, is yet another challenge. That's where Ursa comes in.

The “open source” reference in the previous paragraph is extremely important. With trust comes verification, and the open sourcing of cryptography implementations is crucial to establish trust in the implementation. Very few put trust in such important code that they can’t evaluate, line by line and build from scratch.

Packaging

Ursa takes the “raw” cryptographic algorithms and packages them up so that they can be embedded in, for example, Indy and Aries and used safely. Ursa curates what implementations to support and how to manage support for different versions as the implementations evolve (e.g. 2018 versus 2019). Where higher level standards have been defined, such as the so-called JW* mechanisms (JWE, JSON Web Encryption, JWT, JSON Web Token, JWS, JSON Web Signatures, etc.), Ursa takes care of the details in producing and consuming those data structures.

Usage

With the Ursa packages embedded in Indy and Aries, cryptographic features are relatively easy to implement. When a write transaction is sent to an Indy ledger, it is signed (via a call to an Ursa function) by the transaction author and the signature is verified (via a call to an Ursa function) by the nodes of an Indy ledger that receive the transaction. When a message is sent from one Aries agent to another, it is encrypted and packaged up via a call to an Ursa function.

If you are contributing to Indy and Aries, you do have to be aware of the Ursa libraries being used and the capabilities they provide. However, if you are building applications on top of Aries and using the Indy ledger and verifiable credentials, it’s all hidden from you. Calls your application makes to Indy and Aries functionality in turn make calls to Ursa to handle the cryptography needs. In fact, the main impact the cryptography will have on developers is not how to use the cryptography (phewww!!), but rather trying to debug code when so much of the data is encrypted. That can be a major pain—but MUCH easier than having to write safe and secure cryptography code!

That’s all we’ll cover on Ursa itself in this course. If you are trained in cryptography and interested in working at the cryptography level on Ursa, you’ll know a lot more about this topic than we’ve covered here. Hop right over to the Hyperledger Ursa Rocket.Chat channel and starting talking with the project maintainers!

For everyone else, we’ll give a quick overview of Hyperledger Indy and Aries in this chapter and cover each of them in depth in the following chapters.

Introduction

Hyperledger Indy is the core component of Hyperledger’s identity system. Indy provides code that implements the public **distributed ledger technology** (DLT) and the code to build applications that interface with the ledger. The project consists primarily of two repositories (repos) that contain those two parts:

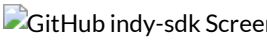
- **indy-sdk**: a software development kit that enables Indy clients (called “agents”) to be built that interact with the Indy blockchain
- **indy-node**: the blockchain/DLT component of Indy

Let’s look into each of these components.

indy-sdk

The **indy-sdk** repository contains the code that allows a piece of software, called an agent (introduced in the previous chapter and that we’ll cover in detail in Chapter 5), to interact with the public ledger and to keep track of the keys and other identity-related data. The **indy-sdk** consists of a series of modules written mostly in the programming language Rust that are compiled into a “C-callable” library (called “libindy”). C-callable means that the library components have a standard interface such that they can be called from a majority of languages (Java, C#, Python, JavaScript, etc.), eliminating the need for an implementation in each language. The **indy-**

sdk repository also includes a number of language specific "wrappers" for the library. The wrappers, currently available in Python, Java, C# (.NET) and node.js, allow the creation of Indy agents in each of those languages.



Let's go through the major subsystems of the **indy-sdk**, covering some important details about each. As this course is being written, discussions are occurring in the Indy and Aries community about how to split up the **indy-sdk** across Indy and Aries. In the sections below, we'll cover how the split might affect the subsystems, including whether it will remain in Indy or move to Aries. While that doesn't matter so much for those that are building SSI applications on top of Aries, it does matters if you or your team wants to be a contributor to Indy or Aries itself.

indy-sdk: Indy Ledger Client

Core to the **indy-sdk** is the code that enables an Indy agent to interact with an instance of an Indy network. The functionality is implemented as a series of messages that enable the client application to read from and write to an Indy network. The high level process for interacting with the ledger is the same for each call:

- The application prepares the data for the message to be sent, e.g. the type of message and all of the parameters necessary for that type of message.
- If the message is a write, the application calls cryptographic functions to sign elements of the message with a private key held by the application for that purpose. This is necessary to prove that the application has permission to write the transaction to the network.
  – Multiple signatures from different parties may be needed depending on the roles of the parties and the impact of the message on the network. For example, if the application is not authorized to write to the ledger, the transaction must be sent to an application that is authorized to write to the ledger (called an "endorser") to sign and transmit transaction.
- The message is sent to a node on the ledger.
- The application waits until a response is received from the ledger with the result code (success or failure) and any related data.
  – If a write transaction succeeds, one of the returned values is the transaction's sequence number on the ledger. That information should be stored immediately by the agent so it can be used when referencing that object in later operations.

Most of the transactions relevant to participants in verifiable credential exchanges (issuers, holders/provers and verifiers) are covered in the next chapter when we talk about what goes on the blockchain. As well, there are a number of transactions that control the operation of the ledger itself, such as adding and removing verifier nodes from the network and controlling permissions about who can write what ledger transactions. We'll not cover those further in this course, but are relevant to those who want to participate in the operation of an Indy network.

The read transactions support retrieving all of the types of transactions that are stored on the ledger. For each type, the agent must know either the transaction sequence number of interest, or a primary key for the item to be read. For example, the primary key for a DID is the DID string (e.g. **did:sov:1213423e239f9g095hg2**).

A surprise for many that are new to Indy is its ledger does not support any form of querying by value or discovery. In the common cases, the agent must somehow know about the transactions of interest on the ledger, either because that agent wrote the data in the first place, or because the agent is configured ("hardwired") to know about the transaction. Although you can't query the ledger, it is a public ledger, so you can read its entire contents and if you want a query capability, you can put it in a database. How? Since the transactions are sequenced, you can just start from one and iteratively query the ledger for all the transactions (1, 2, 3, 4…). Once you have read them all (e.g. the ledger returns "No such transaction"), every few minutes, ask for any new transactions that have arrived. It's onerous and so several people have published websites of "ledger browsers" (e.g. Sovrin Main Net and IndyScan) that continually query the Sovrin MainNet and provide a web interface for searching. Here is a short demonstration about browsing an Indy ledger.

Note: Adding a querying/discovery capability that works in a similar way to the ledger browsers but that is implemented by Indy is likely not a difficult task. It's just not risen high enough on the priority list to be

implemented as a core feature.

The ledger interface capabilities in the **indy-sdk** will remain in Indy as the SDK capabilities are split across Indy and Aries. There will be a Verifiable Data Registry (to use the W3C term) interface created in Aries that in turn calls the Indy SDK ledger interface. That interface will also provide interface implementations for other DID methods built on other blockchains.

indy-sdk: Indy Storage

As mentioned, the **indy-sdk** implements a secure storage component that is used to store the Indy data collected by an identity on an Indy network—DIDs, private keys, verifiable credentials and more. Unfortunately, in the early days of Indy, the term "wallet" was selected for the storage component of the **indy-sdk**, resulting in an application called a digital wallet containing a wallet. Confusing, eh? That term will be dropped when the storage component of the **indy-sdk** is moved to Aries, where it will be called the **key management service** (KMS).

The **indy-sdk** provides a pluggable API for the storage, meaning a capable developer can create another storage implementation should they need one. By default, Indy uses the open source SQLite database, and the Government of British Columbia implemented and contributed to Indy an implementation using the PostgreSQL database for enterprise deployments.

The Indy storage data model is simple with just a handful of tables implementing essentially a key-value pair system. Almost all of the data is encrypted, a detail we'll cover later in the course. The storage is logically divided into two parts:

- "Secrets"—data associated with ledger entries and verifiable credentials, and
- "Non-secrets"—any other data stored by an application in the database

In all cases, for security reasons, operations that use private keys happen within the storage API code—the private keys are never handed out to "user code" (less scrutinized application code that might have security vulnerabilities).

indy-sdk: Verifiable Credential Handling

Indy verifiable credentials handling is currently built into the **indy-sd**k. That code is used to create credentials to be issued, generate presentations to be provided to verifiers, and to verify those presentations upon receipt. As necessary, that code calls the ledger interface to retrieve the necessary ledger data to carry out those operations. We've already gone into detail about those operations (and we'll talk about them again), so no need to dig too much into them now.

The verifiable credential capabilities in the **indy-sdk** will be moved to Aries as the SDK capabilities are split across Indy and Aries. That change is not as obvious as the ledger interface staying in Indy and storage moving to Aries, but is the right option. The Aries verifiable credential capabilities will support plugins, of which one that uses Indy will be but one.

indy-sdk: Wrappers

As noted earlier, most of the **indy-sdk** is written in Rust and provides a C-callable interface to the functions of the SDK. The wrappers are language specific bindings that make it easy for an application in another language to call the Rust functions. The wrapper functions tend to be one-to-one with the Rust calls. Although most of the wrappers were hand-coded, the ideal is that there is so little extra value in the wrappers that the code can be machine generated. Auto-generation would eliminate the manual effort to update the wrappers as the Rust code evolves.

*Note: As capabilities in the **indy-sdk** are moved to Aries, corresponding wrapper calls will also be moved.*

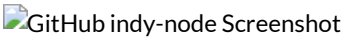
indy-sdk: Tests

A key part of the **indy-sdk** is the test suite that is automatically executed with each update to the repo. While it's comforting to know the test suite is executed (and updated) with each change, another very useful aspect of the suite for those new to Indy is that the tests are working code that can be used to understand how the **indy-sdk** works. Not sure what exactly are the parameters to a specific Indy call? Check out the test suite and you'll find a working instance to study.

*Note: As capabilities in the **indy-sdk** are moved to Aries, corresponding tests will also be moved and new tests added.*

indy-node

The second major component of Hyperledger Indy is **indy-node**, the blockchain/distributed ledger (DL) component of Indy. Written in Python, this codebase embodies all the functionality to run nodes (validators and/or observers) that implement a distributed ledger. We're going to talk a lot about the ledger and what data goes on the ledger in the next chapter, so we'll not spend any more time on it here.



Introduction

Indy (and its underlying Ursa cryptography) enables issuers, holders and verifiers to exchange verifiable credentials and presentations by exchanging sequences of messages. All those issuers, holders and verifiers will use agents (software) built by many different organizations—open source tools, commercial agents available for purchase, and custom-built proprietary agents. How do all those different agent makers build agents that will work with all the other agents? Further, what if you decide you want to switch from the agent you have now to a new one that is even better? You must be free to take your credentials and related data, move them to another agent and continue to use them.

Those are the challenges that started the Hyperledger Aries project. Aries is about collaborating to define and share the message exchange protocols, an agent architecture and tests that enables organizations to independently build agents that work together. Since there are other verifiable credential ecosystems being developed in parallel to Hyperledger Indy, Aries is also intended to be "verifiable credential-agnostic"—the architecture of Aries should be able to support different verifiable credential implementations within a single agent. And, since Hyperledger and the Linux Foundation is about building open source software, Aries includes implementations of the protocols, architecture and tests.

It's All About Message Protocols

Messaging in Aries is defined at several conceptual layers. At the lowest is just the ability for one agent to send a chunk of data, and for another agent to receive that data and understand it. At the next layer is the ability for agents to exchange a sequence of messages to accomplish some shared task. That's a protocol. Messaging protocols are just like human protocols. For example, going out to a restaurant for a meal is a protocol, with both the guests and the restaurant staff knowing the sequence of steps for their roles—greeting, being seated, looking at menus, ordering food, etc. In Aries, an example is the issue credential protocol, where an issuer agent coordinates with a holder agent to offer, be asked for and deliver a credential. Unlike human protocols (etiquette), Aries messaging protocols need to be carefully specified and then implemented (and tested!) by multiple participants.

Aries provides messaging at both layers. In Chapter 5, we'll dig into Aries messaging, covering the DIDComm (DID Communications) protocol that covers the envelopes used to address and send messages from agent to agent, and content protocols—messages with specific data used between agents to accomplish a task. As indicated by the name, DIDComm uses the data associated with DIDs (public keys and service endpoints) to secure and address messages, respectively. DIDComm is an extremely powerful capability!

And Interoperability

As we've discussed in Chapter 2, Indy is not the only game in town when it comes to DID and verifiable credential implementations. With a goal of global use of verifiable credentials (and all the goodness that brings), it's crucial that interoperability across different implementations is a goal. Not all implementations will last, but we're certain that there will be more than one, and we will need interoperability across the major implementations. The Aries architecture defines how different implementations can be plugged into an agent to support Indy and other DID and verifiable credential implementations at the same time. The goal is to bring developers from those communities together to extend Aries to enable verifiable credential interoperability.

And Portability

In the previous section we talked about implementations of DIDs and verifiable credentials. But there will also be many implementations of Aries agents, built by many different organizations. As a person, organization or thing begins to use a digital wallet, they will immediately begin to collect data that is valuable—things they don't want to lose. And, since digital wallets are just specialized pieces of software, they will at some point discover a digital wallet that is even better than the one they started with. Or perhaps they one they started with will become obsolete—no longer supported. One part of the Aries definition (albeit one that has not yet had a lot of focus) is about data storage, and more importantly the export of data from one implementation and the import of that data into another without loss.

Summary

This is the linchpin chapter of the course and should be considered carefully. From this chapter we hope you understand how the problems with identity on the Internet (Chapter 1) and the solutions defined based on self-sovereign identity and the verifiable credentials model (Chapter 2) are manifest in Hyperledger Indy, Aries and Ursa. In subsequent chapters, we'll delve deeper into the projects by looking at what goes on the blockchain (Chapter 4) and how there is more than meets the eye with agents (Chapter 5).

As we did in Chapter 2, let's go back to bullet points in the Chapter 1 summary and see how Indy, Aries and Ursa address the problems we find on the Internet today:

Internet Challenges Today	How Indy, Ursa and Aries Address These Challengess
User IDs/passwords are the norm, but they are a pain to use, and as a result, are susceptible to attack. They are the best we have right now, but not a solid basis for trust. Further, IDs work only one way—users don't get an ID from a service they use.	Aries agents used by individuals and online services exchange at least DIDs, and often verifiable credentials, to reliably identify one another as peers each time they connect. Ursa cryptography underlies the exchange, and Indy credentials are exchanged as needed.
Other personal information and identifiers we have that we could otherwise use to prove our identity are not trusted because it's impossible to tell if the data was actually issued to the person providing it. The many breaches of private identifiers make them impossible to completely trust, and verifying that information adds (sometimes significant) costs.	When collecting information from users, services can use their Aries agent to connect with, request and receive back proofs of claims from a users Aries agent.
Since the identity attributes we could use are not trusted (they are not things only we know), we often have to resort to in-person delivery of paper documents to prove things about ourselves, a further cost for all participants.	The cryptographic guarantees from Ursa and Indy inherent in the proving of claims allows for the delivery (via Aries agents) of digital data in place of paper.
Reviewers of the paper documents we present must become experts in the state of the art in forging and	The cryptographic guarantees from Ursa and Indy inherent in the verification of claims replaces human

falsifying paper documents, a difficult role in which to be placed.	verification. Humans may be needed to assess the trustworthiness of the issuer, but not of the data.
The identifiers we use are correlated across sites, allowing inferences to be made about us, and exposing information we don't intend to be shared across sites. This is annoying at the least, and can have catastrophic results in the worst case.	Aries agents use private, secure messaging to enable interactions without monitoring or correlation exposure. Indy credentials use ZKPs to reduce correlation when proving credentials. Ursa provides the necessary cryptographic primitives to support Aries messaging and Indy credentials.
Centralized repositories of identifiers and data about the people associated with those identifiers are targeted by hackers because the data has high value. This exacerbates the problem of not being able to trust "personal" data presented online (see above).	Aries agents make it easy to decentralize the data, giving it to the subject of the data (us!) to use how and when we want. At the same time, this frees organizations currently managing those repositories from the liability of holding such toxic data. Indy provides the credentials that ensure that the data provided by the subject can be trusted. Ursa provides the cryptography to support the Indy credential model.
Centralized identifiers can be abused by those who control those identifiers. For example, they can be taken away from a subject without due process.	Aries enables you to create (and delete) DIDs that you share with others how and when you want. Indy allows you to put those DIDs on a global ledger for others to access, if that is necessary. Ursa provides the cryptography that allows you to prove control over those identifiers.

Chapter 4: A Blockchain for Identity

Chapter Overview

Hyperledger Indy implements (based on cryptography from Ursa) a purpose-built blockchain specifically for identity on the Internet—enabling certainty about who is talking to whom in a digital transaction. Since Indy is designed for such a special purpose, how does that affect the underlying blockchain implementation? Does it have the same components of other blockchains? What goes into the transactions on the blockchain?

In addition to talking about the Indy blockchain, we'll talk about the Sovrin Foundation, introduced briefly in the last chapter. The Sovrin Foundation is a non-profit that governs the operation of a global, multi-organization, production instance (and several test instances) of the Hyperledger Indy blockchain. We'll talk about how Sovrin makes it even easier to get started using verifiable credentials because you don't have to operate your own instance of Indy.

Learning Objectives

By the end of this chapter you should:

- Be familiar with different types of blockchain and the terminology surrounding them.
- Understand how consensus is reached on the Indy blockchain.
- Know what the Sovrin Foundation is and how it is connected to Indy.
- Be familiar with the term trust or governance framework.
- Understand what gets written to the ledger and more importantly, what does not!

What Is Indy's Blockchain?

In most courses on blockchain-based solutions, the core focus is about how the blockchain works and how applications interact with the blockchain component. Not so much with Indy, Aries and Ursa. The Indy blockchain plays a crucial role in enabling trust on the Internet, but when you are building applications using the technology, the focus is much more on the agents, relationships between agents and on the entities that control them. The blockchain is hidden behind the scenes, operating on a relatively small number of nodes. Regardless of its visibility, it's important in this course that we go over the implementation of Indy's blockchain. In the following, we'll cover how Indy operates versus other blockchains, how consensus is reached and so on.

Note: As mentioned at the beginning of this course, we use the terms blockchain, ledger, decentralized ledger technology (DLT) and decentralized ledger (DL) interchangeably. While there are precise meanings of those terms in other contexts, in the context of the material covered here, the differences are not meaningful.

Like all blockchain implementations, an Indy ledger is immutable—written once and never changed. Since Indy is focused only on identity, it does **not** support the concept of assets being exchanged (with a caveat, mentioned below), nor any sort of smart contract capability. It does have multiple, (largely) independently operated nodes that write transactions to the ledger. The nodes work together to agree on what transactions should be written and in what order. As well, client applications (agents) that need to write transactions to the ledger must prove they are authorized to not only write to the ledger but are authorized to write that specific transaction to the ledger. For example, an update to a DIDDoc (e.g. to change the public key in the DIDDoc) can only be written by the entity that can prove it is authorized to do so based on the information in the DIDDoc itself.

A caveat to the point about Indy not being about the exchange of assets:

Although assets are obviously not the purpose of an Indy ledger instance, there is a capability in Indy to support a pluggable implementation of a payment token. The token isn't used to enable general-purpose smart contract capabilities, or for storing or exchanging value such as on Bitcoin-like networks. Rather, it is to support payments for certain network operations—for example, writing to the ledger, issuing a credential or proving information about an entity. The token could be used to prevent denial-of-service (DoS) attacks (an attacker flooding the network with free ledger write operations, thus preventing legitimate operations) and might serve as a mechanism for funding an instance of the network. The term "pluggable" means that as transactions are processed on the ledger nodes, calls are made to an application programming interface (API) based on when events occur that might involve payments. Anyone operating an Indy network can write a library that implements what happens when those API calls are made. Indy's reference implementation is called "libnullpay," meaning that when the API is called it just returns with no action taken.

Types of Blockchain—What Type is Indy?

There are several different types of blockchain systems characterized along two dimensions: access and validation. You have likely heard of Bitcoin and Ethereum—they are public permissionless networks. That means that anyone can access them (public) and anyone can participate in the validation process (permissionless). Bitcoin mining is the validation process on that network and anyone can run a mining rig—no permission needed.

Similarly, the Hyperledger frameworks (Fabric, Sawtooth, Iroha and Burrow) are (primarily) used for private and permissioned networks, limited to who can access them (private) and who can participate in the validation process (permissioned). As shown in the diagram below, Indy falls between these two models.

Types of Blockchain

Indy is designed to be operated such that everyone can see the contents of the blockchain (public), but only pre-approved participants, known as **stewards**, are permitted to participate in the validation process (permissioned).

Note: Although Indy is designed to be run as a public network, an instance of Indy (or any other public blockchain) could be run as a private network accessible only to those using the network.

The ramification of Indy being designed to be public puts a significant constraint on what data can be put on an Indy blockchain. Specifically, *only public data can go on the blockchain—no other data, even if it is encrypted*. That last part about encrypted data might not be obvious. If the data is encrypted, it can only be accessed by those with

the decryption key, so shouldn't we be able to put any data safely on a public blockchain? Well, that might be true—today. However, Indy is being built to last from decades to generations. What happens if an encryption algorithm used today is broken sometime in the future? Since the data on the blockchain is open to everyone to see and is immutable, encrypted data that can be easily decrypted becomes public information. The Indy designers don't want that, hence the rule—no private data on the blockchain. Later in this chapter we'll cover "What Goes On The Ledger?"—a favorite question from everyone getting up to speed on Indy, Aries and Ursa. Did you catch that part about **NO PRIVATE DATA ON THE BLOCKCHAIN**? Good!


There is No Private Data on the Indy Blockchain!

There Is No Private Data on the Indy Blockchain!

How is Consensus Reached?

As with all blockchain implementations, Hyperledger Indy uses a consensus algorithm to decide the contents of the next block added to the chain. Specifically, Indy uses **Plenum**, an implementation of a Byzantine Fault Tolerance (BFT) algorithm. BFT algorithms are designed to achieve consensus when many of the nodes are not operable or accessible. Specifically, Indy's ledger can achieve consensus when only f (e.g 8) nodes of a 3f+1 (e.g. 25) node network are operational. Plenum performance degrades less than other BFT algorithms (on the order of 3% versus up to 78% for others) when faults occur in the network. As well, underlying Indy's consensus algorithm is a secure and robust messaging system amongst the nodes of the network, including a multi-signature scheme for returning ledger state information. This improves the overall security of Indy. The Plenum implementation has been found to be useful enough to be separated out from the core Indy code base into a separate repo that is used by Hyperledger Indy—and can be used by other Blockchain implementations.

Indy also implements a novel deployment of stewards—the nodes of the network that have permission to participate in the validation process. Since Indy is designed to be a global public network, an instance will have many available nodes located around the world—more nodes available than can be effectively used in the validation process. The validation process needs enough stewards to be robust in the face of faults, but not too many as to degrade the performance in reaching consensus. Interestingly, those testing the performance of Indy have found that sweet spot is 25 nodes—robust, able to survive the failure of eight nodes, but fast enough to support the expected number of write transactions on the network—on the order of 100s of transactions per second.

What if you have more stewards available? Indy's planned solution is to have an optimal subset of stewards as validators nodes actively participating in the Plenum consensus algorithm, and the rest as observer nodes, tracking the growing blockchain, serving reads (thus offloading that work from the validators) and ready to be called on to be validators should they be required. The image below from Sovrin (covered on the next page) shows the division of validator nodes (those currently participating in writing to the ledger) and the observer nodes (read-only nodes, but ready to become validators if needed).

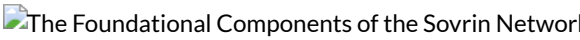

The Division of Validator Versus Observer Nodes
The Division of Validator Versus Observer Nodes

The Sovrin Foundation

At this point in the course, you've heard the name Sovrin a few times. The Sovrin Foundation is a global non-profit that has organized the deployment of Hyperledger Indy code on a (growing) number of nodes to create a running public permissioned Indy blockchain instance. To understand the goal of the Sovrin Foundation, think of the Sovrin instance of Indy as the identity equivalent of the Domain Name System (DNS), which has enabled the global routing of data on the Internet since 1985. The production Sovrin network has been running since July 31, 2017, its launch immortalized in this blog post, "*Launching the Sovrin Network*," from Sovrin Chairman Phil Windley.

The Sovrin Foundation provides the three foundational components of the Sovrin Network in the form of a BLT (not the sandwich):

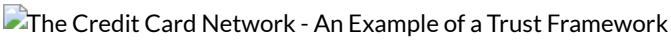
- **B**usiness – the Sovrin governance framework.
- **L**egal – A series of legal agreements signed by the Sovrin Network participants.
- **T**echnical – The underlying software of the Sovrin Network, Hyperledger Indy.


The Foundational Components of the Sovrin Network

All three parts are necessary to enable a global, trusted system for identity. The need for legal agreements and a sound technical foundation are fairly obvious, but what is a governance framework? You may recall we mentioned governance frameworks when we talked about trust over IP (ToIP) in Chapter 2.

The Sovrin Governance Framework

Governance frameworks (also called **trust frameworks**) are common in the digital identity world, but are also used in other contexts, sometimes using other names, such as "operating regulations" and "operating policies." In all cases, they define how to govern multi-party systems where participants desire the ability to engage in a common type of transaction with any of the other participants, and to do so in a consistent and predictable manner. That's a bit of a mouthful, so here's an example. Credit Card networks are operated based on a trust framework. The participants, credit card holders, merchants, clearing houses, banks and credit card companies all operate independently, but according to the rules that have been defined. As a result, the merchant can safely accept a credit card for payment, confident that the other parties will make sure that the merchant receives in their bank account the funds for the payment. There is not one huge piece of software written by one company that makes all that work. Rather, each participant knows there is a set of rules—the trust framework—to follow and they write code (and test and verify) for their part of the overall process.


The Credit Card Network – An Example of a Trust Framework

A governance framework enables an organization to count on the business and/or technical processes carried out by another organization. In many cases, trust frameworks have been able to work and most importantly, scale. Common examples include credit card systems, electronic payment systems and the Internet domain name registration system, which all rely on a set of interdependent specifications, rules and agreements.

An identity governance framework defines the rules for interactions between organizations for handling identity, authentication (who you are), and authorization (what you are allowed to do). The Sovrin governance framework defines those rules for the organizations using the Sovrin Network. Within Sovrin, for example, there are governance frameworks for how the stewards operate, as well as one for how each network participant must operate—endorsers, transaction authors, readers and so on. Sovrin has also defined a generic governance framework that can be the basis for a domain-specific governance framework. For example, banks and credit unions might use the Sovrin governance framework as the basis for defining a governance framework for what credential schemas to use and what processes an issuer must carry out before issuing a credential. Recall what we said in Chapter 3 about the need for a verifier to trust the issuer. It's through a governance framework that a verifier can know about the issuers processes prior to issuing a credential so they can decide if they trust the issuer.

Elements of the Sovrin Foundation

So what is the Sovrin Network? The Sovrin Network is a single, global instance of Hyperledger Indy. Each node is operated by a Sovrin steward, an organization (company, government, university, etc.) that has agreed to a legal agreement that defines how they will operate their node (minimum hardware, network access, monitoring, security, maintenance, etc.) within the rules defined in the Sovrin governance framework. As of August 2019, Sovrin stewards number more than 70 and include banks/credit unions, universities, law firms, and technology companies around the world (see the following image). The Sovrin Foundation, through its governance frameworks, provides governance for the network, the use of nodes and coordinating the software upgrades (including adding new features) to the nodes. The Foundation includes a Board of Trustees to oversee the business and legal aspects of the network and a Technical Governance Board to oversee the technical aspects.


Sovrin Steward Locations (circa March 2019)
Sovrin Steward Locations (circa March 2019)

And of course, the Sovrin Network has users. Some users, called "endorsers" (initially, the trustees and stewards) are entrusted through reputation, legal agreements and their adherence to the Sovrin governance framework to write data to the public ledger for themselves and others. The remainder of the participants in the system (identities) use the steward-operated nodes to read and (via endorsers) to write to the Sovrin public ledger. A key capability of endorsers is that they can "anoint" other identities known to them to be endorsers, provided those identities agree to the Sovrin legal agreement and the governance framework. It is through this "web of trust" that the capacity of the network scales.

With a global network in place, Sovrin can be used to solve the identity on the Internet problem. Users can create and write decentralized identifiers (DIDs) and verifiable credential metadata to the Sovrin Ledger, and use that data to issue verifiable credentials to holders. And, as we know from Chapter 3, holders can prove claims from those credentials to verifiers to enable trusted digital transactions without having to ask the issuer about the holder.

Transaction Author Agreement

An example of the Sovrin governance framework in action is the **transaction author agreement** (TAA). Think of it as equivalent to an end-user license agreement (EULA) that every software package and website makes you acknowledge (and that no one ever reads). Unlike blockchain implementations such as Bitcoin, where anyone can operate nodes and write transactions, with Sovrin, a group of permissioned, known organizations (stewards) operate the nodes and write data to the blockchain. Further, those organizations have lawyers that want to mitigate the risk that their organization will be liable if someone submits a transaction that puts illegal information on the blockchain (such as in this example: illegal prime). As such, with every transaction, the writer (called the transaction author) must acknowledge that they agree to the TAA. The TAA itself is not in the transaction, but the location of the TAA they acknowledged, and the hash of the agreement are in the transaction, along with a code indicating how they reviewed the TAA. You see why it's like the EULA—an agreement that few (no one?) will read, but is written by lawyers, and that they think it will protect their organization. That's a lot different from Bitcoin!

Note: Like the pluggable payment interface, Hyperledger Indy enables the TAA interface, but does not require an instance of Indy to use one. Each Indy blockchain instance is configurable and one set of configuration parameters covers the TAA, if it is required, where the text of the current TAA resides, the hash, and the enumeration of the ways that the TAA can be acknowledged. For the Sovrin instance of Indy, the TAA is enabled, and the TAA used is one agreed upon by the Foundation and its steward members.

Sovrin Payments

At the time of writing this course (September/October 2019), the Sovrin Foundation charges for transactions on the ledger, and plans on activating an implementation of the Indy payment API with a token. The charges are modest, likely necessary as a mechanism to prevent abuse of the network and will be used to fund the operation of the network. Since only issuers of credentials need to write to the ledger, they will bear the brunt of the costs. Later in this chapter, we'll go into what transactions are necessary to issue credentials and we can calculate the costs that issuers will face.

What Goes on the Blockchain

As we have mentioned, there is no private data on the blockchain. Such data is the verifiable credentials that go directly to the person (or enterprise) and into their personal storage—an encrypted wallet. There is absolutely nothing about any individual credential on the blockchain.

When you go to present claims for a credential to somebody for proof, the verifier doesn't go to the issuer to ask, "Hey, can you confirm this information?" Instead, they go to the blockchain, independent of the issuer, and use the blockchain to get the material to verify it. So that's where the blockchain comes in. To put it in simple terms, when you go into a bar and have your driver's license validated, you don't want the bar to call the government and say,

"Hey, they gave me this ID. Can you confirm they are over 18?" In the paper credential world it totally makes sense. Well, the blockchain enables exactly the same thing for verifiable credentials.

So, what does go on the blockchain? Let's go through the two, three or four pieces of data that an issuer puts on the ledger. As well, you can use these instructions (also used in Chapter 3) to look at what the different transaction types look like on an Indy network.


What Goes on the Blockchain

What Goes on the Blockchain (Cont.)

Public DIDs

When the issuer wants to issue a credential, they **MUST** have a DID on the blockchain that allows verifiers to find out who they are, and in the process, why the credentials they issue can be trusted.

Schemas

Next, the issuer **MAY** put a schema on the blockchain that says in effect, "When I issue a credential, it's going to have these fields in it." An issuer might put their own schema on the ledger, or they can use one that someone else put on the ledger. In the current iteration of Indy, the schema is just a list of attribute names. As this course is being written, work is ongoing in the Indy project to add what is known as "rich schema." Once that is in place, a schema will include additional information about the attributes, including a data type and how the information is encoded in the credential.

Credential Definitions

Before an issuer can issue a credential using a schema, they **MUST** put a credential definition on the blockchain. A credential definition says, "I'm an issuer using this DID, I'm going to use that schema for my credentials, and here are the public keys (one per claim) that I'm going to use to sign the claims when I issue a credential." In all, an issue must have a DID, a schema and a credential definition on the blockchain before issuing credentials.

Revocation Registry

The issuer **MAY** want to be able to revoke credentials and if they do, they must also write a revocation registry to the ledger before issuing credentials. The revocation registry links back to the credential definition, and allows the issuer to unilaterally revoke credentials, independent of the holders. No one can look at the revocation registry to see if a specific credential has been revoked. But, with the magic of a zero-knowledge proof (ZKP), a holder can prove (and a verifier can verify) that the holder's credential has not been revoked—provided of course it has not been revoked. This action is taken by the prover when creating a presentation.

While we won't go into the math behind the revocation scheme, here are the mechanics of using Indy/Aries revocation:

- Each revocable credential that is issued has a unique identifier known only to the issuer and holder. Further, there is a number stored on the ledger (called an accumulator) that is updated every time the issuer revokes one or more credentials. If the holder's credential has not been revoked, the holder can calculate and put in the proof a pair of numbers that the verifier can combine with the accumulator (from the ledger) to get a known result proving non-revocation.
- If the holder's credential has been revoked, the holder is unable to generate the numbers necessary to be combined with the accumulator to get the known result. In other words, once the credential has been revoked, the holder cannot prove non-revocation. Yes, that is a double-negative, but that is what is happening.

Through the process, the unique identifier of the credential is not exposed to the verifier. To check again for revocation, the verifier must ask the holder for another proof, which may at first glance seem a little odd. Here's why Indy revocation works that way:

- The verifier is not given an ID for the credential so the verifier has no way to check that a credential referenced in a proof they received earlier hasn't subsequently been revoked.
- The verifier is not given a unique credential ID because it would provide the verifier with a correlatable identifier for the prover—exactly what the ZKP properties of Indy verifiable credentials are designed to prevent. Such an identifier isn't exposed in the presentation of claims, and it is also not exposed in proving non-revocation.

The expectation is that for the most part, issuers will revoke credentials on a periodic basis, such as once an hour, or once a day. The issuer will monitor the revocations through the period, track the IDs of the credentials to be revoked, calculate an update to the accumulator to revoke those credentials and write the new accumulator value to the ledger. If the business needs require more immediate notifications of revocations, the issuer can update the accumulator as each credential is revoked. On the Sovrin Network, there is a cost to write updates to a revocation registry, and that will no doubt play into the business requirements of when revocations occur.

Recap

So that's all that goes on a ledger. An issuer can issue a million (or a billion!) credentials, but none of them go on the blockchain. All of them refer back to those few other pieces of data that did go on the blockchain and they are:

- DID (required)
- schema (required, but could be written by the issuer or someone else).
- credential definition (required)
- revocation registry (optional, used only if it is necessary to be able to revoke credentials)

So, for all the issued credentials of one type from one issuer, there are only two, three or four blockchain transactions executed.

Costs of Writing

As we mentioned earlier, there is a cost to writing to the Sovrin Ledger. Here is a back-of-the-envelope calculatio of what an issuer will pay (as of September 2019) for issuing credentials on the Sovrin Ledger. For this, we'll assume the following transactions:

- a DID for the issuer – $10US
- a schema (the issuer creates their own) – $50US
- a credential definition – $25US
- a revocation registry – $20US

There is an ongoing cost for revoking credentials and updating the issuer's DID. We'll assume the following annual costs:

- one DID update – $10US
- a revocation registry update each business day – 200x$0.10US
- one every second business day real time credential revocation – 100x$0.10US

That's a total of $105US for an issuer to begin issuing credentials, plus an ongoing annual cost of $40US for DID updates and revocations.

Other potential transactions include issuing more credential types (no new DID needed, possibly a schema, a credential definition and likely a revocation registry), additional revocation registries (if the issuer runs out of credential IDs to be issued for a given credential type), and updates to multiple revocation registries. And of course, the rate of revocations might be different than assumed above.

Note that if your application is not using the Sovrin Ledger but rather some other instance of an Indy network, there may or may not be a transaction cost, and if there is a cost, it may be quite different from that of the Sovrin Network.

Summary

In this chapter, you have learned that the Indy ledger is a public, permissioned ledger and that consensus on the ledger is arrived at using the Plenum algorithm. We talked about the Sovrin Foundation and how its single, global instance of Hyperledger Indy is like the identity equivalent of the Domain Name System (DNS). And, we looked at exactly what goes on the blockchain, reiterating (again and again) that no private data ever goes on chain. The main takeaway from this chapter is that blockchain is enabling us to move away from identity challenges we discussed in Chapter 1 and move towards a more trusted, self-sovereign identity Internet. In the next chapter, we will delve into Aries and how agents will enable secure, peer-to-peer communication and exchange of credentials.

Chapter 5: The All-Important Agent, Or Rather, Agents!

Chapter Overview

In the last chapter, we looked a lot at Hyperledger Indy and in particular, what happens on the ledger. This chapter delves into Aries—specifically, Aries agents, how your digital wallet will work and how keys and the crypto behind it all keeps your data your data!

Remember, we want to move away from centralized identifiers, unwanted correlation of data and the issues that come with user IDs, passwords and personal data on servers across the Internet (Chapter 1). The Aries agents and agent protocols we are about to discuss will enable an Internet where the user is in charge and where they control their data! Also remember that the goal of Aries is to be blockchain agnostic so people can start building Aries agents regardless of the blockchain they want to use. Pretty cool stuff!

No More Mr. Login!
By Peter Stokyo

Learning Objectives

By the end of this chapter you should:

- Know that there are many types of agents, with different roles.
- Be familiar with the components of an agent, including the storage and ledger interfaces.
- Understand at a high level how agent messaging works and connections are established.

What Is an Agent?

Formally, an Aries **agent** is a piece of software that enables an entity (a person, organization or thing) to assume one or more of the roles within the verifiable credential model—an issuer, holder or verifier—and allows that entity to interact with others that also have verifiable credential roles. Agents may do (many!) other things, but it's their ability to handle verifiable credentials that is their defining characteristic.

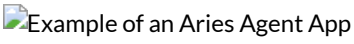
Agents use private, pairwise DIDs to secure their peer-to-peer communications. With a different pairwise DID for every relationship, each with keys and end points that you control, communication between agents is end-to-end encrypted, secure and safe. And with verifiable credentials and proofs easily exchanged between agents, you can be certain who is on the other end of the connection.

There are a number of categories of agents that can be created for different purposes, some obvious and others less so. Let's take a look.

Personal Agents

For a person, an agent will most commonly be a mobile app that feels much like the messaging apps you use today —WhatsApp, Messenger, etc. Your mobile agent will help you establish connections, manage the set of relationships you've created, and you will exchange messages with those connections. Some of those messages you receive will be offers of credentials and others will be requests for your to prove claims from one or more of

those credentials. You can (and should!) also initiate requests for proofs, so that you can verify attributes of those with whom you are messaging. And, like any messaging app, some of those messages can be, well, just text messages that tell you something or ask you something or provide you with a picture of a cat. Of course, with the goal of self-sovereign identity in mind, despite having the feel of a messaging app, the privacy and security offered by the Aries agents are the primary focus.

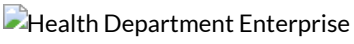

Example of an Aries Agent App
Example of an Aries Agent App
By Streetcred.id

But personal agents don't have to be mobile. They could be run on a laptop or desktop computer as applications. Agents directly controlled by individuals might also be operated by a service and run entirely (including all keys) in the cloud. That model, however, is at odds with our goal of controlling your own agent—and your data. If an agent is running in the cloud, someone else is operating the agent and you are not fully in control. At that point, it is about how much you trust the service.

Enterprise Agents

An enterprise, like a company or a department within a government, will have agents that run on servers to verify proofs of claims submitted by their clients and issue credentials to clients. For example, a health department, prior to issuing an operating permit, might use an Aries agent that messages with the Aries agent of a bakery owner to verify (using proofs) that:

- The bakery is appropriately registered as a company.
- Has proper insurance.
- Has received safe food handling certification.

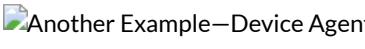

Health Department Enterprise Agent Could Verify Bakery's Proof of Claims
Main image by Peter Stokyo

Periodically, inspectors might visit the location, verify the operating permit using their agent, and after a manual inspection, trigger the health department's agent to issue inspection certificates (good and bad) as verifiable credentials to the owner's agent. In extreme cases, the health department's agent can revoke the operating permit.

As the saying goes, corporations are people too. Well, maybe not, but corporations will need to receive, hold and prove verifiable credentials about themselves. Just as people have birth certificates, companies have papers that document their registration with the jurisdiction in which they operate. As the example above shows, organizations receive various licenses and permits (as verifiable credentials) related to the business they operate and from time to time must present claims about themselves from those credentials. These identity enterprise agents are likely separate from the "issuer/verifier" type enterprise agents talked about above. Enterprise issuer/verifier agents are part of the business processes between the enterprise and its clients, while an enterprise's identity agent is responsible for managing and proving attributes about itself.

Device Agents

Internet of Things (IoT) devices might embed agents capable of issuing verifiable credentials based on data from the device's sensors. The devices themselves might be certified by a standards organization, and by immediately putting the sensor data into a verifiable credential, the data can be tied to the device and proven not to be tampered with as it is used. For example, sensors in your car can generate verifiable credentials to track mileage and maintenance on the vehicle and so that the data can be proven when the car is sold. Or, inspections stickers on gas pumps could be handled by the pump issuing verifiable credentials that could be monitored and only inspected when generating faulty data.


Another Example—Device Agent

Routing Agents

Another category of agents are routing agents. These are agents that serve as intermediaries to facilitate the flow of messages between other types of agents. And it turns out, there are lots of places we want or need to have such intermediaries. Because you cannot directly address a mobile agent across the Internet, a primary example of a routing agent is one that allows a mobile agent to send and receive messages. For example, an enterprise agent (or any other piece of software) can't send a message containing (for example) an offer of a credential directly to a mobile agent (or any other app) on Alice's phone. It's just not technically possible. Instead, a mobile agent must have something that holds its messages until that mobile agent asks for them.

We'll dive into this concept later in the chapter but for now it's enough to know that the idea of the routing agent is not to be a destination for a message, but instead just a facilitator to move messages along, kind of like being a part of the postal system. And, like the postal system, a routing agent should not know anything about the contents of a message, just where the message needs to go. As such, whenever a routing agent is involved in handling a message, the sender puts the message into an envelope so the agent can't see the message, just where it's supposed to go. As you might imagine, such handling is done with encryption.

Routing Agents Move Messages Along

Edge Agents and Cloud Agents

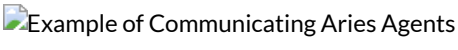
In early materials from Hyperledger Indy and Sovrin, only the terms "edge" and "cloud" were provided as categories of agents. Over time, these terms have continued to be applied, but they don't mesh well with the reality of the categories of agents we've just listed. This note is to clarify their use—and why they can be confusing.

An edge agent is one that is a destination for messages and is directly controlled by its owning entity. Obviously, mobile agents and IoT devices are edge agents. However, enterprise agents, as discussed above, are also edge agents. They are used to send and receive messages for their owning entity and to process verifiable credentials in a variety of ways.

It's with the term cloud agent that the early Indy/Sovrin definition gets a bit, well, cloudy (sorry about that…). Cloud agents in this context are the routing agents that we talked about above—agents that facilitate messaging, but that don't (usually) directly handle the processing of messages. The problem, of course, is that enterprise agents also run in the cloud, running on servers under the enterprise's control and secured using modern enterprise security and key management practices.

What is intended to be conveyed by the terms edge and cloud agents is a level of trust and corresponding security requirements. Edge agents are controlled directly by their owning entity, must be trusted by that entity and must be managed using appropriate security so as to prevent a loss of control and hence, trust. Cloud agents are operated by other than their owner and as such should be trusted less than an edge agent. For example, they can be trusted to facilitate messages that are encrypted such that they cannot read them. And, a cloud agent owner may trust them for some other purposes beyond just routing.

The picture below is an example of some communicating Aries agents. The edge agents are handled by the identity owners themselves—in this case, a consumer and an enterprise. The cloud agents facilitate the messaging by, for example, providing a permanent endpoint for a device that may or may not be online at the time messages are being received.



Example of Communicating Aries Agents

Categorizing Agents

Categorizing agents is tricky. In this section, we've given some reasonably clear definitions of agent categories, covering the common and expected cases. But agents are software and as such, can be extended and deployed in endless ways. For particular use cases, agent type functionality can be mixed and matched. So, while you can use these categories as guidelines, remember that they are not the only possibilities. The important thing to consider as you think about going beyond these categories is the risk that the owner may lose control of the agent, that others may be able to gain access to the messages, keys and data associated with the agent.

Architecture

All Aries agents, regardless of the category, have a similar architecture. In this section, we'll review the major components of Aries agents and talk about how the implementations of these components can be used to give an agent its personality. We'll reference the picture below in summarizing the components.


Agent Components

Key Management Service (KMS)

An Aries agent (usually) has secure storage (called the **key management service (KMS)**) that is used for all of the information collected by the agent. At a low level, an Aries KMS is wrapper code around a standard database for storing DIDs, keys, connections, credentials and any other information the agent tracks. KMS implementations are *pluggable* (new implementations can be independently created and used) and two reference implementations have been created for a range of use cases. A **SQLite** implementation can be used for small-scale agents, and a **PostgreSQL** implementation can be used for enterprise scalable deployments (**SQLite** might be used on a mobile phone to store thousands of records, **Postgres** on a server to store millions (or billions) or records). In either case, (almost) everything in the database is encrypted before it is stored.

There are two caveats in the paragraph above:

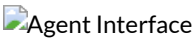
- *A static agent is a special type of agent that does not have a wallet. Instead, a static agent is configured at deployment time with the keys and connections necessary to do its work. In memory and CPU limited devices, a static agent can be used if there are not enough resources for a wallet. No doubt other scenarios will arise where a static agent will be useful.*
- *The only unencrypted data that goes into a wallet are optional tags that might need to be queried in other-than-equality expressions. It's pretty obscure, but if your app needs to search the wallet for a record where the stored data is (for example) greater than a given value, you have to store the to-be-queried data in plaintext. Yes, that's very obscure.*

As mentioned, KMS storage data is encrypted. Ideally, the management of the handful of keys for the encryption of the KMS data is handled by special hardware on the device (mobile phone, enterprise server, etc.) on which the agent runs, such as the biometric services (fingerprint, face ID) available on modern smartphones. Further, anything requiring processing using the many (many) keys stored in the KMS, including creating, storing, querying, retrieving, encrypting, decrypting, signing and verifying data, is only possible through calls to the KMS code. This layering is to prevent private keys from being accessed by generic agent code written by an arbitrary developer. Keeping keys secured within the open source KMS reduces the risk of security vulnerabilities in "user" code resulting in keys being exposed.

Agent Messaging Interface

The **agent messaging interface** enables an agent to establish and manage connections with other agents and send and receive messages to those agents. The mechanisms an agent supports to transmit and receive messages depends on where that agent is deployed. Aries uses DIDComm (DID Communication) messaging protocols and is designed to be "transport-agnostic" meaning that any communication method can be used to send messages. Many agents use HTTP (or HTTPS) to send and receive messages, but any mechanism can be used for a given

agent—web sockets, SMTP (email), XMPP, even hand-written notes (really!) can be used for transporting messages.

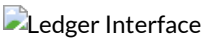


Although some of those transport methods might support encryption, Aries messaging does not rely on that, so even plaintext transport methods can be used. This is possible because the entire Aries message is encrypted (using Ursa), assembled into a standard structure (using JSON Web Encryption (JWE)), converted into a string of characters (usually using base64 encoding) and then sent as the payload using the selected transport mechanism. Anyone looking at the message sees only a really long string of characters. The intended recipient reverses the process to get back the delivered Aries message. The basic structure of Aries messages and the encoding/decoding of the messages during delivery is defined by the **DIDComm envelope protocol**, something we'll discuss later in this chapter.

In addition to sending and receiving messages, the agent interface manages the relationships the agent has with other agents and the conversations that the agent has with other agents. As we'll learn, many conversations between agents require a sequence of messages and the agent interface tracks the state of those sequences (threads).

Ledger Interface

The **ledger interface** enables all of the blockchain operations that we talked about in the previous chapter about Indy—reading and writing DIDs and other transactions to/from the ledger. The difference in Aries versus Indy is that the ledger interface is pluggable, allowing an Aries agent to interact with multiple ledger and verifiable credential eco-systems. That said, at the time of writing this course (September/October 2019), Indy remains the only ledger interface implementation in Aries.



An Aries ledger interface will likely to be broken into three parts:

- DID resolver
- Writing mechanism
- Verifiable credential handler

DID Resolver

The simplest part of the ledger interface is the DID resolver for reading DIDs, which is architecturally similar for every DID method—given a DID, resolve it to return the DIDDoc. Ideally, the work to resolve any DID can be done by code within an Aries agent by deploying a pluggable resolver for each DID method. As of the writing of this course, there were just two DID methods supported in Aries agents—public DIDs on the Sovrin network and private, pairwise DIDs.

A current, and possibly future, problem with the plan of supporting all DID methods is that there are lots and lots of DID methods—too many to support in every agent. The plan to solve that in Aries is to use a centralized **universal DID resolver** service. If a DID is encountered that cannot be resolved using local code, it is sent to the universal resolver, which will (try to) support all DID methods and it will complete the resolution and return the result.

Writing Mechanism

The second part of the ledger interface is the mechanism to write data to the ledger. This must be implemented within the Aries agent to ensure full control over the private keys involved in the transactions. Further, different ledger implementations will involve writing different transactions to the ledger, resulting in both the code and interface being different per ledger implementation. As with the resolver part, an Aries agent implementation will, out of necessity, only support writing for a small number of DID methods, often just one.

Verifiable Credential Handler

The third part of the ledger interface is the verifiable credential handler. In this case the interface is likely to be pretty much the same across implementations, with essentially the same sequence of events occurring for issuing credentials and presenting proofs. That said, the specific actions within the steps will likely be different from one verifiable credential implementation to another. The actions to process an Indy zero-knowledge proof are quite different from the presentation of a credential-based proof.

Controller

he final component of an Aries agent is the **controller**—the element that gives the agent its personality. The controller provides the "business rules" (for lack of a better term) that define what actions the agent will initiate and how the agent will respond to events. For a mobile agent, the controller is a mobile user interface that presents the options to a person who then supplies the business rules. For a verifiable credential issuer/verifier-style enterprise agent within an organization, the controller might be a legacy database system that manages customer data. Such an agent might enable customers to submit data by presenting claims from verifiable credentials instead of typing the information into a web form. That is a win for the organization—better quality data that does not need to be manually reviewed and verified, and a win for the customer—less typing and faster responses because the data does not need to be manually verified.

A controller is the software you build around Aries. All the capabilities that make up Aries go into the agent, and then you, as an external party, write the code (and business rules) to create an agent-enabled application. We'll talk more about building such applications later in this chapter.

Peer-to-Peer Messaging

The key capability of an Aries agent is peer-to-peer messaging—connecting with agents and sending and receiving messages to and from those agents to accomplish some interaction. The interaction might be as simple as sending a plaintext message, or more complex, such as negotiating the issuing of a credential or the presentation of a proof.

Enabling both the exchange of messages and the use of messaging to accomplish higher level transactions requires participants to interact in pre-defined ways, to interact by following mutually agreed upon protocols. Messaging protocols are just like human protocols, a sequence of events that are known by the participants to accomplish some shared outcome. For example, going out to a restaurant for a meal is a protocol, with both the guests and the restaurant staff knowing the sequence of steps for their roles—greeting, being seated, looking at menus, ordering food, etc. Unlike human protocols (etiquette), messaging protocols need to be carefully specified and then implemented by multiple participants.

With Aries, there are two levels of messaging protocols. Since all of the messaging is based on the exchange and use of DIDs, the messaging mechanism is called **DIDComm** for **DID communication**.

DIDComm: The Aries Messaging Protocols

At the lower level is the **DIDComm envelope protocol**, the method by which messages are exchanged, irrespective of the message content. You can think of the envelope protocol as being like the postal service. You write a letter, place it into an envelope, address it to a recipient, give it to the postal service and magically, it appears at the recipient's door. And, just like the postal service, there is a lot of complexity in the sending and receiving of a message.

At a higher level are the **DIDComm content protocols**, hundreds of protocols that define back-and-forth sequences of specific messages to accomplish some shared goal. The simplest of these is one agent sending another agent a text message ("Hi, how are you?"). A much more complex protocol is issuing an Indy-style credential, where it takes at least three messages back and forth to offer, request and issue a credential. The set of messages, the roles of the participants, plus some extra details such as acknowledgments and error handling, define each of the many DIDComm content protocols.

With the DIDComm envelope protocol we don't care about the content of the message, just the method by which a message gets delivered. With the DIDComm content protocols it's the reverse—we know that the messages get delivered (we don't care how), we only care about what to do with the content of each message.

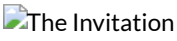
Aries Messaging: Walkthrough

In the following, we present a brief walkthrough of messaging between two Aries agents—establishing a connection and then using that connection for an issuer agent to issue a verifiable credential to a holder.

We start with two agents that are deployed, unknown to one another and that for some reason want to connect. For example, it might be a user trying to connect to an enterprise, or two users with agents trying to connect to one another.

Establishing a Connection: The Invitation

Since the agents don't have any sort of trusted communication path at the start of the protocol, they have to use some sort of out-of-band (non-DIDComm) mechanism to begin communicating. A common way to convey the invitation is that one of the agents generates an invitation and displays it as a QR code that a mobile phone containing the other agent can scan, convert into a string of data and process. The invitation can be sent in other ways as well: emailed, displayed on a screen and copied/pasted, written on paper, etc. Basically, any method of sending text from one agent to another.


The Invitation
The Invitation

The invitation is a chunk of plaintext (not encrypted) data (in JSON) that contains a unique identifier and the same sort of information that is in a DIDDoc—notably a public key and a service endpoint. It is the way an inviter says to the invitee "Hey, connect with me!" The invitation is in plaintext because the inviter has no public key for the invitee with which to encrypt the message.

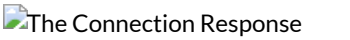
Establishing a Connection: The Connection Request

With the invitation in hand, the invitee can use the public key and service endpoint to get a secure message to the inviter. If the invitee wants to accept the invitation, it creates a new DID and DIDDoc for the relationship that is being formed and places the invitation identifier and the new DIDDoc into a connection-request content protocol message. It also creates a connection record and saves it in its own DID and DIDDoc. The DIDcomm envelope protocol is used to package the content protocol message, encrypting it using the public key from the invitation and sending it to the service endpoint in the invitation.


The Connection Request
The Connection Request

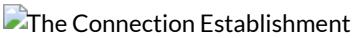
Establishing a Connection: The Connection Response

The inviter listening for messages on the service endpoint receives the message from the inviter and decrypts it to find the connection-request message. The invitation identifier in the message informs the inviter with what invitation to associate this message. It too creates a connection record, saving it in the DID and DIDDoc from the invitee. It then creates a DID and DIDDoc, saves it in the connection record, and packages the data into a connection-response content protocol message. That message is packaged (using the DIDComm envelope protocol) and sent to the invitee.


The Connection Response
The Connection Response

Establishing a Connection: The Connection Establishment

When the invitee receives the message, it saves the other agent's DID and DIDDoc in the connection record and the connection is established. The two agents are connected with a secure, private, end-to-end encrypted messaging channel. That's huge! Of course, the parties still don't know much about each other (just each other's DIDs), but that can come after—by exchanging proofs.


The Connection Establishment
The Connection Establishment

We have skipped over a lot of the details that make this protocol reliable, reusable and secure. At this level, however, it not only shows how connections are established but also provides a good demonstration of content protocols, showing how a protocol:

- Has participants with different roles (e.g. inviter, invitee).
- Uses a series of message types (e.g. invitation, request, response).
- Uses expected data elements in each message type (e.g. ID, DID, DIDdoc).
- Sends a threaded sequence of messages that trigger events to move the protocol through a series of states (e.g. invited, requested, established).
- Sends messages asynchronously. That is, the message is sent and the sender does not wait for a response. Response are sent as new messages. This is different from the HTTP protocol, which is a synchronous (request-response) protocol.

This encompasses a protocol called the **DID exchange** protocol which is documented in the Aries Request for Change (RFC) repository. Because of the foundational nature of this protocol, the details we glossed over make it one of the most complex of the Aries protocols.

The Issue Credential Protocol

To solidify the concepts, let's take a quick look at another content protocol to show how an established messaging channel between agents is used. The **issue credential protocol** is used to enable an issuer to provide a holder with a verifiable credential. Again, we'll gloss over some features of the protocol to keep it simple.

In this protocol:

- There are two participants (issuer, holder).
- There are four message types (propose, offer, request and issue).
- There are four states (proposed, offered, requested and issued).

At the start, the two participants have agents and have established a connection. To issue an Indy credential, the simplest instance of the protocol must have three steps:

- The issuer sends the holder an offer message.
- The holder responds with a request message.
- The issuer completes the exchange by sending the holder an issue message containing the verifiable credential.

Since we want the protocol to support other verifiable credential implementations beyond Indy, that sequence is not set in stone. Some verifiable credential implementations don't have a technical requirement for an offer, so the protocol could be completed in just two steps, a request from the holder and an issue from the issuer.

The propose message is optional and can be used for two purposes. First, a holder can use a propose message to start the protocol, rather than waiting for an offer message from the issuer. As well, the propose message can be used as a way to negotiate with the issuer about an offer. If the issuer sends an offer for one type of credential,

the holder can respond back with a propose message that says, "Hey, thanks, but I would rather have this other type of credential."

And of course, things can go wrong. In executing any protocol, things can go wrong. A problem report message can be used at anytime to notify the other side of a problem in the execution of the protocol.

Lab Time

Let's take a look at the envelope and content protocols in action by stepping through this short "*Aries Connections and Messages*" lab. DIDComm-based development will be a major topic of the developer-focused follow-on course to this one, "*LFS173x: Becoming an Aries Developer*" (coming early 2020!).

Other Protocols

The Hyperledger aries-rfcs code repository contains a growing list of documents about how DIDComm protocols work (the concepts folder) and about specific DIDComm content protocols (the features folder). The repository is a bit overwhelming to those new to Aries. There are many, many protocols in various states of endorsement in the community, from proposed (someone's great/crazy idea—who knows?), to demonstrated (someone implemented it), accepted (several groups have implemented it), adopted (everyone must use it) and even retired (that's so yesterday). Where to start?


Aries RFC Lifecycle
Aries RFC Lifecycle

If you are interested in looking over the protocols, we recommend looking at what protocols have been built into Hyperledger Aries agent implementations. For example, this document, "*Aries RFCs Supported in aries-cloudagent-python*," part of Aries Cloud Agent Python, contains a curated list of the RFCs that are currently implemented in that agent.

Building an Aries Agent Application

To wrap up this chapter, let's talk about how one goes about building an Aries agent application. How do you create something on top of all the Indy, Aries and Ursa goodness? As we talked about earlier in this chapter, that requires building an agent controller application that integrates with Aries agent functionality. To date, there are two approaches for integrating a controller with agent capabilities, the agent framework model and the cloud agent model.

Agent Framework Model

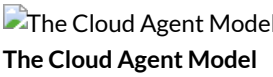
An agent framework allows you to build your controller as an application with the Aries agent embedded in your controller as a library. The controller uses the library to drive the agent according to the needs of your application. With a framework, you are constrained to the languages that you can use with that framework—this is because the language you use must support embedding the agent framework. As of this writing (September/October 2019), there is a complete framework based on .NET (**aries-framework-dotnet**) and one well into development based on golang (**aries-framework-go**). As well, there are groups looking at creating Java- and JavaScript-based frameworks.


The Agent Framework Model
The Agent Framework Model

Cloud Agent Model

The cloud agent model, as embodied by **aries-cloudagent-python**, separates the controller and agent into processes communicating using an HTTP API. The controller receives webhook notifications from the agent as events occur (e.g. messages are received from other agents) and uses an HTTP API exposed by the agent to respond to those events or to initiate new actions. With this approach, the controller is similar to any modern web app, receiving requests and responding over HTTP to an agent service. Although the only cloud agent is written in

Python, the controller is by design completely independent of the agent and thus can be written in any language. A controller could even be an existing enterprise application, extended to use verifiable credentials by adding API calls to an Aries agent process.



The Cloud Agent Model

Trade Offs

What are the trade offs? If you are building a non-mobile agent and you want an easy way to integrate the agent with your existing systems, regardless of the language, the cloud agent model is a great way to go. If you are building a brand new app that will operate standalone, either model will work, and you will want to think about what fits best into your technical environment.

If you are building a mobile agent, you will need to use an Aries framework, with .NET and Xamarin the current best option. Since Python is not readily supported on mobile platforms, the cloud agent model is currently not an option for mobile implementations.

What to do? A good consultant's favorite answer applies here: "It depends…" If you are planning on building an Aries application, you'll definitely be interested in the follow on course, "*LSF173x: Becoming an Aries Developer*" (coming early 2020)!

Summary

There is a lot to get to know about Aries agents, their components and how they work. This is the area in the three projects where most of the activity is currently happening. That includes within Aries, the definition of Aries protocols (through the RFC process) and on top of Aries, with organizations building solutions using Aries. While Indy and Ursa are cool, Aries is where development can and should have the most impact and we encourage anyone interested in making a difference in this environment to dig deeper into the Aries agent world.

Chapter 6: When Things Go Wrong

Chapter Overview

What happens if you lose your wallet? Well, we all know what happens in the paper credential world. You run around like a chicken with its head cut off, frantically searching for it, worrying that it may have fallen into the wrong hands. Will your identity be stolen? How long will it take you to get a replacement credit card? How do you even get a new driver's license? Where is it!? Such a pain!

That same scenario can also happen in the digital world we're talking about. What happens if you drop your phone in a lake? What happens if you lose the device that holds your digital wallet with your credentials and private keys? Can others use them? How will you get your data back? Such a pain! You have probably heard stories of cryptocurrency keys being lost and millions becoming completely inaccessible. That's way beyond a pain!

This chapter explores these challenges that are a reality even in a self-sovereign identity, Indy, Aries and Ursa world. The industry term for the set of practices around securely managing your SSI-related data is **decentralized key management system** (DKMS). We'll look at DKMS and related topics in this chapter. As you'll see, the full set of approaches are not yet fully defined, and there are few implementations. Lots of thought has gone into approaches, but not yet enough action. These are areas that are ripe for innovation and implementation.

Learning Objectives

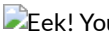
By the end of this chapter you should:

- Understand how agents can be backed up and restored.
- Be familiar with the the idea of a recovery key and how it might work.
- Be able to recall techniques that are being considered for protecting your recovery key.

- Be aware of the steps that you can take to protect your data if someone were to get a hold of your Aries agent.

Mobile Agent Wallet

The first and most likely problem with having your own digital wallet is losing it. One minute your phone is working fine and the next, you drop it into the Grand Canyon and it is destroyed in a 1000m drop to the rock floor. Or, if you aren't in Arizona, an even more likely (and much less dramatic) scenario is that the agent mobile app you are using has a bug, corrupts the data in your Aries agent key management service (KMS), you need to reset the app and all the data is lost. Now what?


Eek! Your Wallet Stops Working
Eek! Your Wallet Stops Working

Even for day-to-day use, backup and recovery capabilities are crucial. For consumer mobile agents, these functions must be automatic, with little effort to set up, no effort to carry out on an ongoing basis, and minimal effort to restore. As well, the backups must be safe. It must be extremely (!!!) difficult for an unauthorized user to access the data in an Aries app backup.

For enterprise agents, the situation is a little easier. Secure database backup and restore should be straightforward in a modern enterprise Information Technology (IT) group since an agent's database is more or less "just another database" from an enterprise perspective. In this section, we'll assume a focus on the use case of restoring a mobile agent KMS, but the same principles apply in the non-mobile agent scenario.

The backup and restore solution falls into two parts. First, the backups must be protected through encryption, redundantly stored, and periodically tested. Second, the private key needed to decrypt the backups must be protected separately from the backups themselves. Only when an authorized person is deliberately restoring the backups should the two come together. We'll look at some novel ways of managing the private key to keep it away from the backups until it is needed.

If we can make sure that backup and recovery is going to be successful, the chance of permanently losing one's wallet will be greatly diminished.

At the time of writing this course (September/October 2019), we are not aware of an implementation of a mobile agent with secure backup and restore capabilities. As such, the following is a high level summary of the expected approach. The approach may change and the details will change as implementations proceed.

Key Management Service (KMS) Keys

In normal operations, there is one key that is used to access all of the data in the KMS. Recall from Chapter 5 that (almost) all of the data in the KMS is encrypted and it is with what we will call the private KMS access key that the data can be decrypted. The KMS access key is protected by the capabilities offered by the platform on which the agent is running. For example, on a smartphone, the KMS access key is protected by the phone's hardware and accessible only using the security features of the operating system, such as face or fingerprint recognition. The single KMS access key is in turn used to decrypt a set of additional key pairs that encrypt/decrypt the data in the database. Those key pairs are only accessible inside the KMS.

We expect that the backing up of the KMS data store will be done using the tools provided by the manufacturer of the database tools. So, if an agent's KMS is using an SQLite database, the backup of the database will be created by SQLite backup tools.

In addition to backing up the database, the keys necessary to decrypt the data in the database must also be backed up and those must be protected. Those keys will be encrypted into a package protected by a second crucial key pair, the recovery key pair. It's the recovery private key that must be separated from the backup itself until it is needed together with the encrypted backups to restore the database. We'll talk about managing the recovery key in the next section. For now, we'll just focus on managing the backups.

So now we have two keys that can be used to access the data in the KMS. The one called the KMS access key is protected by the platform, ideally by hardware that prevents it being accessed by other than the owner.

Note: There could be with exceptions. For example, a court order/other extraordinary measure that forced access through a manufacturer's "back door" if one exists. See for example, this story from the Washington Post: "Apple Vows to Resist FBI Demand to Crack iPhone Linked to San Bernardino Attacks."

The second, called the recovery key, has a public key that encrypts the package of key pairs and the database backups, and its corresponding private key that can decrypt the KMS keys and backups.

Backups versus Exports

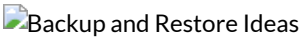
The current Indy secure storage mechanism supports creating an encrypted export of the agent's KMS data. Although that mechanism could be used for backup and restore, we don't expect that is how it will be used. That feature will be used for portability, when you want to move your data from one agent to another, from one vendor to another. While it could be used for backup and restore, we expect that the database tools mentioned previously will be far more effective in managing data.

Managing Database Backups

There are several places where a wallet backup could be stored:

1. On the device itself.
2. On the cloud agent.
3. On another device.
4. On peer-to-peer files.

Let's discuss each idea and how it might work.



Backup and Restore Ideas

Managing Database Backups: On the Device Itself

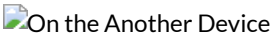
The easiest is storing the encrypted backup on the device itself, so if only the agent storage were to be deleted, a restore could be carried out immediately, on the device. Periodically, the mobile agent would initiate an incremental backup of the KMS database. We'll cover another reason for keeping a local backup in the next section.

Managing Database Backups: On the Cloud Agent

Since all mobile agents must have an associated cloud agent (see the "Routing Agents" page of Chapter 5), another obvious place to store a mobile agent backup would be with its cloud agent. Periodically, the mobile agent would initiate a backup of the wallet, pushing the encrypted data to the cloud agent for storage. A potential risk with that solution is the operator of cloud agent service has access to all of your data, although in an encrypted form. If they are also the maker of your smartphone app, they could in theory access the recovery key. If that is a concern, a separate service that provides backup storage might be an option—perhaps banks or other trusted entities might offer such services.

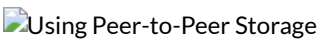
Managing Database Backups: On the Another Device

Another backup location might be on other devices that either you control or are controlled by a person that you trust. When one device is lost, another can be used to restore it. Of course, if you don't have access to another device, or the device is not available when you need it, that's a problem. Another option would be to push the backup to hardware on a device you own, such as a non-mobile device in your home. However, if a disaster occurs such as a fire, and both devices are lost…



Managing Database Backups: Using Peer-to-Peer Storage

Yet another option is to combine all of the above background targets using peer-to-peer file sharing techniques. Parts of the backup could be redundantly distributed to different backup holders, and when needed, the components would be collected from the backups and then restored once together on the target device.



Managing Database Backups: Creating Backups

Each of the non-local mechanisms would be managed by DIDcomm content protocols such as we talked about in Chapter 5. Such protocols would include messages to initialize other agents as backup targets and periodically push backup data to the targets. Restoring a backup from other than the local device would take some additional effort as, with the original wallet lost, connections between the lost agent and the backup holder(s) would not be available. Some way of rebooting those connections and providing sufficient proof of who is asking for the backup would be needed before the holders would hand over the backup data they are holding.

Managing Database Backups: Testing Backups

Recall that we mentioned other uses for keeping a local backup on the device. In addition to being available in scenarios where just the wallet is lost, the backup could be used to verify restoration process. As anyone who has managed a backup and restore process knows, if you haven't tested restoring your backup, you don't know if you have a working process. The worst time to try your first restore is when you've lost the original and the backup is the only copy you have left! One solution is to test restoring using the recovery key with the local backup copy to ensure it can be used to produce a complete copy of your wallet. The local copy can also be used to periodically verify that the remote copies can be collected and assembled into a restorable local backup.

Managing the Recovery Key

The second part of backing up and restoring an agent's KMS database is managing the single **recovery key** that will be used to restore the backup. As discussed in the previous section, we have various ways of distributing and later retrieving backups, but to complete the restoration we need the recovery key. Further, we want to be sure that no unauthorized user can access the key and restore an instance of the wallet.

The recovery key must be protected by its owner. A recovery key might be a seemingly random string of letters and numbers, or might be a string of words from a given dictionary. It cannot be a traditional, simple password as it must have sufficient randomness ("entropy") to be essentially impossible to determine through trying all combinations of characters (called a brute force attack). The key must be created by the owner and then put in a place or places such that it can be retrieved when necessary. As with many security-related challenges, storage will be a trade-off between convenience for the owner and difficulty for attackers. The more convenient to access the key for the owner, the more likely an attacker can also access it. Here's a list of where the recovery key might be stored:

- hot storage – on device
- cold storage – on paper or USB key
- sharded hot storage – social recovery
- memorizing – in your head


The Recovery Key Must Be Protected!

The Recovery Key Must Be Protected!

Managing the Recovery Key: Hot Storage – On Device

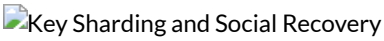
Hot storage refers to any form of online storage, and in this case, it's on the device itself. As with the idea of keeping a backup on the device, the recovery key might reside on the device, protected by the same platform-level security that protects the access key, usually a hardware module accessed with a biometric—a fingerprint or face ID. Of course, on device is not sufficient when we are protecting against the loss of the device itself. In theory, you could also save the recovery key in a password manager (for example, LastPass), in a file in the cloud, or on another device. However, the risk of loss in putting the key in those locations is much higher. Depending on how much value you place on your mobile agent, hot storage other than the device on which the agent operates may be risky as it makes it possible for a remote hacker (someone not physically present) to gain access to it.

Managing the Recovery Key: Cold Storage – On Paper, USB Key

Cold storage is when the key is saved to a device that is not online. For example, the owner could write down the key, print it (perhaps as a QR code of it) or store the key on a USB key that is not left in a computer. Cold storage essentially eliminates the risk of an attacker getting your key remotely because it is not on the Internet. Of course the definition of a safe place for a cold storage device is challenging for several reasons. Keeping it only at home risks losing the key in a fire. Storing the key in a clever hiding place that thieves would never figure out might turn out to be impossible for you to remember when you need it three years from now. Keeping a copy in a bank safe deposit box might be a good idea, but is also costly and a hassle. Keeping a copy at work and/or with a friend or family member might be a good idea, although the more copies floating around, the higher the risk of unauthorized use. Another option is keeping a sealed copy with a lawyer. That approach has the added benefit that the key could be retrieved by a designated proxy should you be unable to retrieve it.

Managing the Recovery Key: Sharded Hot Storage – Social Recovery

A novel approach to the dilemma of wanting convenient, online storage of your key, without the risk of a remote attack is to "shard" the key into pieces and distribute those pieces amongst a set of holders. The sharding process (called "Shamir's Secret Sharing"—a very cool algorithm!) is done such that a subset (for example, any three out of five) of the pieces can be combined to restore the entire key. You need a selection of holders, likely a mix of family members, friends and providers of a holder service. In the future, banks, law firms and cloud providers (such as Google or Amazon) might offer such a service. When you need the recovery key, you request the shards from enough holders through some authorization process where you prove (without your wallet, remember) that you should get the pieces. To prevent the shard holders from colluding to assemble the key without your permission, the holders should not be closely connected and you would not let the holders know who are the other holders.


Key Sharding and Social Recovery

Key Sharding and Social Recovery

Interested in seeing an example of secret sharing in action? Here is a lab demonstrating social recovery using passguardian.com service. The passguardian.com service provides functionality to let you encrypt a secret, create shards, and reconstruct the secret using an implementation of Shamir's Secret Sharing.

While social recovery sounds complicated, a DIDComm content protocol to support the capability would likely be relatively easy to define and implement. The trickiest part will no doubt be recovery, as by definition, social recovery must occur after the owner has lost their agent storage and all their connections. However, as a second or third line strategy, it's rare that it would be used, and if it is ever needed, the hassle will be worth the pain of having nothing.

Managing the Recovery Key: Memorizing

Memorizing a recovery key is a safe way to "store" the key for easy access when needed. Of course, since keys must be sufficiently long and random to be effective, they are also extremely difficult to memorize—and to

remember when you really, really need them. A clever approach that might work for memorizing a sufficiently complex key has been developed by SeedQuest, a team in the US. They've built a video game that looks and feels like a classic Nintendo game but whose purpose is to help you to memorize and (when needed) recover your key.

SeedQuest 3D works by first generating a sufficiently random key, and then guiding you through a game carrying out a short series of 16 actions across four different scenes. Just doing that will produce a key with enough randomness (entropy) to make it impossible to determine using brute force techniques. Later, carrying out the same actions in the same order gives you the same key back. For awhile, you keep the key in readily available cold storage and periodically practice, making sure you always get back your correct key. After you've practiced enough, your memory will (should?) retain the sequence (in theory, at least), and as long as you have access to the game, you can recreate your recovery key. If you have ever become an expert at a video game (Donkey Kong, anyone?), you will understand why this works and is such a clever approach. This can replace a locally stored cold storage key. Of course, you will likely also want an offsite copy or perhaps one accessible via social recovery, should you not be able to remember the sequence or become incapacitated.

The downside? Learning the recovery sequence is more effort than the average person may want to apply to backing up their wallet. Unless of course the game is really fun.

No Backup, No Restore—What's Next?

What if either you have no backup, or all of the recovery techniques you try fail? Your wallet is completely gone. If that happens, you have no choice but to start over, just like you must do if you lose your physical wallet today. Your first stop will likely be to get a foundational credential, such as one provided by a government. You will likely start with paper documents (too bad you didn't keep your recovery key in your documents!) and an in-person visit to a government service center. From there, you will have to go back and collect your other credentials in order of importance, using your foundational credential as a starting point.

Remember we said that wallet backup and restore is crucial? We meant it. Early, small scale proof-of-concepts may be able to get away without reasonable (or any) backup and restore capabilities. However, as soon as there is value in the interactions and effort in restarting from scratch, backup and recovery is a necessity.

Losing a Device

We've covered the case where you need to reinstate either your Aries agent application or your entire device because it is destroyed or you need to replace it. But what if you leave your working phone in a cab and off it goes? Or someone steals your phone. Can a person with your Aries agent pretend to be you? As with backups, the answer is in layers.

Someone Gains Access to Your Agent and Credentials

Your first layer of protection are the mechanisms built into your phone. Presumably, your Aries agent requires the use of the most sophisticated access controls on your phone—ideally biometrics (fingerprint, face ID) or at least requires the entry of a pattern for accessing the phone. This (should) prevent anyone finding the phone from accessing the Aries agent on the phone. If supported, you may also be able to remotely wipe the phone, preventing further use.

The remainder of the protections covered below are necessary based on the assumption that the person (or organization) that has your phone is able to access your Aries agent despite the protections offered by the phone. Remember as you go through this that it is likely that anyone finding a phone would just reset the phone, deleting everything on it, including your Aries agent and KMS data.

The next layer of protection is not yet available in Indy, but has been prototyped. The cryptography to support this is currently being implemented in Ursa. It is a mechanism that adds another step to the presentation of an Indy proof—proof that the device (well, technically, the agent on the device) that created the proof is authorized to do so. Here's how it works:

- When you start to use a device such as a smartphone with an Aries agent, the device is registered in a **device authorization registry** on the Indy ledger you are using. Associated with that device authorization is a list of devices you can use for agency things—proving credentials and the like. Included would be a specific list of devices authorized to revoke the authorizations of other devices. The list of devices authorized to revoke other devices might even include the device of a friend or family member.
- When you create a proof, you use the device authorization registry to prove the device you are using has not been revoked. The cryptographic technique used is similar to proving a verifiable credential you are using has not been revoked. When a verifier looks at the proof, they can check the ledger to verify the device's proof of non-revocation.
- What happens when one of your devices goes missing? The first thing you can do is revoke its authorization to generate proofs by using a device that is authorized to revoke the missing device. From that point on, the missing device cannot prove any credentials, eliminating the opportunity for someone to impersonate you online. A nice bonus? If you find the device is not missing after all (the cab driver returned it, or, you just left it at the office…), you can reauthorize the device and keep using it for creating proofs.

Device Authorization Registry—A Layer of Protection Being Added to Indy

The next layer of protection is in the management of the pairwise DIDs your agent holds for every connection you have. Although someone with your device that has managed to get access to your Aries agent can no longer create proofs using your credentials, they still can message contacts (at least the ones that don't request proofs before acting). Although not yet in any Aries agent implementation, a feature of an Aries agent will likely be created to iterate through all of the connection DIDs and update each DIDDoc. In this case, the change will be to remove the missing device from the DIDDocs, thus eliminating the ability of the device's agent to message any of your contacts.

If you are interested in a longer section on what happens when you lose your device, the Sovrin Foundation has published an article "*What If I Lose My Phone*" that covers this topic in greater depth.

Summary

It sucks when things go wrong but know it will happen. In the case of your digital identity wallet, the need for backup and restore capabilities are paramount. Losing your digital wallet and having to start from scratch is just as painful as with a physical wallet, so the backup and recovery techniques we covered in this section are vital. Further, the techniques for recovery must be easy for everyone to use. Making backups will be automated and easy, but balancing the ease of recovery with the security of the backups will be a challenge. We've described some techniques, but they may not be easy for all users. There are many ideas on this topic and we welcome innovation. And speaking of innovation, this leads nicely into the next and final chapter of the course—all about the possibilities that surround this technology. Carry on!

Chapter 7: Possibilities

Chapter Overview

Through the content and hands-on labs in the course, we hope you have gotten to see some of the potential for this technology—there are so many possibilities! Work is ongoing and ever-evolving as people from around the world use this technology to fix the Internet and address the privacy issues stemming from the missing trust layer. In this chapter, we will explore other possibilities and ways for you or your business to get involved in this important and valuable work.

Learning Objectives

By the end of this chapter you should:

- Understand the main areas where the Indy, Aries and Ursa technology might be applied.

- Be excited about the possibilities that exist for this technology to change the way the Internet currently operates.
- Be raring to delve further into the Indy, Aries and Ursa projects.
- Be sad that this course is coming to an end because it's been fun, right?

Areas of Possibility

Here are seven areas where this technology can play a key role (and more are being thought of every day):

- authentication and authorization
- in-person verification
- proof of training
- decentralized work
- consent receipts record permissions
- delegation of authority with transparency and control
- digital regulatory reporting

Let's discuss each area of possibility.

Authentication and Authorization

We've raised this topic throughout the course, but let's end by asking this question: "Why do we build identity systems?" Two reasons: authentication and authorization.

In computer systems, authentication is the process of verifying the identity of an entity (person, other system, etc.) before allowing access. Who is it that is trying to access the system? Sometimes we need to know the unique person, other times, we don't care who it is, but we do care that when they come back, we connect them with the account previously created for that person. Authorization in the computer system context is determining what an entity is allowed to do within the system—what data they can see, what changes they can make. As we described in Chapter 1, both are historically difficult to do, largely because information from the user cannot be trusted. Authorization and authentication have become enabled by identity providers (like Facebook and Google) providing data about the user. By building authentication and authorization on the foundation of verifiable credentials, the user is back at the center of the process, fully in control of with whom and how much data they share. The challenge is how to transition from the current identity provider world to one that uses verifiable credentials.

One exciting solution that has been developed by New Zealand's Mattr Global, in collaboration with the Government of British Columbia (BC) in Canada, implements an identity provider (**vc-authn-oidc**) that uses verifiable credentials. The solution uses the same OpenID Connect (OIDC) protocols used by Google and Facebook but bases its trust on verifiable credentials. That provides a path for every online service that currently supports "Login by Facebook" to add a "Login by Verifiable Credentials" mechanism.

Example

The Ministry of Justice in BC needs to provide access to some of its online services to actively practicing lawyers in the province. How does the service know someone asking for access (authentication) is currently permitted to practice law in BC? A traditional approach to this problem might involve an integration between the government service and the legal entity that authorizes a lawyer to practice—The Law Society of BC. When a person requests access, the government service must find out (exactly!) who they are, and then ask the Law Society "Is Alice a currently practicing lawyer?" It's complicated.

With verifiable credentials, the challenge becomes much easier. The Law Society tracks (as required by law), who are the practicing lawyers in BC. They have an online membership portal and control access to that of only lawyers. On that site, they provide a service where a practicing lawyer can request a verifiable credential that says "I am a practicing lawyer" (for example, not retired or currently suspended). At the BC government site, a user requesting access is challenged to provide a verifiable credential demonstrating they are a practicing lawyer (provided the government trusts the credential issuing process of the Law Society). If they are, they are granted

access, based on the claims in the credential. If the lawyer retires or is suspended, the Law Society can revoke the credential, and access to the government system will be removed. And, since the BC Government already uses an OpenID Connect process to grant access, the **vc-authn-oidc** module built by Mattr Global fits right into the existing government infrastructure.

Expand that model out. Other government systems grant access to practicing lawyers. Private companies offer online services to lawyers. And not just lawyers use such systems. Other credentialed individuals—doctors, accountants, real estate agents, tradespeople and so one, do as well. In Britain, the National Health Service is working with Truu.id to implement a medical professional credentialing system that reduces to minutes a paper process that can take a month to authorize a doctor new to a hospital to provide medical services. Suddenly a raft of one-off solutions, each with multi-party integrations can be solved with a single question and a single integration: does the party wanting access have the right verifiable credentials?

In-Person Verification

Online trust frameworks are often rigid and scripted (e.g. authentication systems, AirBnB, Lyft). In the physical world, trust frameworks are often more flexible, relying on trustworthy (paper) credentials. When you go into the pharmacy to purchase medications, they might want to know you're over 18. In the US, at least, it's common to ask for a driver's license for this purpose. Why?

A driver's license is an identity credential for use in a specific administrative domain: licensing drivers. Nevertheless, because of its implementation, it is used outside the administrative domain for which it was designed. A driver's license is implemented as a decentralized, trustworthy credential that serves as a container for a specific set of attributes. Its veracity is (in theory) easily checked by recipients.


In-Person Verification

Two important properties allow this kind of use:

- The state is usually seen as a trustworthy party and people believe its attestations about the subject's attributes.
- Driver's licenses are difficult enough to forge that we believe they have not been tampered with.

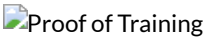
Because of this implementation, people can be the conveyors of trustworthy attestations about themselves (called claims). You can start a business today and decide you want to use a driver's license for proof of address and start doing it immediately. No permission required. No integrations. No APIs.

Using verifiable credentials isn't that easy—you need some technology—but you don't need permission, and the integration is with a global entity (the blockchain), not with a lot of individual services. With verifiable credentials, in-person verification becomes more trustworthy and safer. Take proving your are old enough to drink in a bar, for example. At a VC-enabled bar, you can present claims from a source the bar trusts (like a driver's license authority) that prove you're over the necessary age and your picture (that proves you are you), and you are admitted. No need to show your physical driver's license with its abundance of extra information—your address, driver's license number and so on. Since patrons can come from anywhere in the world, the bar's verifier agent needs to know the identity (the DID) of many driver's license issuers (every state, province, country, etc.), but that's all they need to know about them. Most importantly, they don't need to integrate with each of them for the process to work. That makes forging a verifiable credential essentially impossible.

Back to the pharmacy example. Today, you present a prescription and your driver's license. In the future, you might have to prove that you have a verifiable credential for the prescription, that the name on the prescription matches the name on your driver's license, and that you are old enough to receive the prescription. Such systems are being investigated in proof-of-concepts by a variety of health authorities around the globe.

Real-Time Credentialing

In most of the examples we've talked about to this point, the credentials are long-lived (professional credential, driver's license, etc.) and it's not difficult to imagine trusting those in paper form as a one time check prior to giving a person long term authorization—for example, giving a lawyer a building pass to a courthouse by verifying their paper identity and lawyer credentials. If the lawyer loses the practicing lawyer authorization, the building pass will still work, but we accept that risk as "the best we can do." But what about credentials that are short lived, and that must regularly be renewed? How do we check those in the paper world?


Proof of Training

Often when a construction worker is given access to a site, she is required to have taken site-specific safety training consistent with the current state of construction. If you're in a building that is under construction, there needs to be constant updating of what is the best way to get out of the building in case of fire. That's continuous safety instruction. Today, the certification of the training is paper, but it's too much work to verify those credentials each time access is required. At most, only if there is a problem and an investigation is needed do the paper credentials get checked. With verifiable credentials, the process become proactive and near real-time.

Access becomes based on the presentation of the necessary credentials from the authoritative source. The check is fast, happening with messages going back and forth between the issuer and verifier devices. Since the credentials are carried by the person (e.g. the lawyer, the construction worker in the examples above), they can be checked in real-time as access is needed, regardless of an Internet connection. The check is not of a proxy for the real credentials (not a building pass) but of the actual credentials issued by a trusted authority.

The incentives for changing to this model are interesting. While perhaps there is little risk reduction in the lawyer use case, it does reduce the current "who has a building pass?" problem with which many companies struggle. In the safety case, the incentives are more powerful. At minimum, the construction company can be certain that only individuals with proper safety training are given access to the site. Further, there might be a financial incentive to the company in the form of reduced insurance costs, since they can now guarantee (with evidence) that only safety trained individuals can work on the site.

Decentralized Workflow with Verifiable Credentials

Governments are a major supplier of verifiable credentials. Every license, permit and registration that is issued by a government organization is a credential, and issuing them as verifiable credentials enable the implementation of automated systems for processing such credentials. Such a change will replace all of the paper-based verifications that happen today, improving efficiency for all participants and preventing fraud.

The province of British Columbia is working on the challenge of enabling governments to issue credentials with its Verifiable Organizations Network open source project. In one proof of concept, Greenlight, the VON team is looking at the concept of decentralized workflow.

Example

Here's how it works.

Say an organization wants to get a business permit from their local municipal government for a new business they are starting. They go to the municipality and apply. In doing so, they are informed that in order to get a business permit, they first must prove possession of some other licenses and/or permits. Often the prerequisites must be from organizations in other jurisdictions, such as a provincial tax number, a health authority certification and so on. Investigating these licenses and permits, they find more prerequisites—to get a provincial tax number, you have to have a registered company, to get a health authority certification, you have to have workers insurance. In every jurisdiction, and in specific cases (restaurant versus pub versus retail store versus office building) the requirements differ. And, to "prove" each permit or license, you have to submit a paper copy (or scan) and someone must manually verify it. It's usually a nightmare, significantly increasing the cost, time and frustration of starting or expanding a business. And it happens in every jurisdiction across the globe.

Enter Greenlight. It starts with the idea that all the issuers of permits and licenses are verifiable credential issuers. Next, those same issuers are also verifiable credential verifiers, proving the prerequisites by receiving a proof containing the necessary claims. With those in place, Greenlight automatically calculates for a given permit, what sequence of credentials a business needs to get that permit. Here's how the process works:

- When starting an application for the permit ultimately wanted (e.g. the business permit in the example above), the business gets a proof request (if any) from the application that defines the prerequisite verifiable credentials needed.
- For each credential referenced in the proof request, the step above is repeated of the issuing authority for that credential.
- The process repeats until the root authorities are found—issuers that do not require any prerequisite credentials.


The Government of British Columbia's GreenLight Mapping Tool
The Government of British Columbia's GreenLight Mapping Tool

At that point, a map for the business's journey is displayed—all the credentials they need in the order in which they must be collected.

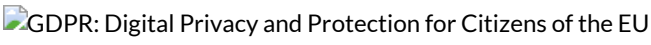

GreenLight—Decentralized Workflow in Action
GreenLight—Decentralized Workflow in Action

Next, the credentials that the business already has are found (if any) and the map is color-coded to indicate what credentials they have, what credentials they can get now (because they have the prerequisite credentials) and what credentials they need but can't get yet (because they don't have the prerequisites). Links are added to the map to allow the user to jump straight to the application process of each issuer, providing the required proof for that issuer. At the Greenlight website you can see the Greenlight proof of concept in action.

The power of Greenlight is that the entire workflow is decentralized and generated dynamically. Since each issuing authority is independent, its application requirements (e.g. prerequisites) can change at any time. It does that by changing its proof request, and the next time Greenlight runs, the updated map is presented.

Consent Receipts

On May 25, 2018, The General Data Protection Regulation (GDPR) was implemented. The GDPR is a regulation in EU law on data protection and privacy for all individual citizens of the European Union and the European Economic Area. It also addresses the transfer of personal data outside the EU and EEA areas.


GDPR: Digital Privacy and Protection for Citizens of the EU

GDPR: Digital Privacy and Protection for Citizens of the EU

The biggest direct impact on users has been all of the notifications appearing on websites asking your consent for the service to track information about you for some purpose. When you click "OK," have you ever wondered where the consent goes? Presumably, it gets saved by the company, and associated with your usage on the site—likely with a cookie of all things.

What if instead of the company recording the event, the company gave you a receipt, just like the receipt you get when you buy something? If the receipt was a verifiable credential, you have indisputable proof of the agreement. What does that enable?

- At any time, you can see to what companies you have given your consent.
- You can use your consent receipt to withdraw your consent, just as you use a store receipt to return merchandise.
- If a company has some sort of negative event (a data breach, a violation of GDPR, etc.), you can find out if you might be affected.

- If you want to participate in a court action because a company violated the terms of the consent agreement, you have evidence of exactly the terms to which they agreed.

The Kantera Initiative is a global consortium improving trustworthy use of identity and personal data through innovation, standardization and good practice. Kantera hosts the Consent & Information Sharing Work Group, which created a consent receipt specification. A number of Kantera members have products based on the specification, some of which are based on verifiable credentials.

Delegation of Authority with Transparency and Control

Delegation of authority as a use case is everywhere. Most of us are familiar with delegations in a personal context—a power of attorney that gives a trusted family member or friend the right to act on our behalf when we are unable to do so. Delegation of authority is everywhere in a business context as well. Any business is ultimately owned by someone or some group. Every employee is given some form of delegation of authority that derives from the ownership of business. The board of directors delegates authority to the CEO, the CEO delegates authority to the executive team, and so on. Delegation goes outside the business as well, such as lawyers and accountants that act on behalf of the business in specific business dealings.

Delegation of authority is a prime use case for verifiable credentials. An issuer, the entity with authority, issues a verifiable credential to the holder, who is being given the authority. When necessary, the holder proves the authority to a verifier. When the authority is to be withdrawn, the issuer revokes the authority, thereby preventing the holder from using it in the future. That's exactly the same as the verifiable credential model.


Delegation of Authority Model Versus Verifiable Credentials Model
Delegation of Authority Model Versus Verifiable Credentials Model

(Continued on next page).

Delegation of Authority with Transparency and Control (Cont.)

Example: Lawyers and Staff

An example of the use of delegation through verifiable credentials is an extension of the lawyer authentication use case from earlier in this chapter. Recall in that example that the authority for identifying lawyers in a jurisdiction (for example, the Law Society of British Columbia), issues a practicing certificate to a lawyer, and the lawyer can use that to authenticate at a government online service. That works in theory, but in a law practice, that's not always sufficient. Using an online service is often done not by the lawyer directly, but by members of the lawyer's team—delegates. Often the lawyer uses his or her credentials to access an online service, but has the work on that service done by a staff member. While this is necessary from a productivity perspective, it's often against the service's terms of use, and possibly a violation of the lawyer's legal responsibilities.

A more effective and transparent approach is to allow the lawyer to delegate their authority to that staff member so the staff member can use their own credentials for authentication. Being considered in the BC use case is that the lawyer can request the Law Society issue a credential to their staff member(s). The responsibility for the behavior of the staff member in using the credential is with the lawyer, and all of the parties are aware of who is doing what on the system. Revocation of a staff member's delegation can happen as needed, and the revocation of the lawyer's credential triggers the revocation of all of the lawyer's delegations.


BC Provincial Government Use Case
BC Provincial Government Use Case

Such a delegation approach can be implemented in many use cases with the verifiable credentials currently available in Indy. The delegating party asks the authority to issue a delegation verifiable credential to a delegate. The delegate can use that credential for authentication and authorization.

(Continued on next page).

Delegation of Authority with Transparency and Control (Cont.)

Delegatable Credentials

While current Indy features can enable delegation, a new mechanism is currently being investigated that promises even more power. A delegatable credential is a special kind of credential that is issued by a service to a holder such that the holder can subsequently delegate the credential to other entities, and those entities can also delegate the credential, ad infinitum. Core to the mechanism are a set of capabilities enabled by possession of the credential. During delegation, only the capabilities the holder has been granted can be delegated, and they can choose to only include some of the credentials. Here's an example, based on the image below.


Delegateable Credentials Example
Delegateable Credentials Example

We start the example with a fancy car that requires a credential with the capability to "drive" the car. It is sold to a car rental company, and the company is given a verifiable credential proving they own the car. To enable the car to be rented, the following delegations can occur.

1. A national rental car company creates a delegatable credential that includes proof of ownership, has capabilities to "rent," "maintain," "sell," "drive" and "delegate," and issues the credential to the Houston office.
2. The Houston office rents the car to Alice and issues a delegation of the credential to her with the capabilities to "drive" and "delegate" the car for no more than 7 days and only within the state of Texas.
3. Alice goes to a restaurant and issues a delegation of her credential to the valet with only the capability to "drive," and only for two hours and within 100m of the restaurant.

Lacking the "delegate" capability, the valet cannot delegate the credential to anyone else. If something goes wrong (Alice is found to be a wanted fugitive), the delegated credentials can be revoked and the capabilities can no longer be used.

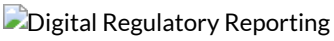
Delegatable credentials open up exciting new possibilities and a wide range of new applications for verifiable credentials. To learn more, check out the proposed Aries Request for Change (RFC) about delegatable credentials.

Digital Regulatory Reporting

Regulatory reporting is a field ripe for innovation using verifiable credentials. There is an ongoing battle between pro-regulation and pro-business proponents about the value of regulation.

A pro-regulation view is that regulation is necessary to keep "bad actors" from reducing their costs by not paying for all the impacts of their products, and leaving it to society to foot the bill. The manufacturers of the 50's and 60's (and beyond) that quietly pumped pollution from their factories into nearby rivers and streams are prime examples of the long term costs of not regulating business activities.

A pro-business view is that regulation means endless paperwork and reporting that is costly to generate and rarely reviewed and monitored because it's too expensive for government to act upon. Good actors adhere to the regulations and reporting despite the costs, while bad actors continue to ignore regulation compliance to limit their costs with little chance of being caught.


Digital Regulatory Reporting

Digital Regulatory Reporting

What if authentic regulatory reporting data could be generated from its source, digitally shared and digitally processed for compliance? The cost of regulatory paperwork would drop and the effectiveness of the regulations would increase. Good actors would be able to prove, at a low cost, their compliance with regulations, and bad actors would be forced to either comply or shutdown. That is all possible based on verifiable credentials.

Consider the energy industry, a contributor of carbon emissions that drive climate change. As the world transitions to other energy sources, incentives will grow for those reducing their carbon footprint, and costs will rise for those failing to do so. Even if governments fail to act, society will (eventually) punish those that knowingly profit from business practices that damage society, as happened with the tobacco and opioid industries. But for companies to demonstrate that they are reducing their carbon footprint, they must present trusted evidence. As the Oxford Institute for Energy Studies (an independent energy markets think tank) recently reported,

"The LNG community needs to replace an 'advocacy' message – based on the generality of emissions from combustion of natural gas being lower than from other fossil fuels – with certified data on carbon and methane emissions from specific elements of the value chain for individual projects."

That sure sounds like a need for verifiable credentials! By collecting authentic data from authorized devices (verifiable credential issuers) at the source—the well head—and continuing through the production chain, energy producers can demonstrate their carbon impact. As production improvements are made, as moves are made to lower impact fossil fuels, and off fossil fuels entirely, the impact of the changes can be quantified and rewarded or penalized as appropriate.

Let's finish with that thought, using verifiable credentials to incentivize companies to fight climate change. That's powerful!

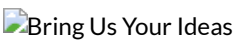
How to Get Started

The Hyperledger projects are open source, where ideas and code can be publicly discussed, created and reviewed. There are many ways to contribute to the Hyperledger Indy, Aries and Ursa community. To get started in any one of the projects, start at the Hyperledger Wiki for the project:

- https://wiki.hyperledger.org/display/indy
- https://wiki.hyperledger.org/display/aries
- https://wiki.hyperledger.org/display/ursa

Upon visiting these pages, you will notice that there are links to all kinds of valuable information including links to documents, data and details about the projects, repositories, project management information, rocketchat discussions, collaboration tools, community meetings and more.

If you are wanting to build on these technologies, focus in particular on Aries. With Aries, you can start building applications that depend on the technologies covered in this course. If you are a developer, you'll want to start with the follow on to this course: *LFS173x: Becoming an Aries Developer* (coming early 2020).



Bring Us Your Ideas

Summary

That's a wrap! Thank you for exploring Indy, Aries and Ursa with us. If you're a developer, or a developer-wanna-be who is not afraid of a command line, come see us in the follow on to this course, *LFS173x: Becoming an Aries Developer* (coming early 2020).

Read More

1 thought on “Introduction to Hyperledger Sovereign Identity Blockchain Solutions: Indy, Aries & Ursa”

ปั๊มไลค์ says:
May 26, 2020 at 6:08 pm

Like!! Really appreciate you sharing this blog post.Really thank you! Keep writing.

Reply

Leave a Reply

Subscribe

Your email address will not be published. Required fields are marked *

COMMENT

NAME *

EMAIL *

Save my name, email, and website in this browser for the next time I comment.

Post Comment

Search …

LEARNTHINGS.ONLINE TELEGRAM GROUP

Subscribe

RSS

IPFS for Beginners – Interact With IPFS By Javascript

In this article, we'll learn how to interact with IPFS by javaScript programming language. It's one way to make your own application to interact with IPFS. The post IPFS for Beginners – Interact With IPFS By Javascript appeared first on LearnThings.Online.

Facebook Rename Its Libra Wallet Project Calibra to Novi

2020 May 26, Facebook rename its Libra wallet project Calibra to Novi. It makes its name more separate from Libra. Novi plans to launch its App in 2020. The post Facebook Rename Its Libra Wallet Project Calibra to Novi appeared first on LearnThings.Online.

Libra Appoints It's General Counsel, a Former HSBC, and Goldman Sachs

On May 19th, 2020, the Libra association appoint Robert Werner, an Ex-HSBC & Ex-Goldman Sachs the founder and CEO of GRH Consulting, as its general counsel. The post Libra Appoints It's General Counsel, a Former HSBC, and

Goldman Sachs appeared first on LearnThings.Online.

Subscribe