Binding Type Implementations

TNO Publiek › TNO 2023 R12079
2 November 2023

ICT, Strategy & Policy
www.tno.nl
+31 88 866 00 00
info@tno.nl

TNO 2023 R12079 – 2 November 2023

Binding Type Implementations

Author(s) Anne Nijsten, Ward van der Schoot, João Diogo de Faria Miranda Duarte and Thijs Laarhoven

Classification report TNO Publiek
Classified by Classification Classified By
Classification date Classification Date Classification validity period
Title TNO Publiek
Managementuittreksel TNO Publiek
Summary TNO Publiek
Report text TNO Publiek
Number of copies 1
Number of pages 20 (excl. front and back cover and distribution list)
Number of appendices 0
Sponsor Ministerie van Justitie en Veiligheid
Programme name VP Veilige Maatschappij
Programme number VPVM
Project name FUTURE-PET
Project number 060.46534

Managementuittreksel

Programma

Programmanaam:
VP Veilige Maatschappij

Programmanummer: VPVM

Project

Projectnaam:
FUTURE-PET

Projectnummer: 060.46534

Summary

The field of Self-Sovereign Identity (SSI) aims to develop solutions in which end users can prove characteristics about themselves while retaining full control of their identity. One of the current issues in SSI is given by the problem of *subject-identifier binding*: how does a verifier assess whether a certain entity is the subject of a given credential, and how does a verifier assess whether two credentials contain the same subject? Recently, Bastian et al. have proposed a solution to this problem in the form of the *binding property*. This document serves as an extension to this proposal, containing a practical explanation of how such a binding property could be implemented in various settings, and how it would solve the issue of subject-identifier binding.

0 Terminology

In this document, the same terminology is used as in the document by Bastian et al. [1].

Contents

1 Introduction

SSI is an approach to digital identity that allows each individual to remain in control over its own identity at any moment. By using SSI, individuals can choose which information to share when they need to prove their identity for services, websites and more when requested by a verifier. Note that it is still up to a verifier to determine whether the information provided by the individual is sufficient for its intended verification purpose. Currently, applications for SSI are found in more and more use cases , with corresponding implementations. Famous examples of how to use SSI are: proving you are at least 18 years old without revealing your specific age; proving eligibility for a mortgage without revealing exact income and savings; proving citizenship of an EU country without revealing all passport information, or even nationality.

This research was originally conducted to enable the use of SSI in the 'Justitie & Veiligheid' domain, however, the obtained result offers a solution for a generic SSI problem.

In its simplest form, the workings of SSI can be summarised as follows. Parties can take on different roles depending on what a party does at a given moment. Issuer, holder and verifier are the roles that are the core of SSI. The main building block of SSI is given by verifiable credentials, which contain information on a certain entity, called the subject, and certain attributes regarding this subject. Credentials are given out by issuer parties to other parties, which then become the holder of the credential. Note that this holder does not necessarily need to be the subject of this credential. Lastly, there is the verifier role, which aims to verify certain properties regarding a credential, when information from this credential is presented to the verifier by a certain entity in the form of a verifiable presentation.

In theory, this allows an entity to prove properties, as established by a certain issuer, about itself to a verifier while remaining in control of the credentials detailing these properties. However, these basic workings bring an issue with them, namely the problem of subject-identifier binding. This problem is best summarised by the following two questions:

1. How does a verifier assess whether a certain entity is or is not the subject of a presented verifiable credential?
2. How does a verifier, when being presented with two different credentials, assess whether the two credentials have, or do not have, the same subject?

1.1 Binding property

A potential solution to this problem was raised by Bastian et al. in [1], called the *binding property*. The binding property is a type of attribute that can be added to a credential and specifies how a verifier can check whether a certain entity is the subject of a credential. A binding property attribute contains the following fields:

- (Optional) Id: A string specifying the attribute
- Type: A name of a mechanism specifying how the binding property can be used to check whether a certain entity is the subject of the credential

- Key: A certain key value which needs to be checked by the specified mechanism.

When a verifier is presented with a credential containing a binding type, the verifier can read out this binding type and look up the related method in a register. Now, given a certain entity, the verifier can assess whether this entity is the subject of the credential by performing the specified mechanism and checking whether the entity can fulfil the required key value.

If the binding type is designed such that only the subject of the credential can fulfil the required key value under the specified mechanism, the binding property gives an answer to question 1 in full. While this might be the case for some binding types, it is likely not to be the case for all binding types. In such cases, the difficulty of fulfilling the required key value under the specified mechanism while not being the subject of the credential, governs to what degree of certainty the specific binding property gives an answer to question 1. This will become clear in chapter 2. An explanation as to how the binding property helps in answering question 2 can be found in chapter 3.

It should be noted that binding types are not limited to cases in which the holder has the be the subject of the credential to fulfil the binding type property. Examples are parents representing their children, or persons representing an organization. For clarity reasons, however, this work will only speak about the binding type in terms of a holder being able to prove it is the subject of a credential. The findings below still hold in other scenarios in which the binding type applies just as well.

1.2 Zero-Knowledge Proofs

In the use of binding types, an entity often needs to prove knowledge or ownership of the key value in some way or another. Occasionally, this key value is a secret value, as otherwise the subject of the credential would not be the only entity being able to prove knowledge of this key value. For the sake of privacy, an entity would hence like to keep this key value secret. However, to establish binding using this binding type, the subject needs to present this secret value to a verifier. This gives an apparent contradiction: an entity wants to keep this key value secret, while it needs to prove knowledge of this value to convince a verifier that it is the subject of the credential.

The solution to this issue lies in the so-called Zero-Knowledge Proofs (ZKPs). Zero-Knowledge-Proofs are a family of cryptographic functionalities which allow a prover to prove knowledge of certain information, without revealing it. This is exactly the sort of functionality we need to resolve the above issue. Zero-Knowledge Proofs can be used to prove a wide variety of tasks. At the least, Zero-Knowledge Proofs can be used to prove any statement which can be written as an arithmetic circuit, including all statements which can be solved on a computational device.

Zero-Knowledge Proofs have been around in literature for many years, but the step towards practical implementation has only recently been made. Because of this, there are only a limited number of organisations offering ZKP solutions and practical implementations of ZKPs are still relatively immature. In particular, most ZKP solutions are currently relatively slow. However, there has recently been a significant increase in the speed of ZKP implementations, as well as the number of companies which offer practical ZKP implementations.

In this work, we will explain how ZKPs can help preserve privacy in binding types, as well as how they can be used to answer question 2 above. To demonstrate its use, one binding type will be worked out in complete detail using a public ZKP library.

1.3 Structure

The rest of this document will consist of two parts. Firstly, in chapter 2, we will populate the binding property with different binding types. This includes a clarification on how the relevant information for establishing the binding is transferred to a subject. Implementations with the use of Zero-Knowledge Proofs will be discussed as well. Subsequently, in chapter 3, we will indicate how the binding property yields a potential answer to question 2 when it is used in combination with Zero-Knowledge Proofs. In addition, one of the binding types will be worked out in even more detail using a public ZKP library in chapter 4, showing an actual demonstration of binding types in practice. Lastly, we will discuss the obtained results and propose a further outlook in chapter 5.

2 Binding Property Specification

In this chapter, we list a couple of examples of binding types and how they can be used in practice. Firstly, for each binding type, we specify it according to the notation used by Bastian et al. [1]. Secondly, the process of establishing the binding type in a credential is discussed in the setup phase. This consists of some form of communication between subject and issuer and results in the issuer obtaining the information required for the binding type. The step in which an issuer provides a holder with the credential is left implicit. Lastly, a specification is listed for how the binding type can be used by a verifier to match an entity to a certain claim.

For both the setup phase and the verification phase there are typically different options available. When applicable, itemized lists for the same step detail the different options that are available. As mentioned before, no binding type gives full assurance that a holder satisfying the binding property is the subject of the corresponding credential. It is up to the verifier to assess whether the given binding type and options used within the binding type give sufficient assurance for the use case. In particular, different options for a given binding type generally give varying levels of assurance.

2.1 Email address

```
"binding": [ {
  "id": ["someID"],
  "type": "accessToEmail",
  "emailaddress": "someEmailAddress"
} ],
```

Setup

- Issuer provides subject with access to mailbox corresponding to `someEmailAddress`.
- Subject provides issuer with email address `someEmailAddress`.

Verificiation

A verifier sends an email to `someEmailAddress` containing a unique passcode. If an entity can provide the passcode, it is identified as the subject of the verifiable credential.

2.2 Address

Can be implemented analogously to the above binding properties specified for email, but with physical mail instead of email.

2.3 Phone number

Can be implemented analogously to the above binding properties specified for email, but with text messages or calls instead of email.

2.4 Knowledge of Private Key

```
"binding": [ {
  "id": ["someID"],
  "type": "knowledgeOfPrivKey",
  "pubkey": "somePubKey"
} ],
```

Setup

Subject provides a public key "somePubKey", of which it has the corresponding private key.

Verificiation

The entity gives a zero knowledge proof of the knowledge of the private key corresponding to "somePubKey".

2.5 Passport

```
"binding": [ {
  "id": ["someID"],
  "type": "show_passport",
  "number": "1345208jvdsakouh9e431"
} ],
```

Setup

Government provides subject with passport. The issuer is then provided with the passport number of the subject, e.g. in one of the following ways.

- Subject shows issuer passport
- Subject provides issuer with passport number (note that using this is a less reliable option than showing the passport, but may be deemed acceptable in some cases)
- Issuer scans passport of subject

Verificiation

The passport number of an entity is provided to the verifier, e.g. in one of the following ways.

- Entity shows passport to verifier
- Entity gives passport number to verifier
- Verifier scans passport of entity

When the passport number corresponds to the value of `number`, the entity is identified as the subject of the verifiable credential.

2.6 Nationality

```
"binding": [ {
  "id": ["someID"],
```

```
  "type": "show_nationality",
  "nationality": "Dutch"
} ],
```

Setup

Government provides subject with passport. The issuer is then provided with the nationality of the subject, e.g. in one of the following ways.

- Subject shows issuer passport
- Issuer scans passport of subject

In case of a non-English passport, the issuer translates the nationality to the English language.

Verificiation

The verifier checks the nationality of an entity as stated in their passport, e.g. in one of the following ways.

- Entity shows passport to verifier
- Verifier scans passport of entity

When the nationality, possibly translated to the English language, corresponds to the value of `nationality`, the entity is identified as a subject of the verifiable credential.

2.7 Fingerprint

```
"binding": [ {
  "id": ["someID"],
  "type": "show_fingerprint",
        "program": "fingerprintComparer",
  "fingerprint": "j489ois46nioghzg6568…"
} ],
```

Setup

- Government provides subject with passport containing fingerprint information of subject. Issuer scans passport of subject and copies fingerprint value to key value in binding type.
- Issuer uses fingerprint scanner to make a scan of a fingerprint of a subject, this is saved as a string value.

Verification

Depending on the method chosen for setup above:

- Verifier scans passport to obtain fingerprint information. When the obtained fingerprint corresponds to the value of `fingerprint`, which can be checked by the fingerprintComparer program, the entity is identified as the subject of the verifiable credential.
- Verifier uses the fingerprint scanner to make a scan of the fingerprint of an entity. If this is close enough to the `fingerprint` value as determined by the `fingerprintComparer` software, the entity is identified as the subject of the verifiable credential.

2.8 Portrait

This is analogous to the fingerprint scan binding property, but possibly with a visual check instead of a check by a program.

2.9 DID

```
"binding": [ {
  "id": ["someID"],
  "type": "DIDAuthenticationKey",
  "did": "did:example:j489ois46nioghzg6568"
} ],
```

Setup

A DID controller issues a DID document for a DID subject, which is the subject of the VC made by the issuer. Subject provides issuer with the key id for the `did` field.

Verificiation

A verifier ensures that an entity has the private key material associated with the value of `did` to confirm that it is the subject of the VC.

3 Implementation of Knowledge of Private Key Binding Type

In this section, a detailed use of the Knowledge of Private Key Binding type will be given. This detailed implementation uses the open source ZKP library ZoKrates to write out the zero-knowledge proofs. The library was chosen after a case study on ZKP libraries, which can be found in the ZKP libraries document from the 2023 FUTURE PET project. While the library itself is easy to use once it is set up, in its current form setup requires knowledge and means to deploy smart contracts on the Ethereum blockchain.

3.1 Setting

In this example, we consider an entity Edward who is receiving a credential from issuer Ivan, which will later be verified by verifier Verity. To set up a binding between Edward and the credential, Ivan will use the "knowledge of private key" binding type. The full practical implementation of the protocol will be discussed below, split into three parts: the issuing phase, the verifying phase, and the zero knowledge proof. It should be noted this is also the sort of binding type that is used in two well-known credentials, namely AnonCreds [2], and the credentials given out by the SSI-app Yivi [3], which uses the IRMA protocol [4].

3.2 Issuing phase

In the issuing phase, Edward and Ivan first establish that Edward should indeed be granted the relevant credential. In addition, Edward provides Ivan with a public key, for which only he knows the corresponding private key. For Ivan to be sure of this, Edward provides Ivan with a zero knowledge proof of this knowledge, which can be seen below. At the end of this phase, Edward is provided with the following credential:

```
{...,
"binding": [ {
  "id": ["someID"],
  "type": "knowledgeOfPrivKey",
  "pubkey": "somePubKey"
} ],
...}
```

3.3 Verifying phase

Then, in the verifying phase, Verity reads out the credential information provided by Edward (via a verifiable presentation). To make sure that Edward is actually the subject of the credential, Verity reads out the binding type of the credential. As the binding type is knowledge of private key, Verity only needs to check that Edward knows the private key corresponding to `somePubKey`. Again, this can be realized by letting Edward provide a zero

knowledge proof of this knowledge to Verity. Verity concludes that Edward is the subject of the credential, and the desired functionality is achieved.

3.4 Zero-Knowledge Proof of knowledge of private key

The protocol hinges on the Zero-Knowledge Proof that Edward knows the private key corresponding to `somePubKey`. For this example, the open-source ZoKrates toolbox will be used. ZoKrates is a toolbox which provides a way to use zkSNARKs on the Ethereum blockchain in the form of imperative programs. The acronym zkSNARK stands for Zero-Knowledge Succinct Non-Interactive Argument of Knowledge, which is a zero knowledge proof that does not require interaction between a prover and a verifier. ZoKrates supports elliptic curve operations, like proving knowledge of a private EdDSA key given the public key. We assume that in our example Edward uses EdDSA keys. The code below is derived from ZoKrates Proof of Private Key Ownership · GitHub.

The code below details how Edward generates a Zero Knowledge Proof for a party (in the above example either Ivan or Verity) of knowing the private key corresponding to `somePubKey`. To generate the proof, the following function needs to be defined:

```
import "ecc/edwardsAdd.code" as add
import "ecc/edwardsScalarMult.code" as multiply
import "utils/pack/unpack256.code" as unpack256
def main(field[2] pubkey, private field seckey, field[10] curveparam) ->
(field):
	field[2] G = [curveparam[4], curveparam[5]]
	field[256] seckeybits = unpack256(seckey)
	field[2] ptExp = multiply(seckeybits, G, curveparam)
	field output = if ptExp[0] == pk[0] && ptExp[1] == pk[1] then 1 else 0
	fi
	return output
```

Here, input fields indicated with private contain information that will not be revealed to a verifier like Verity. ZoKrates outputs a proof that the entity that provided the input knows the secret key related to the public key. Now Edward compiles the code via

```
zokrates compile -i <program_name>.zok
```

Afterwards Verity starts the setup phase via

```
zokrates setup
```

This creates a verification key and proving key, which should be provided to Verity and Edward, respectively. Verity can then create a smart contract with

```
zokrates export-verifier
```

This contract contains the verification key and a verification function. This smart contract should be deployed on the Ethereum blockchain by Verity. Edward then provides the correct arguments to the program

```
zokrates compute-witness -a <pubkey> <seckey> <curveparam>
```

Lastly, Edward generates the proof via

```
zokrates generate-proof
```

This proof contains three elliptic curve points which are used by the verification function in the smart contract to get accepted when Edward calls this function. Further, it contains the public input and return values of the main function. In the meantime, Verity keeps an eye on the smart contract for the return value of this verification function call.

Note that Verity is now convinced that someone knows the private key, but may not know who this party is except when he knows the party behind the public address on the blockchain. Edward should thus also disclose its public address on the blockchain to Verity. More information about ZoKrates can be found at the following links:

Introduction - ZoKrates

Proving knowledge of a hash preimage - ZoKrates

Building Identity-linked zkSNARKs with ZoKrates | by Jacob Eberhardt | ZoKrates | Medium

4 Binding Types as an Answer to Question 2

Recall question 2 from the introduction:

How does a verifier, when being presented with two different credentials, assess whether the two credentials have, or do not have, the same subject?

If the credentials were to contain binding types, there are a couple of ways in which the binding property can help answer question 2. It should be noted in advance that the binding property in no way gives an answer to question 2 in full, but only potentially give an answer in a couple of scenarios. The specific scenarios will be listed, and for each scenario an explanation will be given as to how the binding property can help answer question 2.

1. The two credentials have the same binding and are issued by the same issuer

If the issuer hands out unique ids for each binding type instance and hands out the same id for the same binding type instance each time, a simple equality check between the id fields of the two binding types would be sufficient, assuming the key values are unique for different entities. If the issuer does not give out the same id each time, but it does use the same key-value pair(s) for a given binding type instance, an equality check between the key values could be used, again assuming the key values are unique for different entities. Both of these equality checks can be done using a Zero-Knowledge Proof if the holder wishes to keep the ids or key values secret. Note that this solution is in no way unique to binding types, and can be used with any attribute having a unique field.

2. The two credentials have the same binding type and are issued by different issuers

In this case, the key value from the binding types may be used in the same way as in solution 1. Note however that this only works if the different issuers use the same key value in the binding type instance. Again, this solution is not unique to binding types and can be used for attributes. However, as the key value in the binding type could be a secret value, it will often be a unique value for each subject. For the examples above, it can be expected that different issuers will often use similar key values for a given binding type mechanism. However, the level of similarity and the options that gives for interoperability should be judged on a case by case basis.

3. A certain entity presents the two credentials to the verifier

This scenario is different and is actually specific to the binding type attribute. If an entity presents the two credentials to the verifier, the verifier can use the two binding types of the credential to assess whether the entity is the subject of the credentials. If this is the case for both, the verifier can conclude that the credentials have the same subject. If this is the case for only one, the verifier can conclude that the credentials do not have the same subject. If this is the case for neither one, the verifier cannot conclude whether the subjects of the credentials are the same or different. It should be noted that all the assertions in this scenario are within the confidence that the respective binding types give.

5 Conclusion

In this work, we have extended upon the original proposal of binding types by Bastian et al. in a couple of ways. Firstly, we have listed a variety of potential binding types together with their possible implementations. In addition, we have shown how binding types could be implemented in a secure way through the use of zero-knowledge proofs. To make this concrete, we have even given a detailed implementation of such a zero-knowledge-proof implementation of an often-used binding type. Lastly, we have indicated how the binding type works to solve some of the issues related to subject-identifier binding. We hope that this work strengthens the knowledge on binding types and brings us closer to the practical implementation of binding types at their maximum potential.

6 Bibliography

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

[2] Hyperledger Foundation, „AnonCreds Specification," 2023. [Online]. Available: https://github.com/hyperledger/anoncreds-spec.

[3] „Yivi," Yivi, 2023. [Online]. Available: https://www.yivi.app/.

[4] „Irma protocol," Irma docs v0.14.0, 2023. [Online]. Available: https://irma.app/docs/irma-protocol/.

Distribution list

ICT, Strategy & Policy

Anna van Buerenplein 1
2595 DA Den Haag
www.tno.nl

TNO innovation for life