REBOOTING THE WEB OF TRUST

DESIGNING THE FUTURE OF DECENTRALIZED SELF-SOVEREIGN IDENTITY

A WHITE PAPER FROM RWOT XI: THE HAGUE

Identifier Binding: defining the Core of Holder Binding

by Paul Bastian, Rieks Joosten, Zaïda Rivai, Oliver Terbu, Snorre Lothar von Gohren Edwin, Antonio Antonino, Nikos Fotiou, Stephen Curran, and Ahamed Azeem

RWOT XI GOLD SPONSORS:

Abstract

The W3C Verifiable Credentials Data Model (VCDM) specifies **Verifiable Credentials (VCs)**[1] as a collection of **claims** that are **issued** by a single **party**, and **Verifiable Presentations (VPs)** as a collection of **claims** that a **holder** can construct from different **VCs** issued by different **parties**. Over the last year(s), various issues have been raised that revolve around what has been called 'holder binding'. The term 'holder binding' itself isn't clearly defined, and is in fact quite contentious. This paper seeks to come to grips with this discussion. Our first contribution is the specification of a terminology, which is intended to help readers understand what we mean to say without requiring them to make assumptions about such meanings (as is often the case in discussions about 'holder binding'). Our second contribution is an analysis of a (fictitious) use-case that suggests that **verifiers** typically do not need to know who the **holder** is (i.e. who has presented the **claims** to be **verified**). This analysis shows that **verifiers** need capabilities to (a) learn which **entity** is the **subject** of a particular **claim**, and (b) to know whether or not two **subject identifiers** refer to the same **entity** or to different **entities**. Also, they may need assurances regarding the **party** on whose behalf the **component** that has electronically presented the claims, has been using those capabilities. Our third contribution is a proposal for the syntax and semantics of a new property that can be used in (different parts of) **VCs**/**VPs**, that will provide **verifiers** with such capabilities.

Acknowledgements

This work was partly funded by the eSSIF-Lab project under EU H2020 Research and Innovation Programme - Grant Agreement Nº 871932.

This work was partly funded by the IDunion project supported by the Federal Ministry for Economic Affairs and Climate Action (BMWK) on the basis of a decision by the German Bundestag - Grant Agreement 01MN21002L.

This work was partly funded by DIN(Digital Identity Nordics) a non profit organization in Norway striving to work for better digital identity in the nordics.

The work at RWOT#11 was sponsored by (Gold Sponsors) the Hague University of Applied Sciences, the City of the Hague, TNO (eSSIF-Lab), the Dutch Blockchain Coalition, Digital Contract Design, (Contributing Sponsors) Spherity, Jolocom, and (Collaborating Sponsors) Blockchain Commons and Legendary Requirements.

Terminology

In VCDM issue #902, Orie Steele sighed "My guess is that 'holder binding' would be far less contentious if we could define what a holder is first." While the term 'holder binding' is indeed quite contentious, it is not that the term isn't defined. Rather, its definition is not actually used; it is effectively ignored, so every time someone uses the term, readers need to hallucinate about what it means. It is not a problem if someone uses the term in a different meaning, as long as it is accompanied with a definition that actually allows readers (which include non-native english speakers, non code-writers, etc.) to determine what is, and what is not an instance (example) of the term. The problem is that authors do not make that effort, readers accept this and interpret the term as they see it, and the result is… well, you can see it for yourself.

Terms such as 'holder' also suffer from 'terminological confusion' as a result of writers and readers not being aware of the fact that the meaning of terms is typically limited to a specific scope/context. For example, in the context of the Dutch government, the holder (of an identity document) is defined as "the person in whose name the travel document is issued and for whom it has been issued". In the context of VCDM (and using its definitions), that person would be referred to as the subject (of the identity document), and the holder would be the entity that possesses it and can present it, which could be, but

1 Bolded texts are terms, the meaning of which is specified in the section "Terminology". We have based this terminology on that of the VCDM and of eSSIF-Lab, so that it has the precision that allows us to better (and more formally) identify and express the concerns that this paper seeks to address than would have been the case if we would have simply referred to the VCDM terminology.

is not necessarily, its subject.

In an attempt to prevent any kind of misunderstanding, this section defines the key terminology that we use throughout this paper.[2] Each term starts with a sentence that states the criterion that you can evaluate to determine whether or not something is an instance (example) of that term. Subsequent sentences provide additional information, e.g. the purpose (why we need the term, what you can do with it, and/or specific characteristics that you may want to keep in mind).

The idea behind (a) making our terminology explicit, (b) consequently using these terms as we defined them, and (c) requesting you to interpret them in the way that we defined them, is that this prevents most misunderstandings. The consequence, however, is that we may use terms in a meaning that is different from what you or others typically take it to be[3]. But that's life: authors need to make a real effort to write texts that others can understand, and readers have to make a real effort to interpret these words with the intended meanings.

Actor

> An **entity** that can act (do things/execute actions). This includes e.g., people (human actors), machines (non-human actors), and (running) apps (digital actors). It does NOT include organizations. We can say that a **party** acts, but that should be interpreted to mean that an **actor** exists that performs this action on behalf of that **party**.

Agent (of a party)

> An **actor** that is in the process of executing an action on behalf of that **party** (at runtime). An **actor** that is doing this fulfills the role of **agent** for that **party**. After the action terminates, the actor no longer fulfills the role of agent.

Attribute

> A digital representation of a feature, characteristic, or quality that a **party** has ascribed to a specific **entity**. This data typically comes as a 'key-value' or 'predicate-object' pair.

Authenticate, Authentication

> The process or action executed by a **party** to convince itself that a particular **identifier** actually (truely, genuinely) **identifies** a specific (real world) **entity**.[4] Note that proving control of an **identifier** (particularly for DIDs) in itself does not constitute **authentication**.

Claim

> A digital representation of a statement that a **party** (called the author of the **claim**) has made about an **entity** (called the **subject** of the **claim**). The statement may or may not be true. The **subject** of the **claim** may or may not be identifiable. A **claim** may have a **subject identifier**, with the author being the authority for its dereferencing. A **claim** may also consist of one or more **attributes** that its author has ascribed to the same **entity**.

Component

> An **actor**, consisting of hardware and/or software, that operates in the digital realm. Typically **components** act on behalf of a specific **party** (thus being an **agent** for that **party**). Specific kinds of components will do things that are specific for their kind. Examples include **issuer components**, **verifier components**, or **wallets**.

2 While some readers may find some of the terminology 'selbstverständlich', properly defining it will enable those who use it to efficiently and effectively resolve any issues that may arise from terms that would otherwise have remained ambiguous.

3 Another consequence is that we will not entertain feedback stating or suggesting that some specific term has the wrong definition/meaning. We do not care too much about the terms themselves, because their purpose is to refer to meaning (described by the definition), and this meaning is what we care about.

4 The NIST definition refers to authentication as: "Verifying the identity of a user, process, or device, often as a prerequisite to allowing access to resources in an information system". However, the term "identity" is very difficult and might be misleading, see e.g., eSSIF-Lab.

Credential

A set of one or more **claims** made by an **issuer**. The **claims** in a **credential** can be about different **subjects** (and because of this, there is no such thing as *the* **subject** of a **credential**). See also: **verifiable credential (VC)**.

Entity

Someone or something that is known to exist. This is really anyone/anything, e.g. another person, yourself, some computer, an extinct animal, a thought, an idea, a JSON-object, …. , as long as there is at least one **party** that knows of its existence.

Holder

A role that a **party**[5] can perform as it (a) requests and obtains a **VC** from an **issuer**, (b) manages **VCs** within a **wallet**, or (c) creates **VPs** from one or more **VCs** and sends them to the **verifier** that requested it. A **holder** is usually, but not always, the **subject** of a **claim** in one or more of the **VCs** that it uses to create a VP.

Holding

Any of the following activities:

1. sending a request for obtaining a **credential** (to **issuer components** of other **parties**), extracting the **credential** out of the response, and sending/storing it in (one of) the **wallet** of the **party** on whose behalf this action is executed;
2. (1) receiving a **presentation request**, (2) deciding whether to accept or reject the request, and if accepted, (3) obtaining **VCs** with which to construct the requested **presentation** from the **wallet** of the **party** on whose behalf this action is executed, (4) constructing the **presentation** as specified in the **presentation request**, adding meta-data, and cryptographically sign it, and (5) sending the result to the requester as a response to the presentation request.
3. securely storing and protecting (possibly sensitive) data, e.g. **VCs** that have been received, on behalf of the **party** on whose behalf this action is executed
4. controlling the (create/read/update/delete) access of the securely stored/protected data on behalf of the **party** on whose behalf this action is executed

Identify, Identification

The action or process of asserting that a (real-world) **entity** is known by a specific **party**.[6] Such an assertion can be done by anyone (including that **party**), and can take many forms, e.g. saying/presenting a name, an **identifier** (e.g. an email address, or a username) a (set of) characteristics, or relations with other **entities** (e.g. as in a passport), etc.

Identifier

Data that is used for the purpose of recognizing a (real world) **entity**, typically to distinguish it from other **entities** in some set. The data is typically in the form of characters (or attribute sets), but could also take the form of audio (speech), pictures (portrait), etc., or a combination of those.

Identifier Binding

The situation in which there is an **identifier** that a particular **party** has bound to some **entity** that it knows to exist, and has specified one or more means that other **parties** can use to **identify** and/or **authenticate** that **entity**. Such means are typically specified as part of a **VC**.

---

5 The W3C VCDM defines the holder as an '**entity**', leaving it to the reader to determine from the context whether or not a **party** is intended, or a **component** (an **actor**) that acts on behalf of such a **party**. For a discussion about distinguishing between **parties** and **actors**, see the eSSIF-Lab mental model on Parties, Actors and Actions.

6 This is also sometimes phrased as "selecting (or: singling ont) one **entity** out of a set of **entities**". See e.g. Pfitzmann and Hansen (See Anon_Terminology_v0.34 (tu-dresden.de))

Identifier Semantics (of a Party)

The mapping that a particular **party** uses between the **identifiers** that it uses and the **entities** that it knows to exist. Every **party** maintains such a mapping in its own, autonomous way. In such a mapping, every **identifier** refers to (represents) a single **entity**, whereas an **entity** can be referred to (be represented by) multiple **identifiers**.

Issuer

A role that a **party** can perform as it authors **claims** about one or more **entities**, and creates a **VC** from these **claims**, and transmits the **VC** to a **holder**.

Issuer component

A **component** that is capable of executing the actions on behalf of a **party** in its role of **issuer**. It does so in compliance with that **party's** issuer-**policy.**

Issuing

An activity that consists of (1) receiving a request for issuing a **VC** of a certain type, (2) deciding whether to accept or reject that request, and if accepted[7], (3) constructing the **claims**, adding metadata, cryptographically signing it, and (4) sending the result to the requester as a response to its request.

Party

An **entity** that has its own objectives, knowledge, semantics (which includes its **identifier semantics**), logic (for reasoning), and decision rules, as well as the capability to manage/maintain these in an autonomous (self-sovereign) fashion. Colloquially, it is an **entity** that can be said to have 'a mind of its own'. Their 'minds' (subjective knowledge) are what distinguishes **parties** from each other (every **party** is 1-1 related to its knowledge/mind). Typical examples are individual people and organizations (but also parts of organizations, such as departments). Note that not all **parties** are capable of acting (e.g.: organizations). When we say that a **party** acts, this means that an **actor** exists that performs this action on behalf of that **party**. Further explanations can be found in eSSIF-Lab.

Policy

A (set of) rules, working instructions and/or other guidance for the execution of one or more kinds of actions that **agents** of the **party** that governs the policy have access to and can interpret such that this results in these **actions** being executed as intended by that **party**.

Presentation

Data that a **holder** sends to a **verifier** as a response to a **presentation request**, and that contains data that has been derived from one or more **VCs issued** by one or more **issuers**. See also: **verifiable presentation (VP)**.

Presentation Request

a (signed) digital message that a **verifier component** sends to a **wallet** asking for specific data from one or more **VCs** that may be **issued** by different **parties**, and where this data must satisfy specific constraints and/or come with specific proofs or evidence.

Subject

The **entity** to which a given set of coherent **attributes** relates/pertains. In a **VC**, every **claim** has a subject (**VCs** themselves do not). In an AnonCred (a different **credential** flavor), there is a single **subject** (i.e. all **attributes** in an AnonCred pertain to the same **entity**).

7 If the request is rejected, the requester may be notified of that.

Subject Identifier

An **identifier**, typically in the form of a character string, that is being used for the **identification** of a **subject**. The **party** that has authored the **subject identifier** is the authority for dereferencing it (i.e. for determining the **subject** that it **identifies**). For **subject identifiers** that are used in **claims** (in a **VC**), this would typically be the author of the **claim** (which in most cases is also the **issuer** of the **VC** that contains the **claim**).

Subject Identifier Semantics (of a Party)

The mapping that a particular **party** uses between the **subject identifiers** that it uses in the **claims** that it **issues**, and the respective **entities** to which they refer (the **subjects** of these **claims**).

User

A role that a **party** can perform as it uses a **component** (as an **agent**) to establish and maintain a connection with another **component** of which it seeks to obtain a service. Users are said to request and/or use and/or obtain such a service, which they do through a user-interface of their **agent**. Wikipedia calls this the end-user.

User component

A **component** that is capable of executing the actions on behalf of a **party** in its role of **user**. Typical examples include internet browsers, but also other computer applications (e.g., on mobile phones, tablets or computers). The instructions that **user components** get for acting on behalf of some **party** (the **policy** of the **party** on whose behalf it is acting) can be located in a configuration file, or given in terms of settings/preferences or as instructions that the the person that operates the **component** provides, e.g., through a user interface.

Verifier

A role that a **party** can perform by (a) requesting one or more **VCs** (by sending a **presentation request** to a **holder**) optionally as a **VP**, that are intended to be used for a specific purpose, and (b) receiving a response, of which it verifies the structure and proofs.

Verifier component

A **component** that is capable of executing the actions on behalf of a **party** in its role of **verifier**. It does so in compliance with that **party's** verifier-**policy**.

Verifiable Credential (VC)

A **credential** that is tamper-evident and contains a proof about the **issuer** that can be cryptographically verified. Typically, this proof **identifies** the **issuer**, but it could also be a proof that the **issuer** has been certified by another (possibly **identifiable**) **party**. **VCs** can be used to build **VPs**.

Verifiable Presentation (VP)

A **presentation** that is tamper-evident and contains a proof about its author that can be cryptographically verified. Certain types of **VP** might contain data that is synthesized from, but do not contain, the **VCs** from which the data was synthesized (for example, zero-knowledge proofs).

Wallet

A **component** that is capable of executing the **actions** on behalf of a **party** in its role of **holder**. It does so in compliance with that **party's** holder-**policy**. Wallets typically have additional functionalities that make them useful for particular purposes (e.g. banking). **Wallets** may use local storage or remote storage components for the **VCs** they obtain, and for getting **VCs** from which they construct **presentations**. Multiple wallets may access the same (local or remote) storage provided they are an **agent** of the same **party**.

1 Introduction

All SSI[8] technologies exist for the purpose of supporting **parties** (individuals and organizations) that want to perform some kind of (business) transaction, for which both need to decide whether or not to commit to that transaction. We will see an example of this where Bob (who provides courses) and Alice (who wants to take a course) interact. At various points, Bob and Alice need to decide things, e.g. which course to take, whether or not to enroll Alice as a student for that course, whether or not to provide access to a course to a person claiming to be a (registered) student, etc.

Each such **party** gets to autonomously decide which set of rules it will use for making such decisions. Having done so, the **party** must then know where to get the data it will need to evaluate such rules. If that data is to be obtained from an external source, it makes sense to require that both the integrity and the authorship of such data is verifiable, and that's where **VCs** and **VPs** come in.

However, a **party** also needs to ensure that this data is valid[9] for the evaluation of these rules, because using invalid data may lead to wrong decisions that can cause real harm. What is or is not valid data for a particular rule is a subjective judgment of the **party** that makes decisions based on such rules. Thus, we expect that each **party** specifies which kinds of **claims** and **VCs** it deems sufficiently valid to rely on for evaluating a particular rule — not only because of what the **claim** means, but also because of who the **issuer** is, the kinds of processes that it (claims to have) used that result in it making these **claims**, etc.

While a **party** needs to know who the **issuer** (of a **claim**/**VC**) is as it is gathering data for making transaction-related decisions, it typically has no need to know who actually presents a **claim**/**VP** (the **holder**), or to whom a **claim**/**VC** was issued. This does not mean, however, that a **party** isn't interested to learn who it is that performs other (typically transaction dependent) roles, but those are different matters. We will elaborate on this in the use case scenarios that we describe further on.

Evaluating rules that use **claim** data requires the **verifier** to be aware of what **entity** the **claim** makes a statement about (the **subject** of that **claim**), as well as what this statement actually means. For the purpose of this paper, we consider the awareness of what this statement actually means out of scope.

Concerning the **subject** of the **claim**, we think that a **verifier** would have three requirements that it would like to see fulfilled, which are:

- **identify** the **entity** that is the **subject** of a **claim** (typically the binding between **subject** and **subject identifier**),
- **authenticate** the **entity** that is the **subject** of a **claim** (typically binding between the **subject** and the **claim**) and
- establish whether the **subject** of two **claims** (authored by the same or different **parties**)
    - are in fact the same **entity**,
    - are different **entities**, or
    - are **entities** of which it cannot be determined that they are the same or different.

Providing means to fulfill at least some of these requirements is the topic of this paper.

8 SSI (Self-Sovereign Identity) is a term that has many different interpretations; we use it to refer to concepts/ideas, architectures, processes and technologies that aim to support (autonomous) **parties** as they (electronically) exchange data for the purpose of conducting transactions with one another.

9 Verification and validation are different things. See description of validation by eSSIF-Lab.

The actual support that these SSI technologies provide for **parties** that need or want to exchange information, comes in the form of (digital) **components** that act on their behalf. Within SSI, we are familiar with functionalities that are known as '**issuing**', '**holding**' and '**verifying**'. **Parties** that are said to do this (and thereby assume the roles of '**issuer**', '**holder**' and '**verifier**' as appropriate), will typically employ (and control) **components** to execute such actions on their behalf.

The following figure shows a number of **parties** (most of which will also appear in the use case) and for each of them some (functional) **components**[10] that can execute SSI functionalities on its behalf e.g. for **issuing**, **holding** (**wallets**), or **verifying**, as well as for e.g. securely storing **VCs**.

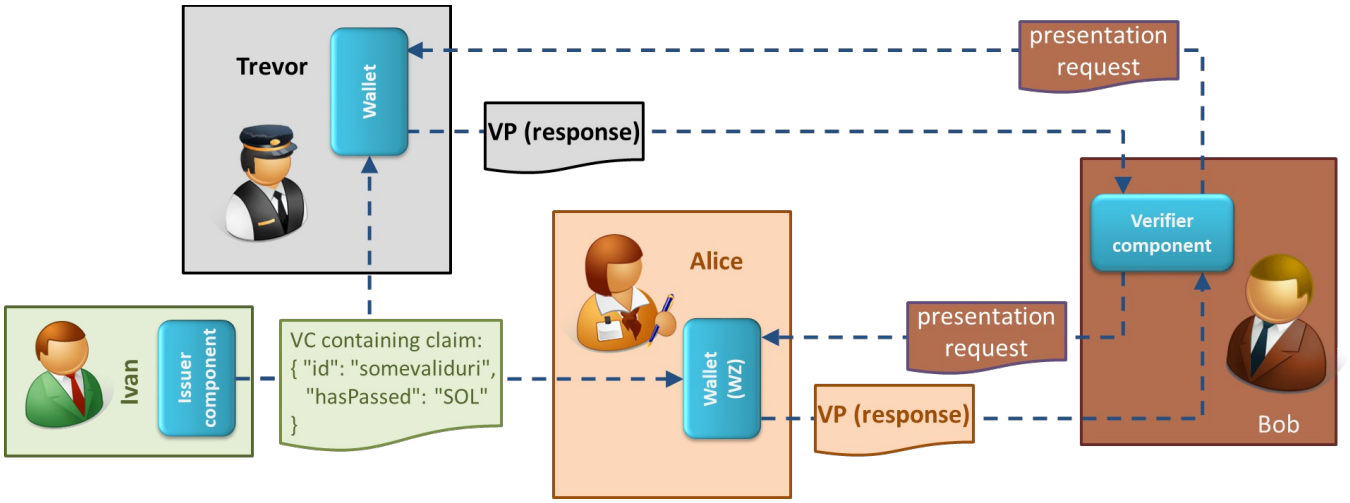

Figure 1: Parties and some (functional) components that work on their behalf..

The colors of the rectangles and messages represent the **party** that controls its contents. Thus, Alice controls a **wallet** and the contents of the **VP** (response) message.[11]

In this paper, we constrain ourselves to what is specified, or assumed, by the W3C Verifiable Credentials Data Model (VCDM). We intend to contribute to its value by further enabling **Verifiable Credentials** (**VCs**) to be used. Specifically, we want to focus on how **verifiers** can use **claims** to actually evaluate their business rules as they set up and conduct business transactions, by providing means for their requirements Req1, Req2 and Req3[12].

A **verifier** that wants to use **VCs** and **VPs** is provided with guidance, e.g. when it comes to the **semantics** of properties used in **claims** (as specified in their `credentialSubject` property), but has little, if any, guidance when it comes to learning which **entity** is the **subject** of a given **claim** (i.e. to which **entity** the **subject identifier** of that claim refers).

The VCDM says that every **claim** (in the `credentialSubject` property) can have an `id` field that is "intended to unambiguously refer to an object, such as a person, product, or organization", suggesting that this would then be the **subject** of that **claim**.[13] However, it does not provide any actual guidance about how a **verifier** can learn which **entity** is being

10 Each such component is expected to be provided with a (machine-readable) **policy** that contains the rules, instructions, and other guidance that ensures they will execute the actions in accordance with the intentions of the **party** on whose behalf they do so. This topic is out of scope for this paper.

11 Note that the figure does not distinguish between online, offline, or mixed modes of requesting and providing presentations.

12 this perspective differs from the usual one, in which a **holder** seems to determine what (not) to present to a **verifier**, or where a **verifier** would need to know that the **holder** is the **subject** of the **VC** (disregarding the VCDM specifications that say that a **VC** can have claims about multiple subjects, none of which is necessarily the **holder**)

referred to.[14] Still, the 'holder binding problem' often seems to be a request for providing guidance on this matter. So how does this work in the real world? What can we learn from what happens there?

In the real world, any **party** that authors (creates) an **identifier** determines the **identifier semantics**, i.e. the mapping between the **identifier** and the **entity** to which it refers. This must particularly be the case if such an **identifier** appears as part of a **claim** that this **party** subsequently signs.

From this, it follows naturally that every **issuer** gets to determine its own **subject identifier semantics**[15]. Responsible **issuers** would also insist on this actually being the case, because if some other **party** could change that semantics, it could also change the meaning of the **claim** that the **issuer** has signed and issued as a **VC** some time after the **issuer** has signed it, without needing to inform the **issuer** (who doesn't then have reason to revoke it), and without leaving any trace that would signal a **verifier** that the statements that he thinks the **issuer** has made differ from the statements that the **issuer** has actually made.[16]

The remainder of this paper is organized as follows. Chapter 2 describes a simple use-case using various scenarios, where we develop the needs of the **verifier**. Chapter 3 makes a proposal that can readily be made part of the VCDM, illustrates how it can be used in different ways, and lists various implementation-, privacy- and other kinds of considerations. Chapter 4 describes how we see it work in practice. We wrap up with conclusions and future work.

13 If the `id` field is omitted, the VCDM says it is a 'bearer claim', meaning that whoever presents the **claim** must be considered its **subject**.

14 The statement that "It is *RECOMMENDED* that the URI in the `id` be one which, if dereferenced, results in a document containing machine-readable information about the `id`." is a statement about the `id`, not about the **entity** to which that `id` refers.

15 It could even be useful for an issuer to use a primary index of his private database as an identifier for the credential subject.

16 Suppose Alice can determine the semantics of the **subject identifier** that Ivan used in a **claim** that supposedly states that Alice has earned some degree. This would happen if Ivan uses a DID that Alice controls (because the DID spec says that "a DID refers to any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) as determined by the controller of the DID"). When Ivan **issues** a VC that contains this **claim** under the assumption that the **subject identifier** refers e.g., to Alice, and Alice changes it to refer to Chuck, then the **VC** contains a **claim**, signed by Ivan, saying that Chuck has earned the degree.

2 Use Case: Course Enrollment

We assume that for the general audience of this paper, the term ‘holder binding’ is quite contentious. Rather than joining the disputes of what it is, or should be, we will revert to a conceptually very simple use-case that can operationally be executed in many ways. We expect that expressing these operational ways using our terminology in a consistent way will illustrate what ‘holder binding’ could (or should) be.

In the subsequent sections, we will use the phrase ‘**issuing** (or: **verifying**) a/the **claim**’ as a shorthand for ‘**issuing** a **VC** that contains a/the **claim**’. Similarly, we use the phrase ‘**verifying** (or **presenting**) a/the **claim**’ as shorthand for ‘**verifying** (or **presenting**) a/the **VP** or **VC** that contains a/the **claim**’. The reason for this is that the purpose of **VCs** and **VPs** is to provide guarantees regarding the integrity and authorship of the **claims** they contain, while for this use case, we want our focus to be on the **claims** and the statements they represent. Introducing these shorthand phrases helps to keep the texts more readable while maintaining their intended meaning.

The conceptual use case is one in which Bob offers the course called "Making Logic Arguments Stick". As a prerequisite for the course, students must have mastered Second Order Logic, which they can prove by presenting a **VC** that is **issued** by Ivan (the teacher or organizer of that course) which contains a **claim** asserting this. The operational use case is where Alice gets enrolled in the course, and gets access to its lessons.

In the following sections, different scenarios are described that lead to Alice being enrolled and her getting access to the lessons (rather than someone else). But first, we provide the background that each of these scenarios builds upon.

2.1 Background

Before Alice can be enrolled, Bob must first put the course’s enrollment process in place, and Alice must collect the things she needs for that enrollment. This section describes the prerequisites from the perspectives of Ivan, Bob and Alice, respectively.

2.1.1 Ivan

Ivan **issues VCs** that contain **claims** of the form `(`*`si`*`,"passed",<exam>)`, where *`si`* is the **subject identifier**, `<exam>` is the name of an exam, and `"passed"` asserts that the **entity** that is **identified** by *`si`* (the **subject** of the **claim**) has passed the exam that has the name `<exam>`.

Ivan uses `"SOL"` as the name of the exam that shows mastery of Second Order Logic.

In order to allow potential **verifiers** (such as Bob) to learn that Ivan issues such **VCs** and to help them to decide whether or not to use them for their particular purposes, Ivan has published an offer that states the **claim** type, its meaning, the endpoint(s) where **holders** may obtain **VCs** containing such a **claim**, the conditions under which such **VC**s will be **issued**, and any other information that **parties** such as Bob may need.

2.1.2 Bob

Bob has decided to offer several ‘course variants’, i.e. options for following the course:

1. a Massive Online Open Course (MOOC), that is available 24/7 online;
2. every semester, a series of online video sessions, run by a real teacher;
3. every summer, a summer school that consists of a series of physical sessions (run by a real teacher) at a designated location.

Enabling people to follow one of the course variants implies enabling the teachers (which we here take to include the MOOC server) to determine whether or not a person that requests access to the course variant (we will call them the ‘requester’) is actually entitled to follow it. To allow the request, the teachers must make sure every of the following conditions is fulfilled:

- a **claim** of the form `(pi,"isEnrolledFor",cvi)` exists (which asserts that the person **identified** by `pi` is eligible to follow the course variant **identified** by `cvi`);
- this **claim** is **issued** by a **party** that Bob can rely on to have ensured that all conditions that he has set for entering the course are fulfilled. For this use case, we only consider Bob to be such a 'trusted **issuer**';
- `cvi` **identifies** a course variant that is being taught by the teacher;
- `pi` **identifies** the requester (i.e. the requester is the **subject** of the claim).

Bob first designs a process for **issuing** such **claims** and then implements an enrollment application that will run this process on his behalf.

In his issuing process, Bob's enrollment application first needs to decide whether or not to accept a request for some person to be enrolled in a particular course variant. After having accepted such a request, the application can continue to construct a **claim** of the form `(pi,"isEnrolledFor",cvi)` and issue this **claim**.

The enrollment application only accepts a request if

- an **identifier** `cvi` is selected by its **user**, that represents the course variant. This can be done by the application presenting the list of course variants (each of which is associated with a particular `cvi`) for which enrollment is (still) possible, having the **user** select one, and using the associated `cvi`.
- all conditions are met that Bob requires to be fulfilled for enrollment. In this use case, we limit that to the condition that the student to be enrolled has mastered second order logic, and can prove that by presenting a **claim** of the form `(si,"passed","SOL")` that is **issued** by Ivan.[17]
- an **identifier** `pi` has been made available that represents the student. This **identifier** can be conveniently copied from the aforementioned **claim**, i.e.: `pi=si`).

After Bob's enrollment application has accepted the request for enrollment, it starts by creating the **claim** `(si,"isEnrolledFor",cvi)`, where `si` is obtained from the **claim** that was **issued** by Ivan and presented by the **user** of the enrollment application, and was inferred by the enrollment application after the **user** selected an element from the list of presented course variants. Then, it constructs a **VC** containing the claim, stores it and sends it to the **user**.

2.1.3 Alice

Alice wants to get herself enrolled for Bob's course "Making Logic Arguments Stick", and sees on the website that this requires her to be able to present a **claim** of the form `(si,"passed","SOL")` that is **issued** by Ivan.

Here, we assume that

- Alice has successfully obtained a **VC** (**issued** by Ivan) that contains the **claim** `("somevaliduri","passed","SOL")`,
- at the time of **issuing**, Alice was the **subject** of that **claim**, and
- this **VC** is stored in a wallet and
- the **wallet** that will be used in the enrollment process, has access to that **VC**

There are many ways in which these assumptions might be realized, the most obvious of which is that a **wallet** belonging to Alice electronically requests such a **VC** from the **issuer component** of Ivan, which then decides whether or not to issue a **VC** to Alice, and if so, what **claim(s)** to include in it (and which **subject identifiers** to use), and then send it to Alice's **wallet** which then stores it securely into Alice's wallet.

17 This is because Bob has searched for such **claims**, found the offer of Ivan, and based on the information he found there, decided to rely on these **claims** of Ivan.

Alternative workflows exist. One is that Alice may ask Trevor to get such a **VC** and devise a way to get it into her own wallet; but there are many others. We will see that in the scenarios we present, the above assumptions are sufficient to get the use case done.

2.2 Registration Scenarios

This section describes different ways in which Alice can be accepted as a student of the course "Making Logic Arguments Stick" that Bob has on offer.

2.2.1 Alice registers herself

She does so by pointing her browser to Bob's enrollment application, where she chooses the course "Making Logic Arguments Stick" (the specific variety is identified with `MLAS-3`). Bob's enrollment application (that includes a **verifier component**) connects to the **wallet** she has chosen to use, and sends it a **presentation request** that asks for a **claim** of the form `(si,"passed","SOL")` that has been **issued** by Ivan. Alice's **wallet** finds the **VC issued** by Ivan that contains `("somevaliduri","passed","SOL")`, sends that **VC** to the **wallet** which in turn constructs a **VP** that includes the **claim** and the proof that Ivan has **issued** it, and sends the **VP** to (the **verifier component** in) Bob's enrollment application.

As the verification checks out, Bob's enrollment accepts the application, creates the **claim** `("somevaliduri","isEnrolledFor","MLAS-3")`, saves it, constructs a **VC** containing the claim, stores it, and **issues** it to the **wallet** of Alice.

2.2.2 Trevor registers Alice

For various reasons, Alice may want (or need) someone else to enroll her for the course. She would do well to only ask someone she can entrust with this task. She decides to ask Trevor.

Trevor proceeds in exactly the same way as Alice did in the previous scenario. He only needs the ability to find the **VC** that was **issued** by Ivan and that contains `("somevaliduri","passed","SOL")`.

There are several ways this can be arranged: Alice can send the **VC** to him, she could provide the **wallet** he will be using (for reading the particular **VC**), or she could provide him with the means that enable him to successfully obtain a **VC** from Ivan that includes that **claim**. The details of this are outside the scope of this particular use case.

Also, Trevor should make sure that the **VC** that was **issued** to his **wallet** by (the enrollment application of) Bob gets forwarded to Alice. This is not a necessity though: if Trevor doesn't do that, Alice could still request the **VC** from Bob as Bob has registered the fact that Alice was enrolled, similar to how Ivan issues **VCs** that contain claims about people that have passed exams.

2.2.3 Mallory registers Alice

We should also consider the possibility that Mallory (a malevolent **actor**) registers Alice for the course while Alice has no intention of taking the course. After all, **VC**s are just sets of **claims**, signed by an **issuer**; they can be transferred at will and it is reasonable to assume that people such as Mallory could get their hands on a **VC** and do ill-intended stuff.

This is not necessarily something that Alice would seek to prevent. That would only be the case if Mallory were able to instill a (legally enforceable) duty on her, e.g. to make some payment. Since Mallory would not be able to sign stuff on behalf of Alice, chances are that Alice wouldn't mind (or care).

However, it may constitute a problem for Bob: a variant of a 'Denial of Service' attack. If Bob were to accept an application that Mallory submits for Alice, and registers Alice as a student for the selected course variant, then the amount of free places would be reduced by one. This may lead to the situation where Bob thinks a course variant is fully booked, when in fact that is not the case. For the MOOC that may not be all that bad, but for physical courses it may. If this risk is not acceptable to Bob, he should take measures to mitigate it, e.g. by adding criteria for the acceptance of enrollment applications. Requiring full (unsubsidized) payment is one option: it could still leave places open, but that would not harm Bob (initially) and it would make the 'attack' more costly.

Another option is that Bob implements the enrollment application such that it asks its **user** to provide a **VC** that can be used to **identify** and **authenticate** the **user**, enabling Bob to find and sue those that have exhibited misbehavior. The enrollment application should make sure that this **identification** and **authentication VC** is valid for that purpose, e.g. by having the **user authenticate** using the **claims** in that **VC**.

2.3 Access Scenarios

Every course variant provides a particular context within which the teacher (physical person or IT component) is required to check whether or not a person that requests access should be permitted access.

2.3.1 Electronic access to the MOOC server

Bob's MOOC server is online 24/7, and students can use it where and when they like. That is to say: anyone using the MOOC service would need a **user component**, e.g. a web browser, that would access the MOOC service on its behalf. That **user component** will request access to the course provided by the MOOC service. The MOOC server must request a **presentation** that includes a **claim** of the form `(si,"isEnrolledFor","MOOC")` which a **wallet component** (that could be part of the **user component**) would provide. The MOOC server then needs to verify that

A. the **claim** has the requested format, and is **issued** by Bob, and that

B. the **wallet** operates on behalf of the **entity** to which *si* refers (the **subject** of that **claim**).

It doesn't matter whether the **user component** provides such a **claim** (as long as the MOOC server can verify it comes from a **VC** that Bob has **issued**), or that the MOOC server checks the registry (database, backend storage facility) in which Bob has stored the **VCs** that he **issued**. Both options work.

What *might* matter is whether or not the **wallet** and/or **user component** operate(s) on behalf of the **subject** of that **claim**. If the MOOC server does not establish that this is the case, then Trevor (having stored the **VC** that Bob issued to him upon his request to register Alice) can also access the MOOC server, thus impersonating Alice.

It depends on the risk assessment and risk appetite of Bob whether or not he wants to have the MOOC server check this. If he does, he may have the MOOC server request **claims** for which the **subject** is the **user component** and/or **wallet**, and that state, e.g., that its **subject** (the **component**):

- has been certified according to a particular scheme (signed by a certification agency),
- has the same integrity that it had immediately after it was installed (e.g. by having the **component** obtain an ephemeral **VC** (from a remote integrity attestation service) that contains a **claim** that states this), and
- has very recently created a **VC** containing a **claim** that says on whose behalf it is currently operating, i.e. that its **user** has logged into the **component** with a particular mechanism, or using a mechanism that comes with some predefined Level of Authentication (LoA).

As establishing this is a matter that appears in all access scenarios, we have dedicated a separate section to this, 2.4 Identifier Binding.

2.3.2 Physical access to a physical location

The summer school setup consists of a series of physical sessions (run by a real teacher) at a designated location. If the teacher of that course wants to ensure that only people that have properly registered for that particular course will attend, (s)he can request a **VP** that contain **claims** of the form `(si,"isEnrolledFor",cvi)` where *cvi* identifies the course variant that the teacher tutors, and the **claims** come from a **VC** issued by Bob. (S)he will need one such **claim** for every student, and finally, (s)he will need to learn which of the students is referred to by the various **subject identifiers** *si*. We refer the reader again to the separate section, 2.4 Identifier Binding, about this.

Note that it is not necessary that a student presents the **claim** `(si,"isEnrolledFor",cvi)` for which that student is the **subject**. For example, a group of students might have organized it such that the group leader would be able to present such **claims** for all group members.

2.3.3 Access to an online video course run by a real person

Access to the online video course can be checked by the video server that runs on Bob's behalf, but also by the teacher of the course that sets up the video conference.

A **user** would typically request access to the video server in the same way as it would request access to the MOOC server and the same mechanism applies (which we shall not repeat here).

However, **users** may also be allowed to access the call purely based on them having a valid URL, or some video server account, in which case the (human) teacher would need to check that all **users** are properly registered students for the online video course. (S)he can do that in the same way as with people that try to access a physical location where the course is held, and the same mechanism applies (which we shall not repeat here). The difference is that a **user** may be kicked out of the conference call rather than off the premises.

2.4 Identifier Binding

A situation that occurs regularly is one in which a **party** has bound an **identifier** to some **entity**, and/or there is a **party** that wants to learn which **entity**, if any, that **identifier** has been bound to. Observing how this works in the actual world helps to properly understand and come to grips with the mechanics involved.

In the real world, every **party** knows about a subset of all **entities** that exist. In order to talk and reason about them, it needs data (which we call **identifiers**) that it can use to refer to such **entities**, i.e. single out individual **entities** from the set of **entities** that it knows to exist. It is easy to observe that **parties** in the real world choose such data as they see fit. People that have been called names will realize that the ones calling them such names have exercised precisely that autonomy. You may now realize that you, the reader, too, have been doing your share of name giving…

We use the term **identifier semantics** (of a **party**) to refer to the mapping that this **party** maintains between the **identifiers** that it uses and the **entities** that it knows to exist. We observe that **parties** autonomously decide what this mapping actually is. This implies that what is an **identifier** for one **party** may not **identify** some **entity** in the view of another **party**. Also, if two **parties** both use some **identifier**, they may have it refer to different **entities** (example: "daddy", or "the best president we ever had").

Many **parties** will make some effort to make their **identifier semantics** 'interoperable' with that of other **parties** i.e. that a subset of the **identifiers** that it uses have the property that these **identifiers** (data) are also used by other **parties** to refer to that same **entity**.

It is a common misconception to assume that this is naturally the case. People that design and implement IT (as well as others) would do well to avoid assuming this. Readers are encouraged to read more about this topic on the site of the eSSIF-Lab framework.[18]

From the above, it follows naturally that whenever a **party** wants to learn which **entity** an **identifier** refers to, it can only do so if this is enabled by the **party** that determines the associated **identifier semantics**. For **identifiers** that are used in **VCs** (e.g. as a **subject identifier** in a **claim**), that should be the **issuer** of that **VC**.

In our use case, Ivan has used `somevaliduri` as an **identifier**. That means that Ivan should control the associated **identifier semantics**, i.e. determine which **entity** it refers to. As a consequence, Ivan is the single **party** that can provide other **parties** with means for learning which that **entity** is.

The current state of the VCDM (and DID) specs is such that there is no guidance whatsoever on **identifier binding**: **issuers** are not provided with (standardized) ways to provide other **parties** with means for learning which **entity** an **identifier**, specifically a **subject identifier**, refers to. As a consequence, **verifiers** have no other option than to make assumptions, which can be made to work in specific cases, but are not generally applicable.

We propose to use the phrase '**identifier binding**' to refer to a situation in which there is an **identifier** that a particular **party**

18 The eSSIF-Lab framework has pages for the terms **identifier** and **identify**, and a mental model on identification.

has bound to some **entity** that it knows to exist, and for which it has specified one or more means that other **parties** can use to **identify** and/or **authenticate** that **entity**. Such means would typically be specified as part of a **VC**. Be aware that '**identifier**' as in 'identifier binding' does not need to be an 'id'-like DID-ish string but can also relate to other data as stated in our terminology. Identifier binding might in most cases be used as 'subject binding' but we do not prefer that term to keep the concept more general. The following chapter contains some proposals for such means.

If such means were to exist, that would conclude the various access scenarios we described above, because then Ivan can add such means to the **VCs** that it **issues** and Alice can (selectively) disclose such means to Bob's teachers, which in turn will be enabled to then verify that it is Alice that has the right to access the particular course variant.

2.5 Discussion

The description of this use case was devised to include a minimum amount of assumptions. In particular, we do not want to assume that 'holder binding' is some kind of 'Deus ex Machina' that solves all sorts of problems. Rather, we have come up with a simple use case that nevertheless we expect to be so rich that if 'holder binding' has a useful meaning, we should be able to pinpoint that.

The above elaboration of the use case shows that it is not relevant who the **holder** actually is. In many situations, it is perfectly acceptable that someone other than the **party** that needs to be **identified** and **authenticated** would **present** the **claims** (in **VCs**) to a **verifier**, and hence is — by definition — the **holder** of these **claims** (**VCs**). Whether or not this is the case is to be determined by the **verifier**. So to prevent misunderstandings, we distinguish between the roles **holder** (that **hold** various **claims** (**VCs**)) and **user** (that interacts with (IT) services of other **parties**, and for which **identification** and/or **authentication** may be required).

All this suggests that the phrase 'holder binding' is not only contentious, but also misleading, as it leads people to focus on irrelevant things (such as proving who the **holder** is). The term contributes more to confusion than to solving actual problems. We propose to use the term **identifier binding** instead, as it better suggests the actual problems we are trying to solve, for which the next chapter will propose some solutions.

3 Proposal

An important part of making the W3C Verifiable Credentials Data Model (VCDM) work is to ensure that the **Verifiable Credentials** (**VCs**) that it specifies can be actually used. It is relatively easy to see how **issuers** and **holders** could use them, but it is not at all obvious how **verifiers** could do that. Use cases are often described using statements that suggest that a **holder** determines what (not) to present to a **verifier**, or that a **verifier** would need to know that the **holder** is the **subject** of the **VC** (disregarding the VCDM specifications that say that a **VC** can have claims about multiple subjects, none of which is necessarily the **holder**).

A **verifier** that does not want to rely on undocumented assumptions has little, if any, guidance when it comes to learning which **entity** is the **subject** of a given **claim** (i.e. to which **entity** the **subject identifier** of that claim refers).

The guidance provided by the VCDM is that DIDs are "most often" used in a **VC** as **subject identifier** but specifies that, if it is provided, it must be a URI. For URIs, there is no further guidance, and for DIDs the DID spec says that "a DID refers to any subject (e.g., a person, organization, thing, data model, abstract entity, etc.) as determined by the controller of the DID", so a **party** should find out who controls that DID. As we mentioned earlier, it is the **issuer** of a **VC** that should control the **identifier semantics**, so what the DID spec text thus implies is that the **issuer** should be the controller of every DID that it uses as an **identifier** (including **subject identifiers**). This is contrary to what many people think. But regardless of that, the DID spec, too, does not provide any guidance about how a **verifier** can learn which **entity** is actually being referred to.

3.1 The `binding` property

We propose to specify a new property, provisionally called `binding`, the purpose of which is to enable **parties** (specifically those in the role of **verifier**), to determine which **entity** a particular **identifier** refers to when it is used in a **VC** or **VP**. We start with elaborating on our proposal, and proceed to give examples of how it can be used.

Here is an example of how this property can be used:

```
...
      "binding": [ {
         "id": [ "somevaliduri" ],
         "type": "didAuthenticationKey",
         "keyId": "did:example:deadbeefcafe#keys-3"
      }, {
         "id": [ "somevaliduri", "anothervaliduri" ],
         "type": "passport",
         "nationality": "NL",
         "passportNr": 012345678,
         "contentHash": "3338be69 ... 2398f392"
```

```
      }, {
         "id": [ "somevaliduri" ],
         "type": "portrait",
         "format" : "png",
         "portrait": "iVBORw0KGgoAAAANSUhEUgAAA8sAAAJl..."
      } ],
...
```

In the above example, the `binding` property is an array that contains three elements, each of which can help **verifiers** to **identify** and **authenticate** the **entity** that the author of this property refers to using the **identifier** `"somevaliduri"`. Let's assume that would be Alice.

The first `binding`-element (of type `"didAuthenticationKey"`, says that the **entity** that is identified with `"somevaliduri"` has the private key material associated with `"did:example:deadbeefcafe#keys-3"`. Thus, a **verifier** can ask a **user** that purports to be identifiable with `"somevaliduri"` to prove she has that private key material. Note that this does not need (and actually also should not be) the private key material that authenticates the DID controller, as explained before. This is an example where the `binding` property can be used for online **identification** and **authentication**.

The second `binding`-element (of type `"passport"`), says that the **entity** that is identified with either `"somevaliduri"` or `"anothervaliduri"` is the holder of a passport issued by the Dutch government, with passport serial number 012345678. Thus, a **verifier** can ask a **user** that purports to be identifiable with either identifier to show a passport, verify that it has been issued by the Dutch government and has serial number 012345678, and from there see if the person matches the photograph and everything else that is customary when **authenticating** someone by her passport.

The third `binding`-element (of type `"portrait"`), specifies a portrait (an image of the front of the head of a person) in `png` format that, when rendered on a graphics device, enables humans to determine whether or not some arbitrary person is the one in the image. Thus, a **verifier** can ask a **user** that purports to be identifiable with `"somevaliduri"` to show her face, compare that with the picture, and decide whether or not they match, i.e. the person is the one represented by the picture (and therefore, also by `"somevaliduri"`).

The example shows that binding-types can be devised for different contexts: the (suggested) `didAuthenticationKey`-type is useful for remote, electronic **identification**, and the `passport` and `portrait` can be used for local and/or deferred physical **identification**. This setup is easily extended with other means that enable **verifiers** to **identify/authenticate** the **entities** that serve as the subject of some **claim**. Of course, it would be useful to standardize, or recommend, a set of simple and common binding-types. That, however, is outside the scope of our proposal.

We propose the `binding` property to be specified as an array of elements that enable the **identification** and **authentication** of some **entity**, where each element consists of:

- an (optional) `id` field, i.e. a (possibly empty) list of the **identifiers** that all refer to the **entity** that can be **identified** and **authenticated** by the contents of this `binding`-element. If the `id` property is not specified, then the binding-element can be used to **identify** and **authenticate** the entity to which the sibling-properties of `binding` are attributed.
- a (required) `type` field, i.e. an **identifier** that specifies the method/mechanism for **identifying** and

authenticating the **entity** from a registry of types, where a registry[19] specifies which key-value pairs it expects (optionally, or required), and how they are to be used to **identify** and/or **authenticate** the **entity** that is bound to (any of) (the **identifiers in the** `id` **field of)** the binding-element.

a set of key-value pairs, where the keys are particular to the specified `binding`type and the values will be used as the method specified in the `type` field, so as to **identify** and/or **authenticate** the **entity**, as intended by the author of the `binding` property.

The semantics of the `binding` property is that if a party executes the method (or uses the mechanism) as specified by the `type` field, using the provided set of key-value pairs, then that **identifies** and/or **authenticates** a specific **entity**. If one more **identifier(s)** is specified in the `id` field, then each of these **identifiers** represents that specific **entity**.

In the following sections, we offer solutions on how to integrate the `binding` property into the current VCDM.

3.1.1 Add a `binding` property to a `credentialSubject` element

Our first proposal suggests adding the `binding` property to the contents of the `credentialSubject` element of the **VC** that has been issued by Ivan to attest that Alice has passed the exam for SOL (**VC** metadata and signature are omitted):

```
...
"credentialSubject": [ {
      "id": "somevaliduri", //optional
      "binding": [ <array of binding-elements> ],
      "hasPassedExam": "SOL"
   }
],
...
```

The `<array of binding-elements>` is the same as in the previous section (and left out for conciseness). Whenever a **verifier** needs to **identify/authenticate** a person as Alice, any of these elements provides a specific way determined by the **issuer** by which the **verifier** can do so, as long as he makes sure that the `id` field of the `credentialSubject` element matches one of the `id` field elements of the `binding`-element that it chooses to use for **identification** and **authentication** of Alice.

Note that if the `id` field were omitted, then the `binding` property could still be interpreted as a way to bind (**identify** and/or **authenticate**) the **claim**'s **subject**. The advantage of doing this is that the `binding` is more concise as it directly refers to the subject and no `id` fields are required, not only in the `credentialSubject` element, but also not in the `binding` array. However, this interpretation is inconsistent with the current VCDM, which says that whenever the `id` field in a `credentialSubject` element is missing, it is to be considered a 'bearer claim'[20], so a decision about this needs to be taken.

In the example above, the `binding` property is contained in the **claim** itself. This construct can be used to create **VCs** that people can use as pure **identification** and **authentication** credentials. They can also be used to construct **VCs** that contain

19The registry itself, where it is located, etc., is outside the scope of this paper.

claims, e.g. about things (containers, pets, etc. - where privacy issues do not play a large role). So when this mechanism is used in a `credentialSubject` element, it could be referred to as `subject binding`.

Thus, if a **VC/VP** were to contain properties such as `holder`, `presenter`, `issuee`, or others[21], this construct could also be used to enable **verifiers** to **identify** and/or **authenticate** the **entity** that (the value of) this `id` field refers to.

3.1.2 Using the `evidence` field

Our second proposal suggests placing the `binding` property in the `evidence` field (as being discussed e.g. in VCDM issue #902) with some minor modifications, as follows:

```
...
"credentialSubject": [
   {  "id": "somevaliduri", //mandatory
      "hasPassedExam": "SOL"
   }
],
"evidence": [ {
         "id": "somevaliduri", //mandatory
         "type": [ "Binding" , "didAuthenticationKey" ] ,
         "keyId": "did:example:deadbeefcafe#keys-3"
      }, {
         "id": "somevaliduri",
         "type": [ "Binding" , "passport" ] ,
         "nationality": "NL",
         "passportNr": 012345678,
         "contentHash": "3338be69 ... 2398f392"
      }, {
```

20 The VCDM text expresses this in terms of **VCs** rather than **claims**, but the intention is the same.

21 The discussion of whether or not to add such fields to a **VC** or **VP** is a good idea are outside the scope of this document.

```
          "id": "somevaliduri",

          "type": [ "Binding" , "portrait" ] ,

          "format" : "png",

          "portrait": "iVBORw0KGgoAAAANSUhEUgAAA8sAAAJl..."

       }]

}

...
```

In this example, the `"evidence"` property contains 1 or more elements, each of which enables verifiers to **identify/authenticate** the **entity** that is **identified** by a particular **identifier in the credentialSubject**. As the VCDM description of evidence property is very vague and allows lots of use cases, we propose to include `"binding"` in the array of types to distinguish from other forms of evidence.

3.1.3 Adding the `binding` property to the VCDM top level

Our third proposal suggests adding a new top-level `binding` property disjunct from the `evidence` field that comprises the array of `binding` elements. Therefore the contents for the proposed identifier `binding` would be the only content of the `binding` property and the 'evidence' property would be used for other use cases or concepts

```
...

"credentialSubject": [

   {  "id": "somevaliduri",

      "hasPassedExam": "SOL"

   }

],

"binding": [ {

          "id": "somevaliduri",

          "type": "didAuthenticationKey",

          "keyId": "did:example:deadbeefcafe#keys-3"

       }, {

          "id": "somevaliduri",
```

```
            "type": "passport",
            "nationality": "NL",
            "passportNr": 012345678,
            "contentHash": "3338be69 ... 2398f392"
        }, {
            "id": "somevaliduri",
            "type": "portrait",
            "format" : "png",
            "portrait": "iVBORw0KGgoAAAANSUhEUgAAA8sAAAJl..."
        }]
},
"evidence": {
     //whatever
}
...
```

3.2 Using identifier binding across multiple VCs

As described in the introduction, the **verifier** requires means to

- **identify** the **entity** that is the **subject** of a given **claim**
- **authenticate** the **entity** that is the **subject** of a given **claim**
- establish whether multiple subjects refer to the same **entity**, or not.

The first and second requirements need not be fulfilled by a single **claim** — nor even within a single **VC**. Under the (reasonable) assumption that an **issuer** will only use unambiguous **identifiers**, the value of `credentialSubject.id` (as used by a single **issuer**) will always refer to the same **entity**. Hence, such identifiers can be used in multiple **claims**, not only as a **subject identifier**, but also as an 'object identifier'; it may also appear in a `binding`-property, which can be part of e.g., a **VC** that is specifically intended to provide **verifiers** with the ability to **identify** and/or **authenticate** the **subject**.

While the first and second requirements are enabled through the `binding` property, the third requirement might need additional work or further explanations. When the **verifier** requests multiple **claims** of potentially multiple VCs, he will need to know how the **subjects** of these claims are related, and to ensure that he can establish such relationships. For relationships such as one **subject** being a parent (child, delegate, friend) of the other, this may seem obvious. However, the **verifier** must also be able to establish that the **subject** of one **claim** is identical to (the same as) that of the other **claim**. After all, the **entity**

that one **party** refers to with **identifier** X may be referred to with **identifier** Y by another **party**.

Here are some examples:

- Two **claims** are issued by the same **issuer** and they have the same `credentialSubject.id`. In this case, since the **identifier semantics** is that of a single **party** (the **issuer**), it is reasonable to infer that the **subject** of both **claims** is the same **entity**.
- Two **claims** (from two **VCs**) come from different **issuers** and the value of `credentialSubject.id` is the same for both **claims**. Since there are two **identifier semantics** involved (one for each of the **issuers**), the **verifier** needs additional information to establish that both **claims** have the same **subject** (or not). This also holds if both **claims** have other attributes (e.g., name, firstName, birthdate) that together form an **identifier** (this is limited to specific contexts and it gets complicated fast)

The assessment of whether two or more **claims** that originate from different **issuers** have the same **subject**, is a difficult matter that cannot be resolved in the context of the VCDM. Rather, it requires **verifiers** to make assumptions that they can ground, e.g., on legislation or governance frameworks that the **issuers** are subjected (or committed) to, or on experience, best practices, or a risk assessment.

4 How `binding` Types Work

This chapter describes various situations in which **identifier binding** should work, and shows that one or more of the solutions we propose actually work. We are demonstrating various options for Bob to realize his service offering using the concepts from previous chapters. None of these examples imply that certain `binding` types or decisions are “the best” way to implement a specific scenario. In fact, there will be many ways to ensure a `binding` and we want to show the different ways all using our proposed properties for the W3C VCDM.

4.1 DIDAuthentication

In this section we are showing a concrete example for the registration scenario “Alice registers herself” and the access scenario “Electronic access to the MOOC server”, used with the **identifier binding** type “DIDAuthenticationKey”.

In this example Alice got a VC from Ivan for her participation and successful exam on the course ‘SOL’ :

```
...
"credentialSubject": [ {
      "id": "https://universityoflogic.com/id/492754832663",
      "binding": [ {
         "type": "DIDAuthenticationKey",
         "didAuth": "did:jwk:123"
      } ],
      "hasPassedExam": "SOL"
   }
],
...
```

Alice wants to register at Bob’s web application for the course ‘MLAS’ and creates a **verifiable presentation** of the VC received from Ivan by authenticating with the DID `"did:jwk:123"`. Bob will check the validity of the VP and check the signature of Ivan. As the VC contains the claim `"hasPassedExam": "SOL"` Bob will enroll Alice for the course and issue her an enrollment VC **binding** it to `"did:jwk:456",` which Alice presented to Bob :

```
...
"credentialSubject": [ {
      "id": "https://universityofbob.com/id/399912",
      "binding": [ {
```

```
         "type": "DIDAuthenticataionKey",

         "didAuth": "did:jwk:456"

      } ],

      "isEnrolledFor": "MLAS-3"

   }

],

...
```

When Alice wants to start or continue the MOOC, she presents the enrollment VC to the login services (on behalf of Bob) and authenticates with the DID `"did:jwk:456"`. The login service verifies the **presentation**, including the `"DIDAuthenticationKey"` `binding`, checking for the claim `"isEnrolledFor": "MLAS-3"` and forwards Alice to the course material.

4.2 Out-of-band Binding / On-Site Portrait Holder Authentication

In this section we show a concrete example for the registration scenario "Alice registers herself" and the access scenario "Physical access to a physical location", used with the **identifier binding** types "passport" and "portrait".

4.2.1 Example 1

In this example Alice[22] got a VC from Ivan for her participation and successful exam on the course 'SOL' :

```
...

"credentialSubject": [ {

      "id": "https://universityoflogic.com/id/492754832663", //optional

      "binding": [ {

         "type": "passport",

         "nationality": "NL",

         "passportNr": 012345678,

         "contentHash": "3338be69 ... 2398f392"

      }, {

         "type": "portrait",
```

22 This use case can also easily be done by Trevor.

```
         "format" : "png",

         "portrait": "iVBORw0KGgoAAAANSUhEUgAAA8sAAAJl..."

      } ],

      "hasPassedExam": "SOL"

   }

],

...
```

Alice wants to register at Bob's web application for the course 'MLAS' and presents the VC received from Ivan. Bob will check the validity of the VC and check the signature of Ivan. As the VC contains the claim `"hasPassedExam": "SOL"` Bob will enroll Alice for the course. When the physical course starts at the university the teacher (on behalf of Bob) will bring a list of Passport numbers and/or portrait pictures and compare those to the people entering the course room.

4.2.2 Example 2

In this example Alice got a VC from Ivan for her participation and successful exam on the course 'SOL' :

```
...

"credentialSubject": [ {

      "id": "https://universityoflogic.com/id/492754832663", //optional

      "binding": [ {

         "type": "passport",

         "nationality": "NL",

         "passportNr": 012345678,

         "contentHash": "3338be69 ... 2398f392"

      }, {

         "type": "portrait",

         "format" : "png",

         "portrait": "iVBORw0KGgoAAAANSUhEUgAAA8sAAAJl..."

      } ],
```

```
      "hasPassedExam": "SOL"

   }

],

...
```

Alice wants to register at Bob's web application for the course 'MLAS' and presents the VC received from Ivan. Bob will check the validity of the VC and check the signature of Ivan. As the VC contains the claim `"hasPassedExam": "SOL"` Bob will enroll Alice for the course and issue her an enrollment VC:

```
...

"credentialSubject": [ {

      "id": "https://universityofbob.com/id/399912", //optional

      "binding": [ {

         "type": "passport",

         "nationality": "NL",

         "passportNr": 012345678,

         "contentHash": "3338be69 ... 2398f392"

      }, {

         "type": "portrait",

         "format" : "png",

         "portrait": "iVBORw0KGgoAAAANSUhEUgAAA8sAAAJl..."

      } ],

      "isEnrolledFor": "MLAS-3"

   }

],

...
```

When the physical course starts at the university, Alice brings her enrollment VC and presents those to the teacher (on behalf of Bob). In the case of the Out-Of-Band `binding` Alice presents her physical passport and the teacher matches the

physical passport with the data from the enrollment VC containing the claim `"isEnrolledFor": "MLAS-3"`. In the case of 'On-Site Portrait Holder Authentication' the teacher compares Alice's face with the image from the VC.

4.2.3 Example 3

In this example Alice has two VCs in her wallet.

The first is a national ID card VC from a government issuer, stating typical personal identification information and an **identifier binding** type for On-Site portrait holder authentication:

```
"credentialSubject": [ {

      "id": "https://stateissuer.nl/id/9021678534", //mandatory

      "binding": [ {

         "type": "portrait",

         "format" : "png",

         "portrait": "iVBORw0KGgoAAAANSUhEUgAAA8sAAAJl..."

      } ],

      "firstName": "Alice",

      "familyName": "from Wonderland",

      ...

   }

],

...
```

The second is a course participation VC from Ivan on her successful exam on the course 'SOL'. This credential does not have an **identifier binding** of itself, but links to the `binding` of the first VC (this also illustrates the req3).

```
...

"credentialSubject": [ {

      "id": "https://universityoflogic.com/id/492754832663",

      "binding": [ {

         "type": "linkedBinding",

         "link": "https://stateissuer.nl/id/9021678534"
```

```
      } ],

      "hasPassedExam": "SOL"

   }

],

...
```

Alice wants to register in Bob's web application for the course 'MLAS' and presents the VC received from Ivan. Bob will check the validity of the VC and check the signature of Ivan. As the VC contains the claim `"hasPassedExam": "SOL"` Bob will enroll Alice for the course and issue her an enrollment VC, copying the linked **identifier binding**:

```
...

"credentialSubject": [ {

      "id": "https://universityofbob.com/id/399912",

      "binding": [ {

         "type": "linkedCredential",

         "link": "https://stateissuer.nl/id/9021678534"

      } ],

      "isEnrolledFor": "MLAS-3"

   }

],

...
```

When the physical course starts at the university, Alice presents both the national ID card and enrollment VC to the teacher (on behalf of Bob). In this example the teacher checks the linkage of both VCs, uses the 'On-Site Portrait Holder Authentication' from the first VC and compares Alice's face with the image from the `binding` mechanism. Note that this is a simplified example and precautions must be made to genuinely identify the correct linked **identifier binding**, such that the issuer of such VC is also securely linked, e.g. by identifiers of issuer and credential.

4.3 Remote Holder Authentication for Mobile Secure Wallet (DIF)

In this section we are showing a concrete example for the registration scenario "Alice registers herself" and the access scenario "Access to an online video course run by a real person", used with the **identifier binding** type "RemoteHolderAuthentication".

In this example Alice has two VCs in her wallet. The first is a national ID card VC from a government issuer, stating typical

person identification information and an **identifier binding** type for remote holder authentication. During issuance the holder's wallet was authenticated and the VC bound to a hardware-backed key with provided holder authentication:

```
"credentialSubject": [ {

      "id": "https://stateissuer.nl/id/9021678535", //mandatory

      "binding": [ {

         "type": "secureWalletRemoteBindingDIF",

         "walletName": "Example Wallet", //optional

         "walletVersion": "1.3.0", //optional

         "hardwarePublicKey": "did:jwk:123", //links and other formats possible

         "holderAuthentication": ["FaceID", "PIN"]

      } ],

      "firstName": "Alice",

      "familyName": "from Wonderland",

      ...

   }

],

...
```

The second is a course participation VC from Ivan on her successful exam on the course 'SOL'. This credential does not have an **identifier binding** of itself, but links to the `binding` of the first VC (this also illustrates the req3):

```
...

"credentialSubject": [ {

      "id": "https://universityoflogic.com/id/492754832663",

      "binding": [ {

         "type": "linkedBinding",

         "link": "https://stateissuer.nl/id/9021678535"

      } ],
```

```
      "hasPassedExam": "SOL"

   }

],

...
```

Alice wants to register at Bob's web application for the course 'MLAS' and presents the VC received from Ivan. Bob will check the validity of the VC and check the signature of Ivan. As the VC contains the claim `"hasPassedExam": "SOL"` Bob will enroll Alice for the course and issue her an enrollment VC, copying the linked **identifier binding**:

```
...

"credentialSubject": [ {

      "id": "https://universityofbob.com/id/399912",

      "binding": [ {

         "type": "linkedCredential",

         "link": "https://stateissuer.nl/id/9021678535"

      } ],

      "isEnrolledFor": "MLAS-3"

   }

],

...
```

When the course starts in the video conference of Bob's university platform, Alice presents both the national ID card and enrollment VC to the web application. The authentication is locally enforced by her wallet according to the binding mechanism. Bob's verifier component then checks the validity of both VCs, checks the linkage and lets Alice enter the video conference.

5. Considerations

This section contains a first set of topics that implementers and others need to consider. It is not intended to be complete; it simply points out some concerns that need to be (better) addressed.

5.1 Implementer Considerations

The **issuer** (author of identifiers) MUST control the **identifier semantics** of all **identifiers** that it uses in **claims** within every **VC** that it issues. This implies that an **issuer** that chooses to use DIDs for such **identifiers**, MUST control these DIDs.[23]

Attributes that are used for specific binding-types may be PII, so this will typically require selective disclosure in order to maintain privacy. It also requires caution when transferring VCs between different parties — to be elaborated. PII might be transferred to places where it should not come (how to prevent that?). Also, there may be nice identification attributes, e.g. a nationality + passport number.

There are a number of SSI frameworks or libraries that provide functions for verifying **VCs** and **VPs**. Examples include but are not limited to DIDKit, vc.js, and Veramo. Those implementations typically have a plugin mechanism to wire up additional proof types and other extension points of the VCDM to add support for these mechanisms.

Largely, those frameworks or libraries do not provide functions for verifying identifier binding because the VCDM does not define such a concept. This leads to custom implementations which are prone to errors and which result in challenges regarding interoperability across the different components involved in typical SSI flows. The proposed approach would allow frameworks or libraries to define functions for verifying identifier binding in a similar way to verifying verifiable presentations. Each specific binding method could be defined as a plugin and **verifiers** may use the ones that are fit for their specific purpose. Also, since a binding can be part of an individual **VC**, **verifiers** can ask for **VCs** that include bindings of the type they need. The result is that the SSI application developers have to provide less code to implement this very common business requirement by just registering the identifier binding methods that are also less prone to errors and increases interoperability across the entire SSI ecosystem.

5.2 Privacy (and Other) Considerations

Concerns have been raised saying that "holder-binding may have unanticipated privacy & correlation issues" and also that "holder-binding may be an entree for parties to create centralization or lock-in, or worse, create human-rights issues" (VCDM issue #988).

The guidance that the VCDM already provides in its sections on privacy considerations, security considerations, etc. are equally applicable to the contents of the `binding` properties. There is nothing very special about them, except perhaps that it may be more important that holders are able to selectively disclose them.

What might be a topic to consider is that, when a VC is transferred from one **party** to another, there would be a way to obscure some of the `binding` elements so that the **party** to which the VC is transferred does not learn all the means by which the enclosed **identifiers** can be dereferenced. One mechanism could be to use the third option of our proposal, i.e. **issuing** the binding properties as stand-alone **VCs**, and RECOMMEND that **issuers** would only **issue** these **VC**s to a **wallet** that is an **agent** of the **party** that is or owns the **entity** to which the **identifier** refers and to which the binding properties apply.

With regards to creating centralization or lock-in, this does not seem to be a problem as we only propose that the **issuer** provides means to other **parties** to dereference the **identifiers** that it has authored, for which it is the (sole) authority.

This does not imply that there may not be any issues left, waiting to be found and addressed. However, as this paper is intended to contribute to the holder-binding discussion rather than to provide the final solution, we will leave this topic as future work.

23 This contradicts the SSI principle that holders should control their identities. However, it is inevitable as the DID spec says that a **party** that controls a DID gets to determine its **identifier semantics** (i.e.: the DID subject).

6. Conclusion and Future Work

VCDM issue #902 shows that 'holder binding' is not only quite contentious, but also that many members of the SSI community have a habit of using terms that are defined in the VCDM, yet in other meanings as they are defined. While we cannot prevent readers from doing the latter, we can provide them with criteria by which they can determine themselves whether or not something is an instance (example) of a term we use. Taking the VCDM terminology as a starting point, we clarified what we thought was appropriate, and added some terms we deemed necessary. Any term that is used in this paper and as a definition in the terminology section, is used in the way it is defined there. Outside the scope of this paper, terms are expected to have other meanings as well. Whether or not this practice is followed elsewhere, is a useful discussion but outside the scope of this paper.

Using our terminology, and looking at use-cases in the real world, we have found that the typical use-cases that claim to need 'holder binding', i.e. in which the **verifier** needs to know who the **holder** is, are in fact specializations of use-cases in which it would be perfectly acceptable for the **verifier** to *not* know who the **holder** is, typically because while a **party** may transact on its own behalf, another **party** may conduct that same transaction on the first **party**'s behalf.

The main requirements that **verifiers** have in such cases are their need for a capability to:

- **identify** the **entity** that is the **subject** of a **claim** (typically the binding between **subject** and **subject identifier**),
- **authenticate** the **entity** that is the **subject** of a **claim** (typically binding between the **subject** and the **claim**), and
- establish whether the **subject** of two **claims** (authored by the same or different **parties**)
    - are in fact the same **entity**,
    - are different **entities**, or
    - are **entities** for which it cannot be determined that they are the same or different.

Currently, **verifiers** do not have such capabilities. The result in practice is that they revert to means such as (tacit) assumptions that have no basis in specifications. For example, to **authenticate** the **entity** that is **identified** by a DID that shows up as a **subject identifier** (i.e. as the value of the `id` field in one of the `credentialSubject` elements of a **VC)**, which is not compliant with what the DID spec says. Examples such as these, and the fact that such practices are defended, clearly demonstrate the need for properly discussing and standardizing means that contribute to providing such **verifier** capabilities.

Our paper proposes a generically useful property that we call `binding`, which aims to make such contributions. We specify an extendable syntax and have drafted a specification for its semantics. This `binding` property is not only useful to 'bind' an **entity** to a **claim** as its **subject**, but can also be used to 'bind' an **entity** that is related to the **subject** of the **claim** by means of a predicate (one might call this `object binding`). Also, if ever the properties of `holder` or `issuee` were to be standardized (which is totally unnecessary once the `binding` property is used), then the **entities** fulfilling such roles could be 'bound' by the binding property.

We show how this property can be used in **VCs** and **VPs**, in various ways, both in online and offline use-cases. This is shown in various operational scenarios of the same use case. This also shows that, depending on the kind of operational scenario, **verifiers** typically have additional, scenario-specific needs that result from their business-thinking, and are reflected in their business rules.

A particularly important addition comes from the fact that it is typically not **parties** that interact with each other, but that each of them also uses some (often IT) **components** to contribute to that interaction on their behalf. **Users** would typically employ a mobile phone or tablet, which in practice may be shared with others. A **verifier** whose business rules state he MUST know who the **party** is that operates a **component** that has sent a **VC**/**VP**, may need to establish the relation between that IT **component** and this **party** on whose behalf it operates, which is a topic that we consider future work.[24]

24 We might look at what it takes for such **components** to issue **claims** that state who the **party** is on whose behalf it operates, and how a **verifier** can trust that **component** to make such claims.

Additional Credits

Lead Author: Oliver Terbu

Authors: Paul Bastian, Rieks Joosten, Zaïda Rivai, Oliver Terbu, Snorre Lothar von Gohren Edwin, Antonio Antonino, Nikos Fotiou, Stephen Curran, and Ahamed Azeem

Sample APA Citation

Bastian, P., Joosten, R., Rivai, Z., Terbu, O., Edwin, S. Antonino, A., Fotiou, N., Curran, S., and Azeem, A. (2023). Identifier Binding: defining the Core of Holder Binding. Rebooting the Web of Trust XI. Retrieved from https://github.com/WebOfTrustInfo/rwot11-the-hague/blob/master/final-documents/identifier-binding.pdf.

About Rebooting the Web of Trust

This paper was produced as part of the Rebooting the Web of Trust XI design workshop. On September 26th to 30th, 2022, over 60 tech visionaries came together in The Hague, The Netherlands to talk about the future of decentralized trust on the internet with the goal of writing at least 5 white papers and specs. This is one of them.

• **RWOT Board of Directors:** Christopher Allen, Joe Andrieu, Erica Connell.

• **RWOT11 Coordination Team:** Will Abramson, Christopher Allen, Joe Andrieu, Shannon Appelcline, Erica Connell, Eric Schuh, Carsten Stöcker.

• **Workshop Credits:** Will Abramson (Producer), Christopher Allen (Founder), Shannon Appelcline (Editor-in-Chief), Erica Connell (Host), Amy Guy (Ombudsperson), Willemijn Lambert (Graphic Recorder), Eric Schuh (Ombudsperson), Carsten Stöcker (Co-Producer, Demo Organizer), Dorothy Zablah (Facilitator).

• **Gold Sponsors:** The City of the Hague, Digital Contract Design, Dutch Blockchain Coalition, The Hague University of Applied Sciences, eSSIF-Lab.

• **Contributing Sponsors:** Blockchain Commons, Legendary Requirements, Spherity.

Thanks to all our attendees and other contributors!

What's Next?

The design workshop and this paper are just starting points for Rebooting the Web of Trust. If you have any comments, thoughts, or expansions on this paper, please post them to our GitHub issues page:

https://github.com/WebOfTrustInfo/rwot11/issues

The twelfth Rebooting the Web of Trust design workshop is scheduled for late 2022. If you'd like to be involved or would like to help sponsor the event, email: Leadership@WebOfTrust.info