SolidVC - A Decentralized Framework for Verifiable Credentials on the Web

by

Kayode Yadilichi Ezike

S.B. EECS | Massachusetts Institute of Technology | 2017

Submitted to the Department of Electrical Engineering and Computer Science
in Partial Fulfillment of the Requirements for the Degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2019

Author .....
Department of Electrical Engineering and Computer Science
February 20, 2019

Certified by .....
Lalana Kagal
Principal Research Scientist
Thesis Supervisor

Accepted by .....
Katrina LaCurts
Chair, Master of Engineering Thesis Committee

SolidVC - A Decentralized Framework for Verifiable Credentials on the Web

by

Kayode Yadilichi Ezike

Submitted to the Department of Electrical Engineering and Computer Science
on February 20, 2019, in Partial Fulfillment of the
Requirements for the Degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

Credentials are an integral part of our lives, as they express our capabilities and enable access to restricted services and benefits. In the early 2010s, the Verifiable Claims Working Group of the World Wide Web Consortium (W3C) proposed a specification for what is now the Verifiable Credentials Data Model. This living specification, which is still in development, outlines a cogent framework for the issuance, storage, presentation, and verification of credentials on the Web. Many of the leading Verifiable Credentials projects leverage Distributed Ledger Technology (DLT), potentially compromising Web interoperability and sometimes exposing otherwise personal data. SolidVC is a decentralized Verifiable Credentials platform built with the open protocols of the Web. It is implemented on top of Solid, a Web framework developed at MIT in 2016 that allows decentralized applications to interact with personal user data to provide services in an access controlled environment.

Thesis Supervisor: Lalana Kagal
Title: Principal Research Scientist

Acknowledgments

Every great feat involves the confluence of a diverse set of people contributing a diverse set of skills, ideas, and overall support at the perfect time. In this section, I will attempt to express due gratitude to everyone that made this thesis possible.

I'll start by thanking Tim Berners-Lee and Lalana Kagal for accepting me into the Decentralized Information Group and giving me the opportunity to execute on an amazing project. Tim, your visionary leadership is one that I have never taken for granted and your passion for improving your revolutionary brainchild project, the Web, inspires me to continue to fearlessly seek better alternatives to the status quo. Lalana, I couldn't have asked for a better research supervisor. Your brilliance is not only in your ability to understand the current needs of academia, but in your ability to communicate these to a young and relatively inexperienced student and in a way that illustrates its significance. Thank you for showing me nothing but the best of guidance, patience, and support throughout my time in your group.

Thanks to Dmitri Zagidulin and Ruben Verborgh for lending your expertise and patient support for all things Solid and Linked Data. Thanks also to Juan Freuler for providing insights on the role of Web technologies on policy and society. I have a tremendous level of respect and appreciation for the three of you.

Anne Hunter, the Course VI department truly has an angel in you! Your ability to interact so graciously with students in the face of an onslaught of responsibilities is a quality that I have always admired. Additionally, your lighthearted approach to life has reassured me that one can certainly lead a successful, yet fun life.

Throughout my half-decade stint at MIT, I have encountered some of the best and brightest people in the world. Thanks to every Professor, Teaching Assistant, Research Assistant, faculty, staff, and student of every level with whom I have interacted. You are all very special. Special thanks to Intervarsity, Chorallaries, and African Students Association for giving me a community away from home!

I would like to take this time to thank all of the friends who have supported me throughout my time here. Hassan Kané, thanks for helping me live life on the edge and for being an inspirational friend. Dayanna Espinoza, thanks for keeping up with me throughout all of our all-nighters and for being a caring friend. Ryan Robinson, thanks for taking on the risk as my first friend at MIT and for being a supportive friend. Urenna Nwogwugwu, thanks for tolerating such a goofball of a friend since elementary school and for being a constant over the years. Aritro Biswas, thanks for being by my side since freshman year and for serving as a reliable source of comic relief throughout my toughest times at MIT. To those not mentioned, thank you sincerely for shaping my MIT experience in your own unique way.

To the Nigerian Catholic Community (NICCOM) and New York's Nigerian/Igbo community at large, thank you for believing in me from day one. This thesis is the reward of every aunty, uncle, and kid that supported me over the years. Igbo Kwenu!

Family, there is not enough paper in the world to duly thank you for all the love and support you have shown me from day one. Mommy and Daddy, thank you for being the best parents that a Nigerian boy growing up in America could possibly ask for. Your sacrifices over the years have been second to none (and they are paying off!). I love you guys! Jide, thanks for being a great role model. I have always looked up to you and I appreciate the time I have gotten to grow closer to you over the past 3-4 years in Boston and as a roommate over the past half-year. Best of luck with your PhD program at MIT! Adaugo, thank you for being more than a sister. You are one of my best friends and I don't take our relationship for granted. Best of luck charting your academic, professional, and personal path beyond Cornell and please know that I am always here for you! Last, but not least: Dioka! You are the light of the household and you have always inspired me to achieve more than I believe possible at times. I pray that you continue to grow and develop into a strong, healthy, and independent man that can contribute meaningfully to society. I love you Didi!

Contents

List of Figures

List of Tables

Chapter 1

Introduction

There has always been a need for us to reliably identify and vet individuals in our society. Everything from deciding whether someone can re-enter a live concert to determining whether a person is allowed to run a medical practice requires a robust mechanism for ascertaining one's identity and capabilities. Any system, technical or otherwise, which provides this critical service of certification must achieve a core set of desiderata, chief of which include secure distribution of credentials to deserving individuals, seamless assertion of credentials at a later point in time, and reliable verification of the provenance of credentials.

With the overall deterioration of public trust in politically unstable countries, including none other than the United States and Italy [12], the work around verifiable identity claims has become more important than ever before. People are desperately seeking solutions to everything from "fake news" [13], which has become a veritable commandment for major news outlets and social media platforms, to online identity fraud, which has attracted the attention of technical and financial institutions alike [14]. While there is no magic-bullet solution for these issues, the identity and credential communities have produced a plethora of proposals, specifications, and implementations for technologies of varying degrees of security, complexity, and scope that solve different segments of the puzzle [15]. However, the technical development in this space has not come without its limitations.

Many credentialing services have wooed the masses with their highly technical

approaches to certification, that almost certainly has blockchain or some other form of Decentralized Ledger Technology (DLT) baked into its construction. Others have sacrificed privacy and self-sufficiency either out of lapse in foresight or perception of technical debt. Whatever the reason, I believe that there is still space for a certification service that uses the best of what the Web has to offer to create a decentralized ecosystem of standardized, verifiable credentials.

In this paper, I propose SolidVC as a robust solution to many of the issues outlined above. SolidVC is a decentralized Verifiable Credentials framework that empowers users to seamlessly manage their personal credentials, while leveraging the best of time-tested technologies, such as Linked Data (or the Semantic Web); Digital Signatures; and, most fundamentally, the World Wide Web (henceforth, the Web).
SolidVC provides mechanisms for the following critical services:

- Requesting credentials from any other participating user
- Issuing credentials to any other participating user
- Verifying the credibility and provenance of a credential
- Revoking credentials that were previously issued

Also included are the following supplementary services:

- Downloading credentials to one's personal computer
- Sharing credentials with any other participating user
- Reviewing relevant messages from other participating users (such as credential requests and credential issuance notifications)

This thesis includes a comprehensive discussion of my approach for providing the aforementioned credential management services as well as an outline of an expected use case.

Chapter 2

Background

In order to effectively identify the contributions of this paper, I present in this chapter a few fundamental technologies that underpin SolidVC [4].

2.1 Verifiable Credentials

In modern society, there are several aspects of identity that citizens are expected to produce for privileged access to certain activities and services. Drivers are expected to conjure up their license on demand upon a police officer's request. International travelers are expected to locate their passport when booking a flight. Young adults are expected to present government ID before ordering an alcoholic beverage. Each of these documents serve as evidence for an individual's right to partake in a particular activity. However, as constructed, they each bear considerable risk of loss, theft, and exploitation, not to mention that they are often cumbersome to manage and assert.

Verifiable Credentials (previously Verifiable Claims) is a specification that was introduced by W3C's Verifiable Claims Working Group in the early half of this decade. The goal was to establishing a standard for issuing and verifying statements associated with an entity in a secure, private, and mechanic way [6]. Each credential contains a series of claims (i.e., RDF [23] statements) asserted by an authoritative entity (the *Issuer*) about another entity (the *Subject*), owned by yet another entity (the *Holder*; often identical to the Subject but, not always [7]) and verified by a final

entity (the *Inspector*/*Verifier*). If implemented with open standards, as is the hope of many interest groups [6], users will be able to store their credentials in an identity provider (IdP) of their choice, migrate them seamlessly between IdPs, and assert them at will. It is the belief of many in the Web credentialing community that Verifiable Credentials will automate authentication and authorization for consumers and consequently reduce the cost of service delivery for providers.

2.2 Solid

Solid is a Web-based framework of decentralized applications interacting with carefully managed personal data to provide desirable services to users [2]. The goal of Solid is to empower users to regain control of their data by removing it from the sticky "data silos" of social media and consumer application juggernauts and returning it to the full ownership and control of users, where it once belonged.

In the Solid ecosystem, users have access to applications that provide services such as contact management, meeting scheduling, and e-mail. These users store their personal data in a personal online datastore (pod), so that whenever an application wants to access information about a user, it must first perform an authentication protocol to discover their identity and profile data via a unique WebID assigned by an identity provider of the user's choice. One of the key requirements of pod servers, which store Solid pods, is that they must support Access Control Lists (ACLs), which place constraints on the type of access that applications may exercise on resources. For example, one constraint that a user might decide to apply to their pod is that applications may only append to documents or containers. Properties like these will be key for the SolidVC architecture, discussed in later chapters.

2.2.1 Solid Benefits

There is a great variety of tools that could have been used to offer credential services, as will be discussed later. The following are some of the reasons I decided to leverage Solid to implement SolidVC:

- Solid was designed with the goal of empowering users to establish a greater control of their data. It uses ACLs as a mechanism for safeguarding access to their personal data and decentralized storage as a means of minimizing data exploitation by third-party applications.
- In Solid, users store their data in a pod that can be hosted anywhere of their choosing (including their own personal computer).
- Solid uses Linked Data (discussed in Section 2.3) to help users learn useful information about their data and the data of others to the extent that they are allowed.
- Solid exposes a SPARQL [30] endpoint for users to perform authenticated queries on user data. SPARQL will be discussed in greater detail in Section 2.6.
- In Solid, there is no notion of a central database that stores and manages all data and facilitates communication between users. Rather, Solid applications leverage Linked Data and WebIDs (discussed in Section 2.4) to discover other users on the platform, learn publicly available information about them, and interact with them.

2.3 Linked Data

In the early days of the Web, it operated solely as a global Document Management System (DMS) [1] that stored documents for later use and could be linked to other documents via hypertext. As many readers in this community are likely familiar, these "links" (as they are colloquially referenced) are typically in the form of formatted text that navigate the document's viewer to another document. If the viewer is lucky, these links are sometimes titled with human readable language and/or described with surrounding text. However, this is by no means a requirement for publishing Web documents. Worse yet, even this construction proves challenging for machine readability, and consequently data relationship discovery.

If it is unclear why the above setup foils most machines, imagine designing a Web crawler program charged with the rather onerous task of discovering all relationships intended by Web developers in a subset instance of the Web as constructed above, based solely on the text in the connected documents. For example, in a research setting, this Web crawler would have to learn to classify some links in a research paper as literature references; others as personal contributions, such as a URI of the resource implemented by the author; and others as personal information, such as a link in the Acknowledgments section that advocates for a local pizza shop that offered the best slices in town, great customer service, and a friendly studying environment during the writing period of the paper. With this arbitrary set of relationships in a research paper alone, hopefully the reader can glean a clear sense of how such an underspecified task as relationship discovery can be challenging for humans and machines alike. It requires a non-trivial level of machine processing, parsing, interpolation, and intelligence to discover meaningful relationships. Additionally, even if one could build an accurate enough system that meets these specifications, at the end of the day, it's just that: accurate enough. There is no reliable way to guarantee 100% relationship discovery accuracy unless the Web was annotated in a more meaningful way.

The root of the aforementioned problem is that the Web in its native form is like an unlabeled graph. The hypertext links often represent some sort of semantic relationship between documents intended by the author and are such an integral lever of our experience on the Web. However, there are no explicit semantics associated with these links. The only way that data consumers can ascertain with 100% accuracy the intended relationships of documents published by Web developers is if they had an explicit mechanism of naming links. This is where Linked Data comes in. Linked Data (or the Semantic Web, as it is sometimes referred) is a set of principles for generating semantic information on the Web. The most natural mental model for Linked Data is a labeled graph. Each node in the graph represents an arbitrary entity, which can be anything from a person to an organization to even a grouping of relationship attributes [24].

A sizable subset of the nodes in Linked Data are documents that can be deref-

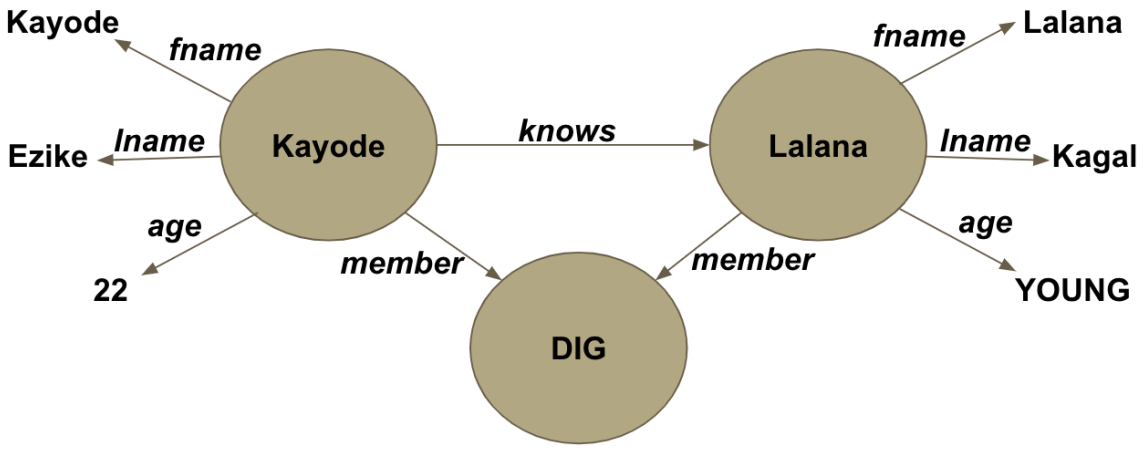

Figure 2-1: Sample RDF Graph

erenced (visited on the Web) from the Uniform Resource Identifier (URI) - which also serves as a Uniform Resource Locator (URL) in this case - that identifies the document on the Web. Inside these documents are a series of statements about the document that are expressed as a series of (SUBJECT, PREDICATE, OBJECT) triples [1]. An example triple is the following:

<Kayode Ezike> <knows> <Lalana Kagal>

This abstract representation of relationships on the Web is known as the Resource Description Framework (RDF) [23] and is often represented as a graph structure. There are many types of statements in RDF, including assertions of employment at a company, ownership of a phone number, or association of a signature with a document. Figure 2-1 illustrates a sample RDF graph that encompasses the simple RDF statement above.

2.3.1 Ontology Design

The discussion above explains most of the key components of Linked Data, but it is missing one critical part. As presented thus far, Linked Data does not allow for a

[1] These are often called quads when referenced from outside of the context of the document. In this terminology, the fourth element is an identification of the remote document storing the statement

meaningful Web, as each document simply has its own set of statements that declare relationships between nondescript entities. For example, in the statement above, there is no way for a consumer of this document to disambiguate the name "Kayode Ezike" (uniqueness notwithstanding). This is not a label that is globally identifiable, since there is no uniform means of resolving it as an identifier. Neither is "knows" or "Lalana Kagal". The type of identifiers used for the subjects and objects in this case are application dependent, but are commonly expressed as WebIDs or DIDs. In Section 2.4, I will discuss WebIDs in greater detail, as this is the more relevant identifier scheme in the current Solid and SolidVC ecosystems. As for predicates, such as *knows*, there are stricter requirements for what they mean, so that any two consumers reading a document have the same understanding of its meaning. This disambiguation is outlined in an ontology. Ontologies describe the meaning of terms, such as *knows*, *name*, *homepage*, and *publicKey*. They are specified at a URL that both identifies the ontology unequivocally and explains to consumers reading an RDF document what exactly is meant by the contained predicate terms, including their domain (subject space) and range (object space). For example, there is a popular ontology called Friend of a Friend (FOAF) that describes many different types of social attributes and relationships. With this framework established, a more precise representation of the running RDF statement in this chapter is the following:

<https://kezike.mit.edu/profile#me>
<http://xmlns.com/foaf/0.1/knows>
<https://lkagal.mit.edu/profile#me> [2]

SolidVC includes an ontology known as *svc* [5] that describes the various kinds of credentials and messages that are managed, displayed, and transmitted on the platform.

2.3.2 Linked Data Benefits

There are many advantages for Web users to leverage Linked Data on the Web:

[2]These are not real websites and are only used here for demonstrative purposes

- Users are able to query the Web in a manner similar to that of querying a relational database. Tools such as SPARQL [30] support this kind of access.
- Users are able to discover useful information about other people on the Web. This opens up an opportunity for increased collaboration and community building around certain skills, traits, interests, values, and ideals on the Web.
- Stakeholders have a standard means of defining, annotating, and understanding data on the Web.
- Users are encouraged to share, reuse, and extend open ontologies so that a network effect of sorts is created around established communication standards on the Web. In other words, with this open source philosophy, more users will be able to understand each other and there would be less confusion in communication and information processing online.
- Users enjoy a concise means of representing data and asserting claims on the Web. For example, an individual's personal website can simply be a compilation of statements that link to other documents, such as their CV/resume, their alma mater's homepage, their Github account, their Twitter account, and a professional image.

For many of the Linked Data operations, I employed an open source library called *rdflib* [25]. This tool is one of the primary technologies of the Solid ecosystem that enables users to read, write, update, and delete local and remote RDF data in Solid. In later sections, I will explore even further why Linked Data is such an integral component of SolidVC and Solid (short for Social Linked Data) at large.

2.4 WebID

WebID is an authentication protocol that uses URIs to uniquely identify users on the Web. In its original specification, it employs public-key cryptography to associate a user with a WebID. For some time, Solid was using this variant of WebIDs until it

decided to switch to the standard username-password authentication protocol. However, WebIDs continue to be a staple of the platform and the Web for purposes of account access and identification. The typical form of a WebID is the following:

https://uname.example.com/.../profile#XYZ, where:

- *uname* is the user's username
- *example.com* is an IdP
- */.../* is a valid path hosted by the IdP
- *profile* is the name of the user's profile document, which contains RDF statements about them
- *XYZ* is a URI fragment that allows relative access to a secondary resource from a primary resource

WebIDs are the identifier of choice in the SolidVC ecosystem because of Solid's native support thereof. I am content with this decision because it is the foundation of much of the functionality on the platform, including Linked Data Notifications, discussed in Section 2.5. Additionally, I believe that http(s) URIs are great identifiers because many people are already familiar with this Web-based construct and already own software that readily supports its resolution and dereferencing: the Web browser.

2.5 Linked Data Notifications

Most messaging platforms today are closed systems that rely on a controlled, centralized means of coordinating communication in a particular format that is available only to participants in the platform. This is great for many use cases, but not all. There are situations in which a user may want to share information with a group of people without the need for agreeing on a single third party application for each recipient to use. In other words, they would simply like to use the Web - arguably the

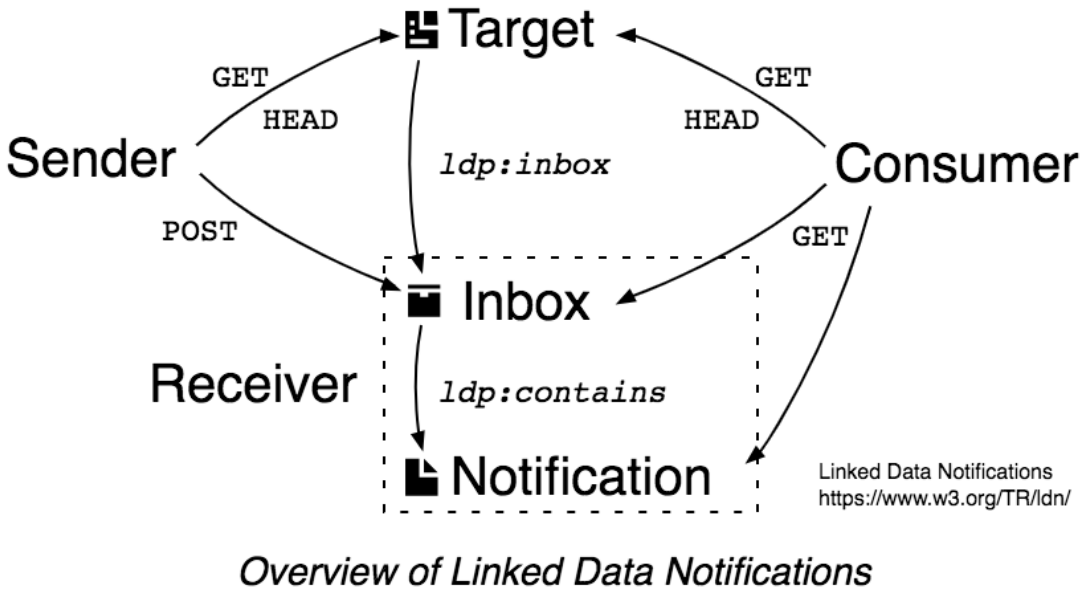

Figure 2-2: LDN at a Glance (sourced from https://www.w3.org/TR/ldn)

most universal communication application known to mankind - to share information with others globally. Linked Data Notifications is perfect for this latter use case.

Linked Data Notifications [27] (LDN) is a protocol that was developed by the Social Web Working Group of the W3C for the seamless sharing and reuse of notifications that are generated by producers in one context and received by consumers in another context. In the specification, notifications are treated like persistent, first-class objects in that they have their own URI and may be fetched for later use in different contexts. The goal of LDN is to democratize access to data that could be useful to a broad audience and encourage productive collaboration on the Web.

The following is a high-level overview of the Linked Data Notifications workflow:

1. A user desires to send a message from its machine (the *Sender*) to a user on another machine (the *Target*)

2. The Sender issues a SPARQL request to the Target to discover its inbox (the *Receiver*), which is defined by the Linked Data Platform (LDP) ontology as *LDP:inbox*

3. The Receiver reveals the data in an access controlled manner to a third party (the *Consumer*)

4. The Consumer discovers the inbox in the same manner in which the Sender does, ultimately processing it as they see fit. This request often involves fetching the content of the inbox via a SPARQL request, which is defined by the LDP ontology as *LDP:contains*

Linked Data Notifications are a core part of SolidVC. There are a number of notifications that are necessary or convenient for the application, including credential request messages, credential issuance messages, and credential sharing messages. In later chapters, it will be made clear just how important Linked Data Notifications are in SolidVC.

2.6 SPARQL

Earlier, I mentioned that one of the benefits of Linked Data is that it enables us to query the Web more efficiently by organizing data and representing relationships in a more meaningful way. This capability seems like an aspirational feature for the Web to provide, as it would enable insight discovery across globally distributed data sources. The SPARQL Protocol and RDF Query Language (SPARQL) was developed in 2008 by the namesake W3C Working Group in order to bring this aspiration to fruition. As the name suggests, the goal of SPARQL is to provide a robust querying language for RDF data on the Web. SPARQL allows users to enforce particular graph patterns that serve as filters while searching the Semantic Web. For those already familiar with Structured Query Language (SQL), it will not require much of a leap in knowledge to understand SPARQL.

The best way to understand SPARQL is with a simple example. Here is a sample data source, query, and output in SPARQL:

Data:

```
@prefix vcard: <http://www.w3.org/2001/vcard-rdf/3.0#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix john: <http://example.com/JohnSmith/profile#> .
@prefix sarah: <http://example.com/SarahJones/profile#> .

john:me a foaf:Person ;
    vcard:FN "John Smith" ;
    vcard:Given "John" ;
    vcard:Family "Smith" .

sarah:me a foaf:Person ;
    vcard:FN "Sarah Jones" ;
    vcard:Given "Sarah" ;
    vcard:Family "Jones" .
```

Figure 2-3: Sample SPARQL Data

Query:

```
SELECT ?y
WHERE
{ ?x <http://www.w3.org/2001/vcard-rdf/3.0#FN> ?y }
```

Figure 2-4: Sample SPARQL Query

Output:

```
-----------------
| y             |
=================
| "John Smith"  |
| "Sarah Jones" |
-----------------
```

Figure 2-5: Sample SPARQL Output

SPARQL is an essential component of most Linked Data applications, as there will almost always be a need to read, write, update, and delete RDF data that

match a particular pattern. In SolidVC, the need for this tool arises during LDN's inbox discovery mechanism, public key posting during setup and discovery during verification, and updating of credential status list during revocation and discovery during verification. These are just the beginnings of SPARQL's utility in SolidVC and Linked Data at large.

2.7 Digital Signatures

We use signatures everyday to associate our identity to documents as a mark of approval or endorsement. In the physical world, we secure the veracity of these signatures by signing documents with financial and legal institutions, which record our signatures for later consultation and verification. While this works for many cases, it is not always effective. For one, there are people who make a living off of accurately forging signatures. In fact, they could likely forge the current reader's signature if given enough time to practice. For this reason, there is also no way of inextricably associating an individual to a signed document, as any professional forger could have generated the signature. Finally, there is no way to detect alteration of the document in transit from the signer to the verifying party, a loophole which could falsely associate the signer to forged claims.

Digital signatures solve all of the aforementioned problems observed with physical signatures. These are mathematically based schemes for ascertaining the authenticity of documents and messages in digital format. The use of the word "authenticity" here refers precisely to the absence of the shortcomings discussed in the previous paragraph. Without delving too deeply into the detail, the typical digital signature scheme involves the following critical operations:

- *generateKeyPair*: Most digital signature schemes employ asymmetric cryptography, which feature a public key, which may be shared with others, and a private key, which must be kept as a secret in order to deliver on the guarantees discussed above. The *generateKeyPair* algorithm is responsible for producing this key pair for users.

- *sign*: Before signing a document, it is important to get it into a uniform format, so that anyone seeking to verify a signature at a later point in time can be sure that they are referring to the same underlying representation of the data. This step is called normalization (or canonicalization). For example, one simple normalization method for a key-value store represented in JSON is to recursively order each key alphabetically and select predefined delimiters for representing key-value mappings and for separating these pairs (so long as these delimiters are forbidden from existing in the input documents). The normalized message is then typically passed into a hashing algorithm, which converts an input sequence of bits to a deterministic output sequence of bits. The output of the hashing algorithm is a digest that is unique to the input bit string in the ideal case. Finally, this digest is encrypted with the private key to produce a signature value that others can verify at a later time.

- *verify*: Whenever a third party wants to verify the authenticity of a signed message, they use the public key to decrypt the signature value and check that it matches the hashed message.

2.7.1 Linked Data Signatures

Linked Data Signatures is a specification that was developed by the Web Payments and Credentials Working Group at the W3C. The specification proposes a mechanism for applying digital signatures to Linked Data documents. As alluded earlier, it is very critical to establish a standard means of normalizing Linked Data documents for signature and verification purposes. This is because there are so many different serializations of RDF (N3, JSON-LD, RDF/XML, etc.) and because the presence of blank nodes [24] complicates this process, since they are named arbitrarily by different applications. However, after years of research into this problem, it has ultimately been solved by various skolemization, leaning, and labelling mechanisms [37] [38].

Most of the important elements of Linked Data Signatures have already been discussed. At this point, the reader knows that a Linked Data document contains a set

of claims, statements, and assertions whose relationship terms are defined by various open ontologies. Additionally, the reader knows that digital signatures enable a strong association between individuals and digital claims through the use of mathematical schemes that normalize, hash, encrypt, and decrypt messages in deterministic ways. Therefore, the only thing missing in the discussion of Linked Data Signatures is the mapping between these two technical domains.

It is important to note that Linked Data is simply data that is structured in a certain format that optimizes for knowledge discovery on the Web. In other words, any operations that can be performed on arbitrary data can theoretically also be applied to Linked Data. The main challenge is in coming up with a uniform representation for this data so that it can be signed with confidence that Verifiers will be able to unequivocally link it to the signer. The Linked Data Signatures specification does not prescribe particular canonicalization, digest, or signature algorithms, but allows for users to attach a *proof* to their signed document, which specifies these algorithms, as well as any necessary parameters and metadata that were used to configure the signature process, including the creator, the signature date, and the plaintext signature value itself, among other other things. Aside from these essential items, signers are free to add any other RDF statement to the document.

Linked Data Signatures is an integral component of SolidVC, as it is used to issue and verify signed credentials in the platform. I used an implementation of Linked Data signatures that is implemented and maintained by Digital Bazaar called *jsonld-signatures* [29] that operates in the JSON-LD space. This is a developing, but robust, tool for Web-based signatures that exposes an API for various signature algorithms (or suites), including RSA, Koblitz, and Ed25519, among others. The *jsonld-signatures* library serves as the engine for a lot of the activity in SolidVC, including credential request, issuance, verification, revocation, and eventually sharing (which currently is an unsigned process). In later chapters, I will explore how exactly these signatures are used and why they are so important.

Chapter 3

Related Work

There exists a large number of certification services that offer a unique set of utilities. In this chapter, I present a broad survey of these credentialing platforms and the pros and cons of each. This is by no means an exhaustive set of systems, but rather a reasonably representative group that demonstrates the different types of approaches that have been implemented in certification systems to date.

3.1 edX

There has been a surge in Massive Open Online Courses (MOOC) [16] over the past 8-10 years with an emerging realization that there is no formal connection between class size and learning outcome [18]. With this developing online ecosystem, many communities have developed unique credentialing mechanisms for its users. Among the more robust platforms in this space is edX. In this system, users can obtain a special certificate for completing a course. This certificate is then hosted on edX for future expression by the Subject and consumption by Verifiers.

As great of a social service as this is, it is missing many of the important elements of a certification service, including self-sufficiency and portability. The Verifier must consult the Issuer of the credential and parse the site-specific credential into a uniform representation, a task that will inevitably be different for each site, considering that each will likely have its own unique way of representing credentials.

3.2 Mozilla Backpack

Prior to Verifiable Credentials, Mozilla had specified a model for representing, signing, and verifying credentials digitally. Conceived in 2011, Open Badges [19] would prove to be the undeniable predecessor to the Verifiable Credentials specification [20]. Perhaps the most popular implementation of Open Badges is Mozilla Backpack, which allows users to collect badges from disparate sources into a single location, like a digital wallet of sorts. This tool has achieved a moderate degree of success throughout its lifetime, enjoying participation from thousands of organizations worldwide and an order of badge instances in the millions [21].

Mozilla Backpack was a major step in the direction of enabling a robust ecosystem of personal, interoperable, and well-defined digital credentials. The problem is that in its current form, it is a centralized system: should Backpack disappear tomorrow, so too would all of your precious credentials.

3.3 uPort

As Distributed Ledger Technology (DLT) continues to penetrate digital services at an alarming rate, they have made a name for themselves in the Identity Management (IdM) realm. In 2016, ConsenSys introduced uPort, a self-sovereign identity mobile platform that is backed by the Ethereum blockchain. Much of its value proposition is in its accommodation for individuals to persist and manage representation of their identity and credentials on a globally consistent storage and runtime in Ethereum. This is a big win for the DLT community, but there are still a number of challenges to address. For one, the degree and complexity of control for the user is overwhelming, as the uPortID recovery protocol can lead to a compromised uPort instance and undetected trustee replacement, which could lead to an irreversibly compromised uPortID. Another crucial problem is the use of a centralized registry to map uPortID to identity attributes, as this public resource could leak otherwise private information about user-provider relationships [17].

3.4 Blockcerts

Continuing the series of work around DLT-enabled certification systems, Blockcerts [9] was developed in 2016 as a Bitcoin-backed credentialing platform that was designed for the "context of academic, professional, and workforce credentialing" [10]. The four main components of the Blockcerts ecosystem are the Issuer, Certificate, Verifier, and Wallet. Like Mozilla Backpack, Blockcerts produces certificates that are compliant with Open Badges, strategically ingratiating itself with a strong ecosystem that is quickly becoming a canonical element of enterprise Information Management Systems (IMS) worldwide.

For all that it has going for it, Blockcerts falls short of an ideal certification tool for at least one critical reason: centralized credential storage during revocation. Since the Issuer stores the credential revocation list, including entire credentials and not just their status, there is an undesirable coupling between credential and Issuer. Maintenance of the revocation list by the Issuer also implies privileged knowledge of existence and use of credentials, rendering them liable to exploitation.

3.5 Hypercerts

Hypercerts was developed as an extension to Blockcerts, specifically to provide a more independent revocation mechanism and a persistent storage solution for credentials. It uses Ethereum smart contracts to manage the status of credentials, a feature that elegantly supports revocation by Issuers, Subjects, and third parties alike. Additionally, Hypercerts leverages InterPlanetary File System (IPFS) as the storage solution in order to produce permanent, content-addressable credentials. Its primary focus is to provide a certification service that overcomes the shortcomings in privacy, interoperability, and self-sufficiency that befall predecessor credentialing systems.

Hypercerts is an amalgamation of bleeding edge technologies that combine to make a robust credentialing system. However, that is also part of the problem: it introduces a lot of complexity that it could have potentially done without. Blockchains and DLTs

Table 3.1: Credential System Comparison

System	Verifiability	Self-Sufficiency	Interoperability	Revocability	Web-Native
edX	✓	×	×	✓	✓
Mozilla Backpack	✓	×	✓	✓	✓
uPort	✓	✓	✓	✓	×
Blockcerts	✓	×	✓	✓	×
Hypercerts	✓	✓	✓	✓	×
SolidVC	✓	✓	✓	✓	✓

are great for achieving credential consistency and persistence, but they bring with them their own baggage. Part of the beauty and the horror of blockchain technology is that there is such a great diversity of them. It's good in that each one offers its own set of benefits in privacy, security, and temporal/spacial performance. It's bad in that there is little to no interoperability between these systems and platform adoption becomes as much of a sociopolitical problem as a technical one. Additionally, there are legitimate concerns about the privacy and scalability of these systems [22], among other things. These concerns do not represent a normative indictment on DLT, as the potential for the technology is almost undeniable. Rather, I believe that for a system as critical as credentialing, it is important to deploy solutions that have been battle-tested on the order of generations.

Chapter 4

System Overview

In this chapter, I present SolidVC [4], a decentralized implementation of the Verifiable Credentials specification, that leverages various ontologies, protocols, and specifications of the Web to deliver a robust and extensible credentialing system.

4.1 Design Principles

SolidVC represents the confluence of a number of core design principles. This section provides an in-depth discussion of these principles and explains how they inform some of the major design decisions on the platform.

4.1.1 Web Interoperability

Many of the design decisions in SolidVC are based on the natural constructions of the Web. As discussed earlier in the Background chapter, I have elected to utilize many of the open Web protocols and specifications that already exist, such as Linked Data Signatures, Linked Data Notifications, SPARQL, and WebID. The primary contribution in SolidVC is its composition of these technologies in a novel way that leverages the access controlled data management environment offered by Solid.

My reasoning for this prioritization of the Web over other platforms is that the Web is a time-tested application that already solves many of the engineering challenges

that SolidVC would otherwise need to solve on its own. The clearest examples of such challenges are name resolution and stakeholder communication. For example, there are instances in which SolidVC accepts from the user the temporary location of a credential and facilitates a series of actions that involve fetching a document for rendering or processing purposes. Similarly, there are many instances in which a user is prompted to enter the ID of an individual for the ultimate goal of communicating with the individual associated with that ID or discovering an attribute about the individual. The Web and its sibling technologies offer each of these core functions out of the box, allowing SolidVC to focus on the other critical aspects of its service.

4.1.2 Identity Flexibility

A natural extension of Section 4.1.1 is SolidVC's use of the natural identity and authentication scheme in the Linked Data context: WebIDs. As alluded previously, much of the interactions on the platform are most naturally performed with WebIDs. However, SolidVC makes accommodations for alternate means of identifying credentials on the platform. This is because the primary use for credential identification is for updating its status and not necessarily locating and fetching it.

In SolidVC, I have a general set of recommendations for identity schemes that are sensible and tractable. One such prescription is that the identity-generating stakeholder (i.e., the Issuer) use IDs that are maintained in an isolated namespace controlled by them, such as URIs that reside in a path that they own. However, this is only a recommendation that I believe would be beneficial to the user and is by no means an enforced requirement.

A final note on identity management is that SolidVC makes no provisions for Issuers to manage credentials. It is left as an exercise for Issuers and other identity-generating stakeholders to decide on a system for mapping users to credentials if necessary. This was a deliberate decision that is informed by the philosophy that personal data tracking should not be a default feature, but rather an intentional process that requires a non-trivial amount of effort to implement. This is in line with Solid's proclaimed mission of enhanced personal data control for users.

4.1.3 Credential Sovereignty

One of the primary goals of SolidVC is to treat credentials as first-class objects. In line with the Verifiable Credentials specification, a credential should contain within it all that is needed to verify its provenance and its dynamic status. There should be minimal reliance on any of the stakeholders producing or consuming the credential. There should also be minimal requirements for persistent location. Particularly, a user should be able to control and migrate their credentials as they see fit, so long as the integrity can be secured via mechanisms such as digital signatures. In Chapter 5, I will discuss a special case for accommodation of persistence for certain types of credentials in which negative historical evaluations are business critical.

4.1.4 General Accessibility

SolidVC is an open source tool that anyone can download and use. There are no restrictions on the stakeholder roles that users can assume on the platform. I have expectations for which profiles will gravitate toward each role, but these are not necessarily enforced by a central SolidVC committee (or anything of the sort) that assigns roles. For example, while it is not difficult to imagine that large institutions, such as governments, universities, and banks would experience the greatest benefit in being an Issuer, there is nothing preventing a citizen from being an Issuer for their own personal causes and ventures. Additionally, there is nothing preventing users from assuming multiple stakeholder roles for different purposes, as SolidVC exposes a seamless mechanism for switching roles. For an example of where this multiplicity of roles might be useful, consider the following scenario: Employee A, who has collaborated with Employee B on multiple projects in the past, would like to vouch for them to enable future employment opportunities for the latter. While Employee A owns their own personal credentials, such as an academic diploma and a driving license, they also have the desire to deliver credentials to others.

4.2 Stakeholders

There are three main stakeholders in SolidVC: *Subject*, *Issuer*, and *Verifier*. These are no different than those specified in the Verifiable Credentials specification [6]. The reader should note that SolidVC makes no distinction between Subject and Holder, choosing to adopt the former terminology throughout the system terminology. Additionally, SolidVC adopts Verifier as the term for the stakeholder that inspects the validity of a credential, alternatively known as Relying Party in traditional certification ecosystems, and more recently known as Inspector in the Verifiable Credentials specification. In Section 4.4, I will specify the behavior of each of these stakeholders.

4.3 Ontology

The fuel of any credentialing system is the verifiable credential that proves skills ownership and enables service access. In SolidVC, I represent credentials with a custom ontology called *svc* [5]. This ontology encapsulates metadata such as the credential type (i.e., Education, Health, Finance, etc.), the embedded claims, the Subject, the Issuer, and a proof. This ontology standardizes credentials on the platform.

The main resource types in the *svc* ontology are *Credential* and *CredentialStatusList*. *Credential* represents the self-sufficient credentials that are requested, issued, shared, and verified in SolidVC. Some of the key properties of this resource are *id*, *domain*, *issuerId*, and *subjectId*. Meanwhile, the *CredentialStatusList* resource type represents the evolving status of a credential and includes properties such as *credentialStatus*, *credentialId*, *revocationReason*, and *revocationDate*.

In the *svc* ontology, it is clear that there are parallels between some of the properties I have defined and some defined in other open ontologies. For these properties, my decision to include them nevertheless is due either to a slight nuance in meaning or to a perceived necessity to outline all of the integral credentialing properties on the platform, for the sake of future users and students of *svc*.

Figure 4-1: Sample Credential

4.4 Protocols

SolidVC consists of a number of well-defined protocols for managing credentials. In this section, I will outline these protocols in detail.

4.4.1 Setup

In order to use SolidVC, there are a number of dependencies that the user must install in their local environment (henceforth, *svcLocal*) and in a public SolidVC-provisioned folder that resides in the user's Solid pod (henceforth, *svcRemote*). This process requires that the user has a Solid account, complete with a WebID and pod (See [2] and [3] for specification of a Solid-compliant pod). I have provided a script (henceforth, *svcSetup*) that checks and establishes that the aforementioned prerequisites and constraints are maintained in *svcLocal* and *svcRemote*.

Before anything else, *svcSetup* downloads a number of software dependencies that SolidVC needs to operate properly. After this process, the user is prompted to provide the credentials of a Solid account in their control. Users are encouraged to have a Solid

account beforehand, as SolidVC depends heavily on the ownership of this account. With this information, *svcSetup* authenticates the user to their account in preparation of privileged access.

After *svcSetup* authenticates the user to their account, it proceeds to generate and store a cryptographic key pair, which will be used for signing and verifying credentials on the platform. With this preliminary data, SolidVC is poised to perform its first privileged access of the user's account: publication of the user's public key to *svcRemote*.

Next, *svcSetup* prompts the user to provide the desirable location for their public key in *svcRemote*. This takes the form of an existing, public folder, which often resides in the user's pod, but not necessarily. The public key is then posted to this location. However, this public key is only as useful as it is discoverable. This is where Linked Data makes its first appearance in SolidVC. Particularly, *svcSetup* updates the user's Solid profile document to point to their public key. Now, when future stakeholders need to access a user's public key - as they will during the Verification protocol in Section 4.4.5 - they can easily find it by referring to the user's profile.

At this point, most of the setup for SolidVC is complete. The only thing that remains is setting up the remote credential status folder. As with the public key folder, *svcSetup* prompts the user for an existing, public folder that will store metadata about credentials, such as expiry and revocation. I will discuss this folder in detail in Section 4.4.6.

By the end of the Setup protocol, *svcSetup* has output a convenient configuration file that stores information such as the user's Solid WebID, the location of their public key folder, and the location of their credential status folder. This information is partially for the user's sake (i.e., in the case of WebID) and partially for SolidVC's sake (i.e., in the case of the public key folder and the credential status folder).

4.4.2 Request

To kickoff the credentialing process, the Subject may optionally request the Issuer for a credential. This process is not included in the Verifiable Credentials specification;

the Subject and Issuer may communicate in an offline communication channel or the Issuer may already know to issue the credential, depending on the context. Nevertheless, I have decided to include an interface for the Subject to explicitly request a credential from the Issuer via Linked Data Notifications as a convenience to the stakeholders involved.

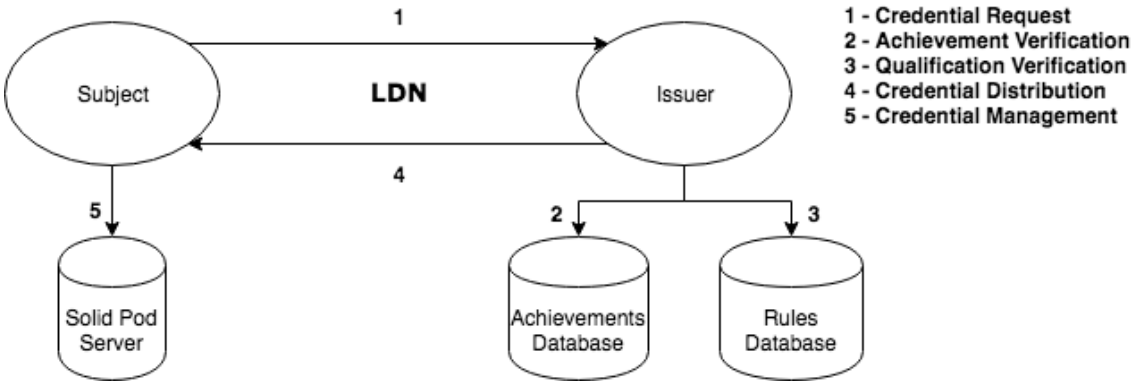

Figure 4-2: The Request and Issuance Protocols

A credential request is a lot like a credential. For one, it includes a series of metadata about the credential, such as the domain, title, description, and Issuer ID. In essence, it is a credential specifying the desired credential and, in fact, a credential request is represented by the same resource type as a credential proper in the *svc* ontology. Another important similarity between requests and credentials is that they are signed by the creator.

In Solid, there is no notion of filtering spam or otherwise suspicious messages in the inbox. Everything that is sent to a user is processed and displayed the same way. With Linked Data Signatures, this could at least be partially addressed by requiring that each message is signed by the sender. This is precisely the utility that the Request protocol provides. When a Subject requests for a credential, the application automatically signs the request. With this provision, a user cannot request for a credential on behalf of another user without detection, which reduces the potential for the type of Denial of Service (DoS) attack that involves launching credential

requests as a proxy for an unsuspecting victim from multiple different Issuers.

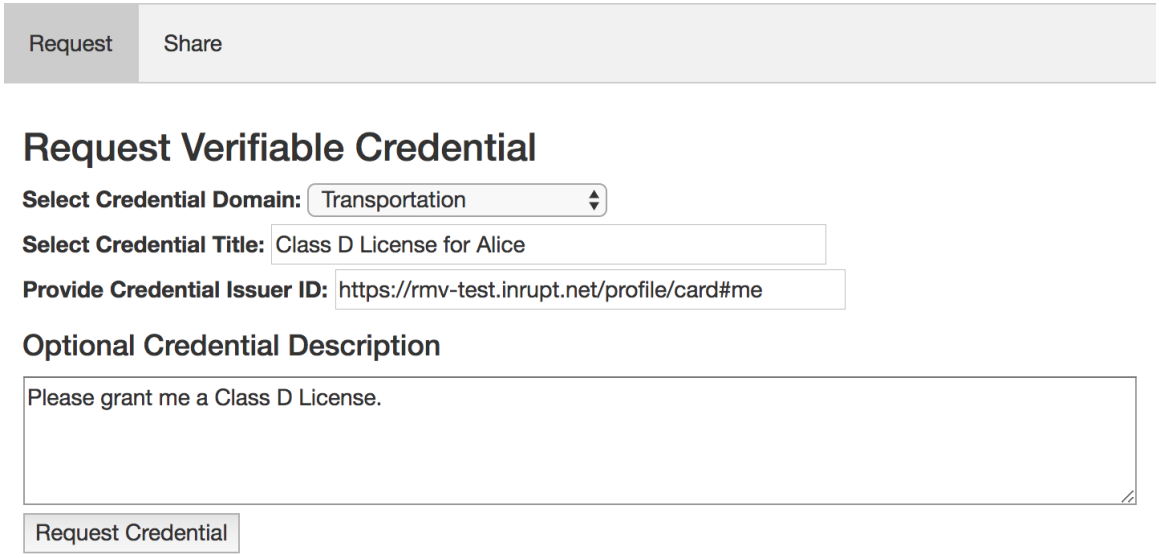

Figure 4-3: The Request Interface

Because a credential requires the verifiable completion of a set of achievements, the Subject is encouraged to provide a minimal set of data for identity and capability resolution. Figure 4-3 illustrates the Subject's perspective on the platform during a typical request. Additionally, Section 4.6 will explain in greater detail what this process entails.

4.4.3 Issuance

An Issuer can directly create, sign, and issue a credential to a knowingly deserving Subject. This interface is useful if the Issuer already has the proof and user information that they need to reward a Subject with a credential and, consequently, a credential request would be superfluous. As alluded earlier, the issuance of a credential is a lot like the request for one, as they each involve specification of claims about a resource, where the resource in this case is the Subject, as opposed to the credential in the latter case.

The Issuance protocol features, underneath the hood, the first instance of Linked

Data Signatures. To implement the digital signature infrastructure, I utilized *jsonld-signatures* [29], as discussed in Chapter 2.
The Issuance protocol consists of the following detailed steps:

1. Parse N3-serialized credential into *rdflib* [25] quad store.
2. Add credential metadata (i.e., *id*, *domain*, and *credentialStatus*) statements to quad store.
3. Serialize credential quad store into JSON-LD in preparation for *jsonld-signatures* activity.
4. Load key pair from *svcLocal*.
5. Sign credential with key pair.
6. Submit signed credential to Subject of interest.
7. Instantiate new *rdflib* quad store that will contain credential status metadata.
8. Add credential status metadata (i.e., *credentialId* and *credentialStatus*) to quad store. Note that the Issuer has complete freedom over the choice of credential ID, as there is no expectation that this ID be dereferenceable on the Web, considering the sovereignty of credentials on the platform. I will revisit this idea later.
9. Serialize credential status quad store into N3.
10. Submit new credential status document to credential status folder in *svcRemote*.

From the user perspective, the issuance process is relatively simple. It involves entering the domain (i.e., Education, Health, Finance, etc.) of the credential, the ID of the Subject, and the credential in plaintext (See Figure 4-4 for a visual illustration of the Issuer's view during this process). Currently, the toughest aspect of the issuance process may be producing the plaintext credential depending on the user's familiarity and proficiency with RDF, as it involves entering a credential in the form of RDF

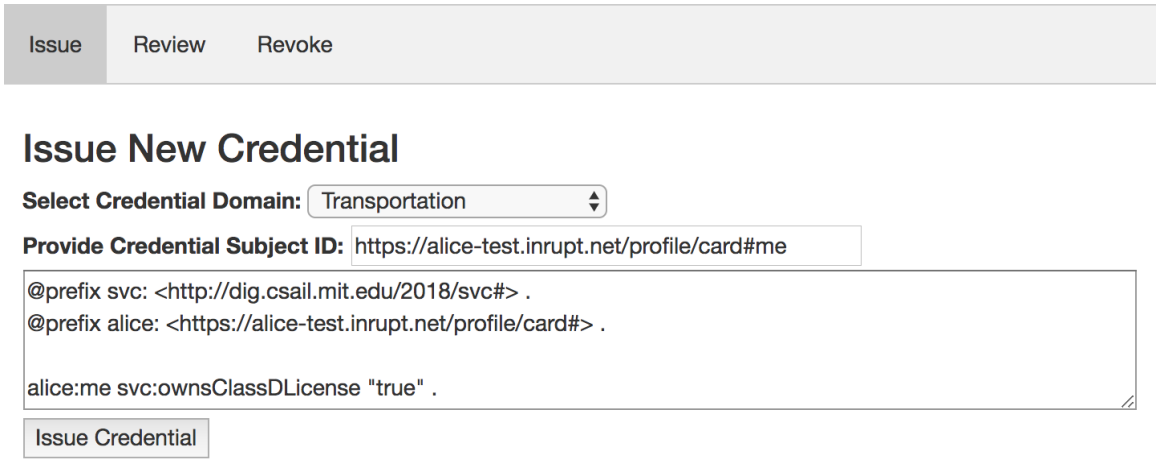

Figure 4-4: The Issuance Interface

statements. At the time of this writing, the platform only accepts data in the N3 RDF serialization format, but there are already steps being made to accommodate other formats, including JSON-LD and N-Quads. In fact, there are also possible accommodations that can be made for human readable claims, which could simply dump the statement(s) into the value field of a single RDF statement with a special predicate designated by *svc*. Finally, there will likely be provisions for submitting credentials via file upload.

Another note on RDF serialization in SolidVC: although credentials are accepted from Issuers in N3, they are stored in the JSON-LD format and in the *text/plain* Multipurpose Internet Mail Extensions (MIME) type. The reasoning behind this roundabout means of processing and storing credentials has to do with a seeming oversight in *rdflib*, in which the library parses documents signed by *jsonld-signatures* in a shortcut "syntactic sugar" kind of N3 format, but is not able to parse these credentials back into an *rdflib* quad store; in a sense, *rdflib* is writing checks it can't cash in this edge case. Additionally, Tabulator (Solid's visual interface) has limitations that does not yet allow users to view documents in the *application/json+ld* MIME type and does not even allow the storage of documents in the *application/json* MIME type (let alone viewing); hence, I am intermittently storing credentials in the *text/plain* MIME type. So long as the credential can be serialized into JSON-LD,

this proves not to be a problem at rendering time. In any event, I plan on working with the Solid and *rdflib* developer communities to address some of these issues so that credentials can be processed and stored in any format.

4.4.4 Sharing

Sharing is another relatively simple process. There are two ways to share credentials with stakeholders:

1. **File Upload:** In this mode, the Subject wielding control over the credential can upload a local credential-bearing file from their computer. This method illustrates the Credential Sovereignty philosophy, as discussed in Section 4.1.

2. **URI Provision:** In this mode, the Subject wielding control over the credential provides the URI of the credential and the ID of the stakeholder, often a Verifier. This triggers a fetch of the credential from the URI and an LDN-supported communication of the content. To reiterate, the URI itself is not shared with the stakeholder, since this is an unreliable attribute of the credential in SolidVC. Rather, the actual content of the credential is shared.

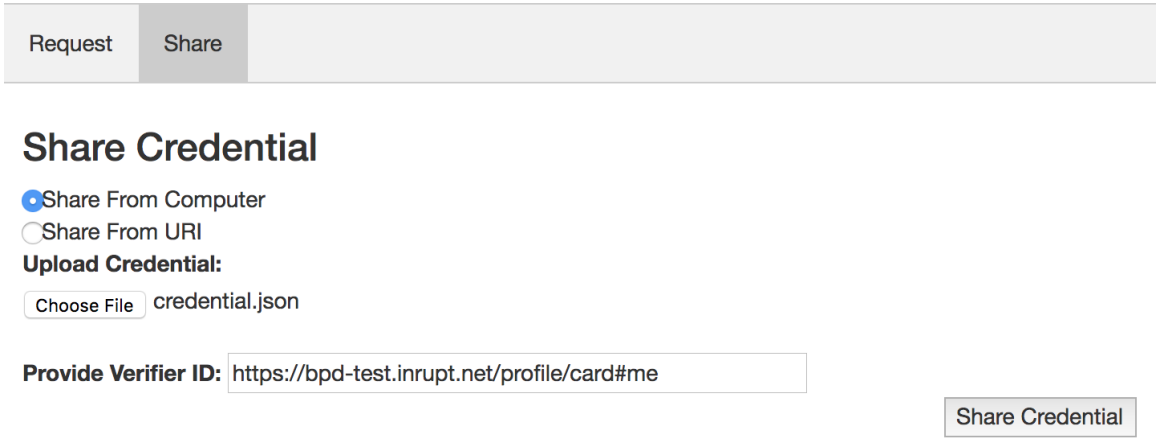

Figure 4-5: The Sharing Interface

Figure 4-5 illustrates visually the Subject's experience in SolidVC during a typical

request. One desirable feature addition of the Sharing protocol would allow users to sign shared credentials and specify metadata about the usage of credentials, such as expiry and other useful conditions. I will discuss this in greater detail in Chapter 5.

4.4.5 Verification

A Verifiable Credentials management system is only as useful as its ability to verify signed credentials. For this reason, I have provided a rather trivial means for verifying credentials. The user simply needs to enter the temporary URL of the credential to verify its provenance. The Verification protocol involves a number of key steps:

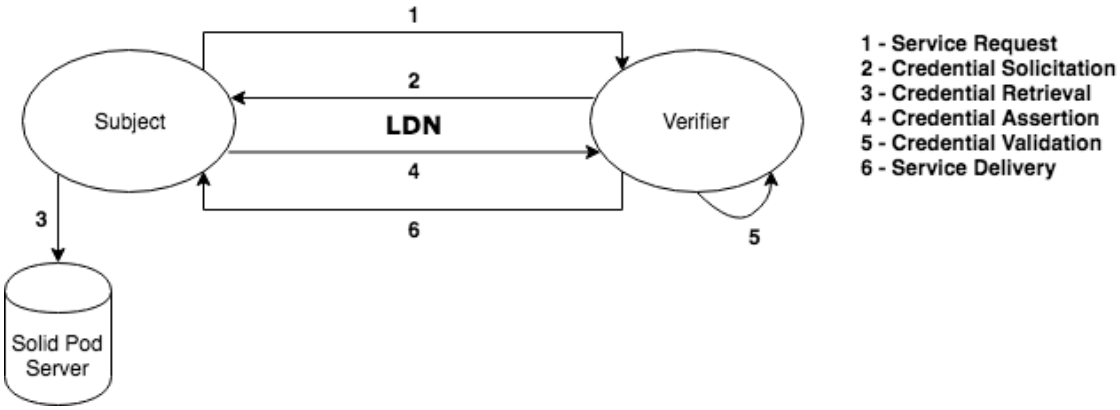

Figure 4-6: The Verification Protocol

1. Fetch credential from user-provided URL.
2. Search credential for Issuer ID.
3. Discover Issuer public key from Issuer ID via LDN.
4. Search credential for URL of its status document.
5. Fetch credential status (i.e., **ACTIVE**, **EXPIRED**, **REVOKED**, etc.) and related metadata, (i.e., *expiryDate*, *revocationReason*, etc.) from remote credential status folder established during *svcSetup*.

6. Search credential for *jsonld-signatures* proof.

7. Use combination of public key and proof to verify that the nominal Issuer was indeed the Issuer of the credential. Note that SolidVC (and all other Verifiable Credentials platforms) does not have a mechanism for preventing fabrication of credentials. In other words, a credential is only as valuable as one's trust in the Issuer. This is a social problem that is yet unsolved, and consequently out of scope of SolidVC, as it requires insight into the heart and intent of the Issuer.

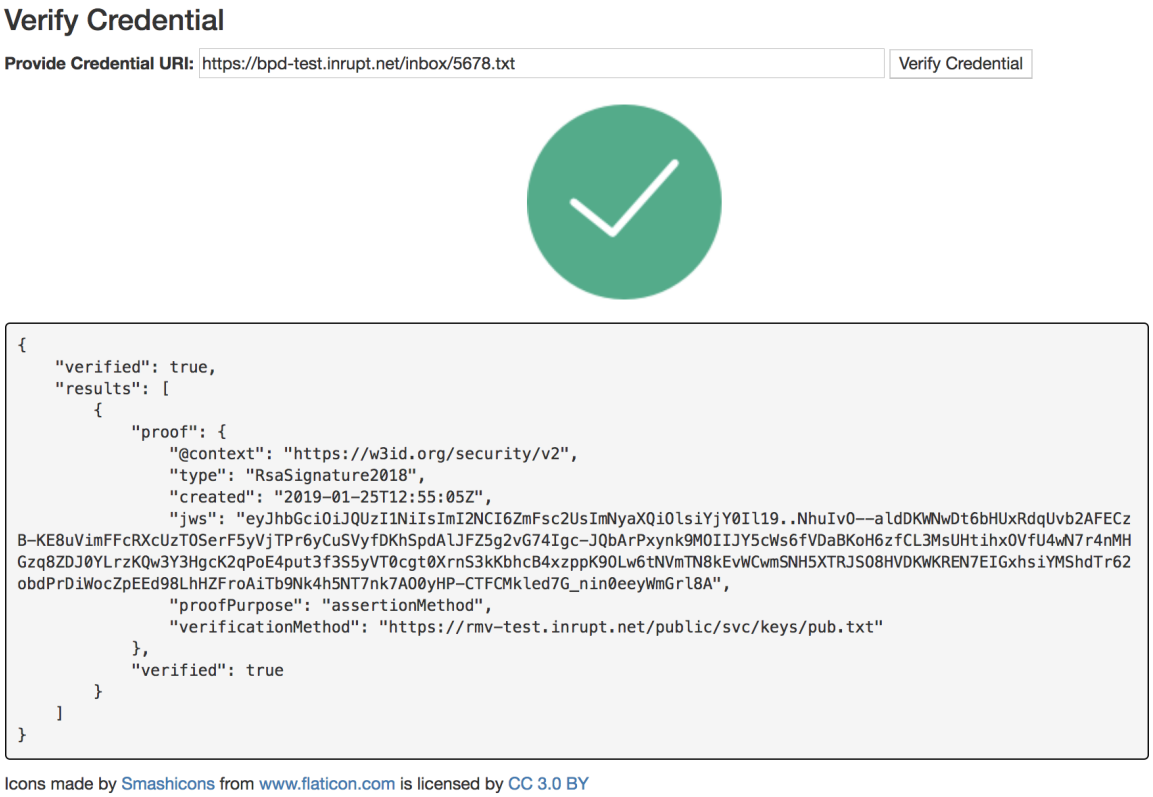

Figure 4-7: The Verification Interface Upon Success

Realistically, credential verification would typically happen in the background as an application or service consumes a credential. Nevertheless, I have provided in SolidVC this utility interface for verifying credentials as a sanity check. To bring this work full circle, I envision an emerging ecosystem of applications operating in their own context and consuming credentials whenever it is necessary to permit access to

restricted services. Refer to Figure 4-7 for a depiction of the Verifier's experience upon verification of a valid credential. Additionally, Figure 4-11 exposes an example of when verification exposes an invalid credential.

4.4.6 Revocation

A truly robust certification service should provide a clean mechanism for revoking credentials. The Verifiable Credentials specification prescribes the use of the *credentialStatus* property to augment the credential with useful metadata [6]. This field points to another credential that describes the state of the original credential, including revocation information, if necessary. An interesting property of this design is that it is effectively using a credential to describe another credential, similar to credential requests. In SolidVC, the user is simply prompted to enter the ID of the credential and the reason for revocation. Figure 4-8 presents the modest interface that users engage during revocation.

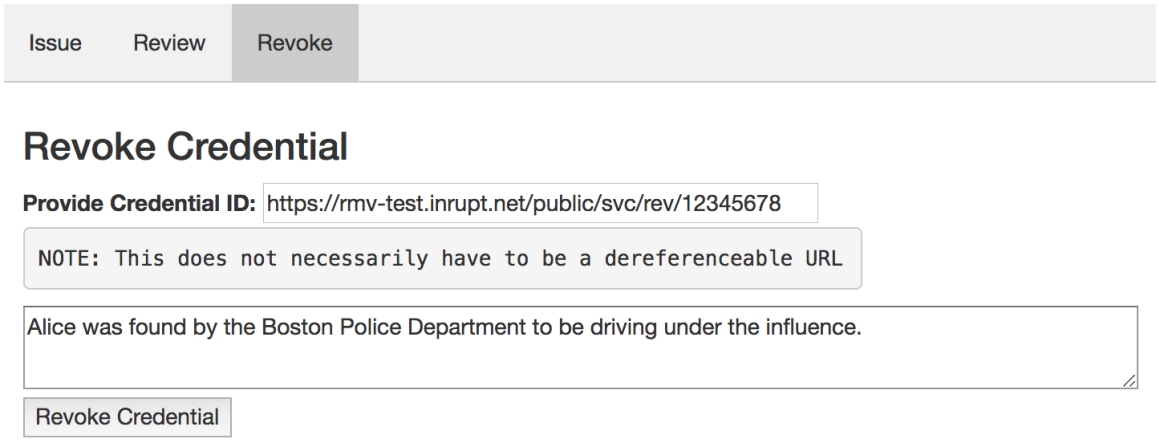

Figure 4-8: The Revocation Interface

The following is a comprehensive outline of the technical steps involved in the Revocation protocol given minimal user input:

1. Fetch local credential status folder into local *rdflib* quad store.

2. Search for credential status document with credential ID provided by user.

3. If credential ID is not present in the user's credential status folder, return and report to user. Otherwise, prepare to update the relevant credential status document with revocation metadata.

4. Load revocation metadata into a local *rdflib* quad store. Relevant metadata include reason and date of revocation, as well as the **REVOKED** state among other things. In a separate quad store, load RDF statements that may be living in the credential status document, but is incongruent with a revoked credential. Such data includes the previous state (i.e., **ACTIVE** or **EXPIRED**) or other metadata associated with that state

5. Submit a SPARQL PATCH request including these insertions and deletions to the credential status.

6. Report whether credential status update was successfully recorded.

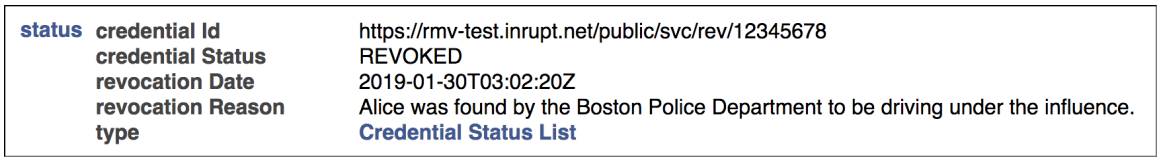

Figure 4-9: Tabulator View of **REVOKED** Credential Status

An important note to make about revocation that has been mentioned in various ways but cannot be overstated is that the user input is a unique ID, but not necessarily a URL. This is the ID of the credential and is stored in the credential status document simply as a means of referring to the credential in a user's credential status folder for purposes of update. It is also important to note at this point that while this design was implemented for credential-location independence, there is still a need for the credential status documents to be location-dependent, since the credentials refer to these. For this reason, the credential status document is stored as a URL in the credential object.

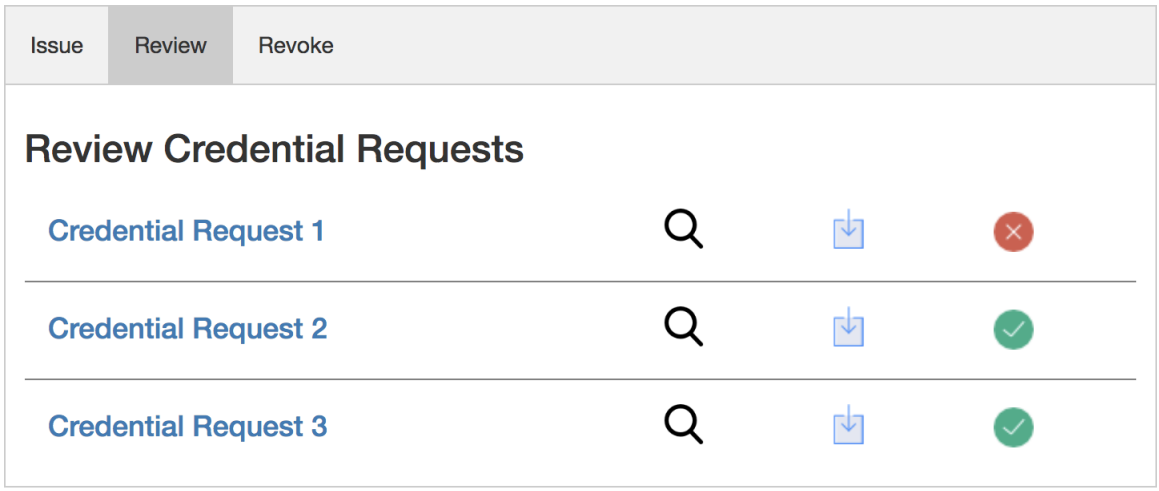

Figure 4-10: The Review Interface

4.5 Additional Features

4.5.1 Credential Review

In the Issuer view, a user can review their outstanding requests and decide whether to approve or reject depending on the perceived credibility of data that was provided by the Subject. In the event of a rejection, the Issuer is encouraged to include a useful feedback message that the Subject can use to improve their chances of approval in the future. Upon approval, the document is normalized and signed with the Issuer's cryptographic key pair. The document is finally delivered back to the deserving Subject, once again via Linked Data Notifications. The Subject now owns that credential in their access controlled Solid pod and is free to manage visibility as they please. [1]

Another important aspect of this view is the verifiability of credential requests. In particular, next to each message is a thumbnail image of a check mark or an ex mark, indicating whether the credential was signed by the proclaimed subject. This is important for filtering out spam-like requests. See Figure 4-10 for an illustration of this.

[1] **Note:** The Issuer can ultimately reject a request for any reason, including questionable public history and suspicious user identity.

4.5.2 Credential Download

To emphasize that credentials are self-sufficient and location-independent, SolidVC allows users to download credentials from their inbox onto their personal computer. With a local copy of credentials, users have the flexibility to migrate their credentials from Solid/SolidVC to another platform and to share their credentials with stakeholders outside of the platform.

4.6 Use Case: Driving License

In this section, I provide a discussion of a typical use case of SolidVC in the context of motor vehicle certification. The point of the ensuing scenario is to incite the imagination of readers for the kind of credentialing activity that SolidVC supports. [2]

4.6.1 Setup

Alice is interested in receiving a driving license in the State of Massachusetts. Before she can engage in this transaction with the Massachusetts Registry of Motor Vehicles (RMV), she first downloads SolidVC and executes *svcSetup*. Assume that the RMV has already downloaded SolidVC and executed *svcSetup*. Fortunately for users, it is necessary to run *svcSetup* only once. The output of this single run enables arbitrarily many interactions with future stakeholders.

4.6.2 Request

First, Alice visits the SolidVC Subject interface. Next, she enters the WebID of the RMV, a SolidVC-compliant Issuer, and selects the *Transportation* option. She proceeds to enter the relevant request metadata, including the WebID of an RMV employee named Chris, and an optional title and description for the request.

[2] Many of the figures and claims in this section, such as personal names, institutional policies, and organizational affiliations with SolidVC are fictional, but are referenced henceforth as real for demonstrative purposes alone

In future versions of SolidVC, the entry of metadata such as the *Transportation* credential type might lead to a series of other user prompts, such as the Country/State and type of driving license of interest. Ultimately, the output of this process may be a credential request template for Alice to complete or a useful Web link informing her of required documents to upload and include in the request [3]. See Chapter 5 for a discussion on how to improve the request process, including Issuer discovery.

4.6.3 Issuance

When Chris visits the SolidVC Issuer interface, he finds Alice's request waiting for him in his inbox. Chris reviews Alice's request and the attached information and immediately returns a rejection notice, explaining that Alice must complete a month-long driving course and take an hour-long driving test in person. Alice uses the linked resources in the rejection notice to apply for the driving course and independently prepare for the driving test. After a month, Alice has completed the course and taken the test and Chris assesses that she is certified to drive in the State of Massachusetts. Because he already has some of her information in the system from their previous correspondence, he can proceed directly to the *Issue* panel of the Issuer interface to create, sign, and return a new motor vehicle credential to Alice.

4.6.4 Verification/Revocation

Elated about her new license, Alice sneaks out in the evening with her Mother's car to celebrate with friends. By the end of the night, she had been intoxicated to the point of motor deficiency, but was confident that she could safely return her friends to their destinations. However, within moments on the road, Alice was stopped by the Boston Police Department for questioning, suspecting that she had been driving under the influence. After a brief assessment which involved ascertaining the validity of her license via the SolidVC Verifier interface, the cops reported Alice's poor behavior to Chris from the RMV, who subsequently decided to revoke the new license using

[3] https://www.mass.gov/passenger-class-d-drivers-licenses

SolidVC's revocation functionality. The result is a **REVOKED** status that would appear in the credential status document associated with Alice's license (See Figure 4-9 for reference). In the future, there is the possibility that SolidVC could incorporate a reporting mechanism that can be used to report adverse behavior of Subjects to Issuers for the sake of encouraging disciplinary action, such as revocation. This functionality could very likely be subsumed by the Request interface.

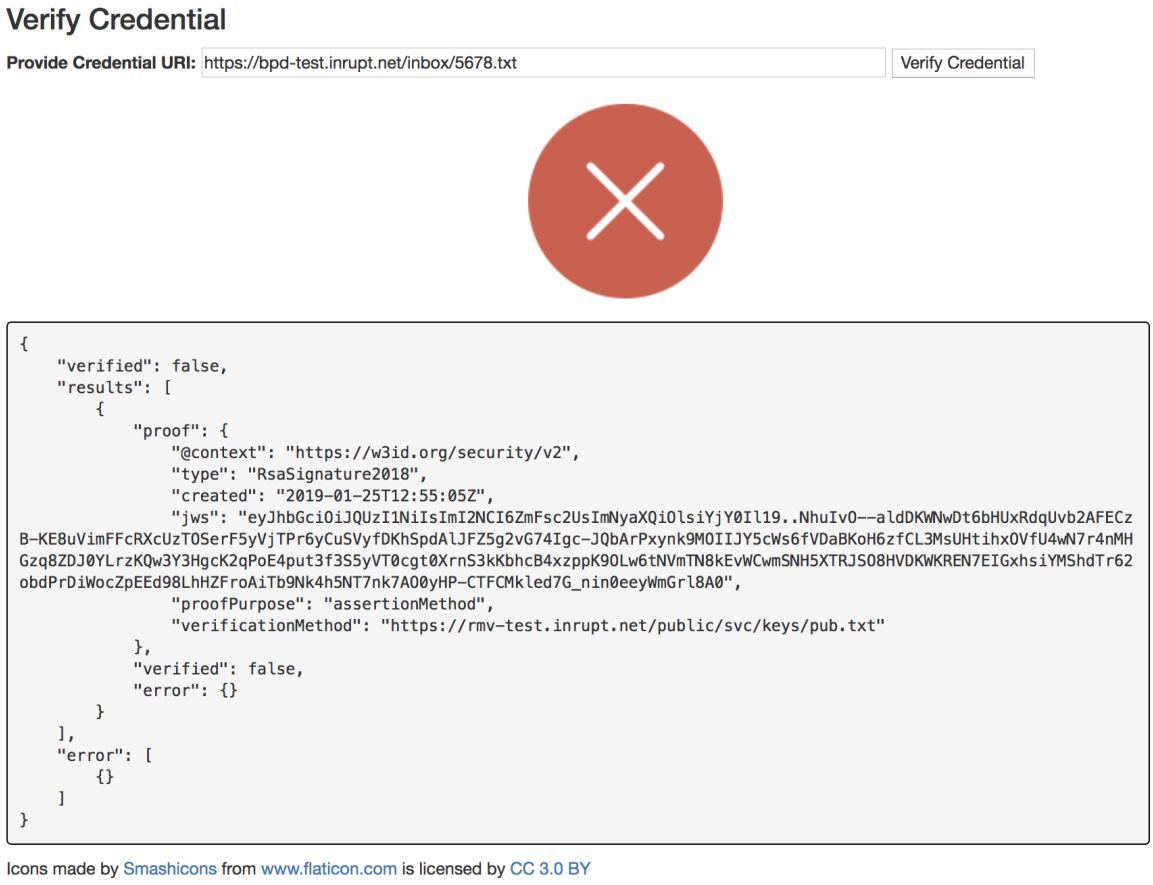

Figure 4-11: The Verification Interface Upon Failure

Chapter 5

Future Work

There are many conceivable ways in which SolidVC can be extended to provide a more robust certification service. The following is a discussion of the major improvements that would bring SolidVC closer to its full potential. Many of the items in this chapter are already underway and may already be implemented by the time of this paper's publication. Nevertheless, I find it necessary to clarify them as work items in progress.

5.1 Command Line Tool

As constructed, SolidVC is great for non-technical users that wish to issue and receive verifiable credentials. However, I believe that a more technical crowd would appreciate a command line tool that would allow for them to interact with the system from the comfort of their favorite command-line interface. With such a tool, Subjects would be able to request credentials and review their credential repository and Issuers would be able to review and service credential requests all from a tool like Terminal on Unix-based systems or Command Prompt on Windows-based systems.

An additional use for a command line tool is the support of rule-based credentialing systems. In many use cases, it is best for human beings to issue credentials if the policy checking and achievement resolution involves a series of human interactions, such as third-party reference checking and other such forms of due diligence. However, for other credentials, issuance requires checking adherence to a set of encoded rules.

So long as the credential request can express the attributes of interest in a machine-readable manner, automated credential issuance would be a natural extension and economic convenience for many extant credentialing systems.

5.2 Extended RDF Serialization Support

As mentioned in Section 4.4.3, SolidVC only supports N3 in its current form. However, there are many other popular RDF serializations, such as JSON-LD and N-Quads. As such, I am planning to add support for these and possibly other additional formats as the relevant developer communities iron out some of the wrinkles in the supporting technologies, per an earlier discussion.

5.3 Expiry Support

Previous sections outlined some of the accepted values in the range for *credentialStatus*. One such value was **EXPIRED**. This status is important in the credentialing ecosystem because it hints at the potential for skills to wane over time. For this reason, SolidVC ought to have a robust mechanism for enforcing expiry. For many of the possible values for *credentialStatus*, it makes sense for a human being to explicitly apply the new status. The challenge with expiry is that while specification of the date is rather simple (just another statement in the credential), application of the new state in the credential status document is not as simple. Therefore, in order to enforce the invariant that the status of the credential resides reliably in the credential status document, revisions can be made to add an *expires* field that stakeholders can use to express the expiry date in the input credential. During issuance, the SolidVC Issuance protocol would search for this field in the credential and add it to the credential status document if it exists. Later, during verification, SolidVC would check this field, compare it to the current time, and report to the interested party if the associated credential has expired. Alternatively, SolidVC could include a cron job that periodically searches through the credential status folder and updates the status

of the relevant document to be **EXPIRED** if necessary.

In addition to credential expiry, there ought to also be a mechanism for verification expiry for shared credentials. In other words, a user should be able to specify in the credential sharing process that the verifier can inspect it for only a restricted time frame, after which access would be curbed. This kind of restriction requires additional thought to enforce. Upon embarking on a thought experiment similar to that which explores issuance-time credential expiry specification, per the previous paragraph, one would find that it is no problem to tag a credential with the *expires* field during credential sharing, but that it is another enigma altogether to enforce that the verifier can no longer access the credential, since it is a self-sufficient entity that is not managed by a central authority and, hence, can be inspected to the extent desired once received. One idea to combat unfettered access is to encourage users during sharing to migrate extra-sensitive credentials to a folder that only the Subject can write and update. Expiry would then be enforced via manual or automatic deletion at the appropriate time and reported to the verifier via a "404 Not Found" error from subsequent reads.

5.4 SolidVC Extensibility

In Section 4.6, I discussed a very specific use case for SolidVC in the context of managing motor vehicle credentials. However, there are so many other types of credentials that could potentially be supported by SolidVC. These include academic credentials, citizenship credentials, and even professional credentials, such as the certification to practice medicine or law. A case could even be made for temporary credentials that require periodic certification examinations for renewal.

The key point of uncertainty with extended support is whether SolidVC should offer it natively or if the better approach is to allow the emergence of an ecosystem of apps and services that will port with SolidVC, almost like what Blockstack [31] and even Solid have managed to do. I can conceive potential pros and cons with each approach and believe that this design decision should be made with careful

consideration thereof.

5.5 Credential Persistence

Decentralization is both a friend and an enemy of SolidVC. On the one hand, it allows for any entity to deliver a credential to any other entity and for the receiving entity to wield ultimate control over that credential. On the other hand, it may give the Subject too much control. Consider a more complex credential that records credit scores. If a Subject receives a poor credit score, they may simply decide to delete it or the Subject may even prevent an Issuer from issuing credentials on their behalf. This is an example of a use case where credential persistence is perhaps just as important as credential control, as it is more critical for a service such as a Real Estate agency to have this information than it is for the aspiring tenant to be able to hide it.

This is an open area of research, but some preliminary approaches that come to mind include the VeresOne Method [33] and the WebLedger Protocol [34], which leverage Decentralized Identifiers (DID) [32], among other relevant technologies. Should SolidVC choose to adopt some of these technologies, the preferred application domain would be toward the persistence of the credential status documents. Ideally, the credential should remain location-independent. With this modification, the risk that Issuers currently pose on the platform, in the form of a lost or damaged credential status document, is significantly reduced.

5.6 One-to-Many SolidVC-Solid Account Mapping

Currently, there is a one-to-one mapping between SolidVC accounts and Solid accounts. I would like to expand this to a one-to-many mapping, so that users can enjoy the utilities of SolidVC from any and all of their Solid accounts. My initial vision is to enable account switching, such that the application experience is modeled as a context switch. However, the long-term vision is for SolidVC to behave like a mashup [36], managing and displaying in a single interface credential informa-

tion sourced from multiple Solid accounts. With these changes, SolidVC could truly become a one-stop shop for all things credentials on the Web.

5.7 Issuer Discovery

At the moment, there is no explicit means for discovering SolidVC-compliant Issuers on the platform. When a Subject wants to request a credential, they are just expected to know the WebID of an Issuer that leverages SolidVC to deliver credentials, let alone of the kind in which the Subject is interested. More research should be invested in discovering Issuers in a reliable and efficient way.

Chapter 6

Conclusion

SolidVC is a decentralized, Web-based, Verifiable Credentials framework that prioritizes democratized access and personal credential control. The history of credentials on the Web is a storied one that has witnessed a fair share of political and capitalistic manipulation, technical mismanagement, and outright corruption. With a careful and minimalist Web-native approach, such as SolidVC, I believe that the fate of Web credentials is in good hands.

Bibliography

[1] Berners-Lee, Tim; *Weaving the Web: The Original Design and Ultimate Destiny of the World Wide Web by its Inventor*; Harper 1999

[2] Mansour, Essam; Sambra, Andrei Vlad; Hawke, Sandro; Zereba, Maged; Capadisli, Sarven; Ghanem, Abdurrahman; Aboulnaga, Ashraf; Berners-Lee, Tim; *A Demonstration of the Solid Platform for Social Web Applications*; WWW '16 Companion Proceedings of the 25th International Conference Companion on World Wide Web (WWW) 2016.

[3] *Solid Homepage*; `https://solid.inrupt.com`

[4] Ezike, Kayode; *Solid Verifiable Credentials*; `https://github.com/kezike/solid-vc`

[5] Ezike, Kayode; *Solid Verifiable Credentials Ontology*; `http://dig.csail.mit.edu/2018/svc`

[6] Sporny, Manu; Longley, Dave; *Verifiable Credentials Data Model*; `https://w3c.github.io/vc-data-model`

[7] Lee, Sunny; Otto, Nate; *Verifiable Credentials Use Cases*; `https://w3c.github.io/vc-use-cases`

[8] *uPort: Open Identity System for the Decentralized Web*; `https://www.uport.me`

[9] *Blockcerts: The Open Standard for Blockchain Credentials*; `https://www.blockcerts.org`

[10] Schmidt, Philipp; *Blockcerts: An Open Infrastructure for Academic Credentials on the Blockchain*; https://medium.com/mit-media-lab/blockcerts-an-open-infrastructure-for-academic-credentials-on-the-blockchain-899a

[11] Santos, João; *Hypercerts: A Non-Siloed Blockchain-Based Certification Service*; https://github.com/inesc-id/dclaims-pm/blob/master/thesis-project-doc/Hypercerts_project.pdf

[12] Friedman, Uri; *Trust Is Collapsing in America*; https://www.theatlantic.com/international/archive/2018/01/trust-trump-america-world/550964

[13] Ahmad, Wajeeha; *Dealing with Fake News: Policy and Technical Measures*; https://internetpolicy.mit.edu/wp-content/uploads/2018/04/Fake-news-recommendations-Wajeeha-MITs-IPRI.pdf

[14] McWaters, Jesse; *A Blueprint for Digital Identity: The Role of Financial Institutions in Building Digital Identity*; http://www3.weforum.org/docs/WEF_A_Blueprint_for_Digital_Identity.pdf

[15] *ID2020*; https://id2020.org

[16] *Massive Open Online Course*; https://en.wikipedia.org/wiki/Massive_open_online_course

[17] Dunphy, Paul; Petitcolas, Fabien A. P.; *A First Look at Identity Management Schemes on the Blockchain*

[18] Lederman, Doug; *Does Class Size Matter?*; https://www.insidehighered.com/views/2007/12/06/does-class-size-matter

[19] *Open Badges*; https://openbadges.org

[20] Appelcline, Shannon; *Open Badges are Verifiable Credentials*; https://github.com/WebOfTrustInfo/rebooting-the-web-of-trust-spring2018/blob/master/final-documents/open-badges-are-verifiable-credentials.md

[21] Surman, Mark; *An Update on Badges and Backpack*; https://medium.com/read-write-participate/an-update-on-badges-and-backpack-5a06fab252ea

[22] Blenkinsop, Connor; *Blockchain's Scaling Problem, Explained*; https://cointelegraph.com/explained/blockchains-scaling-problem-explained

[23] *Resource Description Framework*; https://www.w3.org/RDF

[24] *RDF 1.1 Concepts and Abstract Syntax*; https://www.w3.org/TR/rdf11-concepts

[25] *Linked Data API for JavaScript*; Read-Write Linked Data Organization; https://github.com/linkeddata/rdflib.js

[26] *WebID*; https://www.w3.org/wiki/WebID

[27] Capadisli, Sarven; Guy, Amy; Lange, Christoph; Auer, SÃűren; Sambra, Andrei; Berners-Lee, Tim; *Linked Data Notifications: A Resource-Centric Communication Protocol*

[28] Longley, Dave; Sporny, Manu; Allen, Christopher; *Linked Data Signatures Specification*; https://w3c-dvcg.github.io/ld-signatures

[29] *JSON-LD Linked Data Signatures Library*; Longley, Dave; Lehn, David I.; Sporny, Manu; Collier, Matt; Wood, Harlan T.; Duffy, Kim H.; https://github.com/digitalbazaar/jsonld-signatures

[30] Prud'hommeaux, Eric; Seaborne, Andy; *SPARQL Query Language for RDF*; https://www.w3.org/TR/rdf-sparql-query

[31] *Blockstack Apps*; https://app.co/blockstack

[32] Reed, Drummond; Sporny, Manu; Longley, Dave; Allen, Christopher; Grant, Ryan; Sabadello, Markus; *Decentralized Identifiers*; https://w3c-ccg.github.io/did-spec

[33] Sporny, Manu; Longley, Dave; Webber, Chris; *Veres One DID Method*

[34] Sporny, Manu; Longley, Dave; *Web Ledger Protocol*

[35] Lemmer, Christopher; Miller, Mark S.; *Linked Data Capabilities*

[36] Endres-Niggemeyer, Brigitte; *Semantic Mashups - Intelligent Reuse of Web Resources*; Springer 2013

[37] Hogan, Aidan; *Skolemising Blank Nodes while Preserving Isomorphism*; WWW '15 Proceedings of the 24th International Conference on World Wide Web (WWW) 2015.

[38] Hogan, Aidan; *Canonical Forms for Isomorphic and Equivalent RDF Graphs: Algorithms for Leaning and Labelling Blank Nodes*; ACM Transactions on the Web (TWEB) 2017.