ΟΙΚΟΝΟΜΙΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ ΑΘΗΝΩΝ

ATHENS UNIVERSITY OF ECONOMICS AND BUSINESS

School of Information Sciences and Technology
Department of Informatics
Athens, Greece

Master Thesis
in
Computer Science

Verifiable Credentials Selective Disclosure, Challenges and Solutions

Vasilis Kalos

Supervisor: Prof. George C. Polyzos
Department of Informatics
Athens University of Economics and Business

Committee: Prof. Vasilios A. Siris
Department of Informatics
Athens University of Economics and Business

Asst. Prof. Spyridon Voulgaris
Department of Informatics
Athens University of Economics and Business

October 2021

Vasilis Kalos

Verifiable Credentials Selective Disclosure, Challenges and Solutions

October 2021

Supervisor: Prof. George C. Polyzos

Athens University of Economics and Business

School of Information Sciences and Technology

Department of Informatics

Mobile Multimedia Laboratory

Athens, Greece

Abstract

Verifiable Credentials (VCs) provide a way for entities (users, applications, devices etc.) to prove claims, like a user's name and occupation, a device's manufacturer, an applications security level etc. In the VCs ecosystem, there are three main entities; the Issuer that creates the VC (e.g., an airline company that issues a ticket), the Holder that controls the VC (e.g., a traveler that bought the ticket) and the Verifier that checks the validity of that VC (e.g., the airport that validates the ticket).

Although VCs have been met with great enthusiasm, there are still some open problems that need to be addressed. The biggest of those issues is the disclosure of the user's private information to anyone they send their VC. Credentials are likely to contain information that the Holders will want to keep private, like their place of residence, medical records etc. Most of the current implementations however, use traditional digital signatures that require the disclosure of the entire VC to the Verifier, limiting this way the usability of Verifiable Credentials. To earn the trust of the users and make Credentials widely accepted and secure, the use of BBS+ signatures has been proposed. BBS+ signatures provide the means to selectively disclose only part of a VC. This way, the Holder can provide the Verifier with the information they need, while at the same time keeping their private information hidden.

The use of BBS+ signatures with Verifiable Credentials has drawn a lot of attention lately, with many implementations adopting the BBS+ linked data proof spec, a proposed specification by the W3C Verifiable Credential's working group. In this work we will present two new security vulnerabilities of that specification, along with some proposed solutions. Both Vulnerabilities have been presented and discussed with the W3C working group, while some of the proposed solutions have already started to be implemented. Next, we will propose an alternative, more efficient way, of using BBS+ with Verifiable Credentials, that avoids those problems and allow us to prove the end-to-end security of our scheme, something that has not been done for the existing specification. Our solution uses the more standard JSON representations (compliant with the JSON Encryption (JOSE) family of specs), that allow us to construct a mathematical modeling of a credential, we will eventually use for our security proofs. In addition, we evaluated the efficiency of both

the current “normalization” algorithm (the algorithm responsible with transforming the credential to the appropriate format before it is signed with BBS+, see Section 3.2.1) and the one we propose. As we will see, our algorithm is 5 to $25\times$ times faster than the one currently used.

Περίληψη

Τα επαληθεύσιμα διαπιστευτήρια (Verifiable Credentials) παρέχουν έναν τρόπο σε οντότητες όπως χρήστες, εφαρμογές, συσκευές κ.λπ., να αποδεικνύουν την εγκυρότητα διάφορων ισχυρισμών, όπως το όνομα ενός χρήστη, το επάγγελμα του, τον κατασκευαστή μιας συσκευής, το επίπεδο εξουσιοδότησης ενός υπαλλήλου κ.λπ. Οι ισχυρισμοί αυτοί επαληθεύονται και κατασκευάζονται από τον Εκδότη του διαπιστευτηρίου (π.χ. μια αεροπορική εταιρεία που εκδίδει ένα εισιτήριο). Στη συνέχεια, μετά την κατασκευή του, το διαπιστευτήριο αποστέλλεται από τον Εκδότη στον Κάτοχο, ο οποίος είναι και υπεύθυνος να αποθηκεύει και να ελέγχει το διαπιστευτήριο (π.χ. ο ταξιδιώτης που αγοράζει το εισιτήριο). Τέλος, ο Κάτοχος μπορεί να στείλει το διαπιστευτήριο στον Ελεγκτή, ο οποίος θα εξετάσει την εγκυρότητά του (π.χ. ο υπάλληλος του αεροδρομίου που θα επικυρώσει το εισιτήριο). Εφόσον ο Ελεγκτής αποδεχτεί την εγκυρότητα του διαπιστευτηρίου, το οποίο πετυχαίνεται μέσω κρυπτογραφικών αποδείξεων (που συμπεριλαμβάνονται στο διαπιστευτήριο από τον Εκδότη και τον Κάτοχο του), μπορεί να εμπιστεύεται τους αντίστοιχους ισχυρισμούς, στο βαθμό που εμπιστεύεται τον Εκδότη.

Παρόλο που τα Verifiable Credentials έχουν γίνει αποδεχτά με μεγάλο ενθουσιασμό, εξακολουθούν να υπάρχουν ορισμένα προβλήματα τα οποία απαιτούν περαιτέρω εξερεύνηση. Το μεγαλύτερο από αυτά είναι η αποκάλυψη των ιδιωτικών πληροφοριών του χρήστη. Για πολλές εφαρμογές, τα εν λόγω διαπιστευτήρια είναι πιθανό να περιέχουν πληροφορίες που οι Κάτοχοι τους θα θέλουν να κρατήσουν μυστικές, όπως ο τόπος διαμονής τους, στοιχεία επικοινωνίας, νομικά στοιχεία, το ιατρικό ιστορικό τους κ.λπ. Οι περισσότερες από τις υπάρχουσες εφαρμογές όμως χρησιμοποιούν παραδοσιακές κρυπτογραφικές μεθόδους (ψηφιακές υπογραφές), οι οποίες απαιτούν από τον Κάτοχο να στείλει ολόκληρο το διαπιστευτήριο στον Ελεγκτή. Έτσι και ο Ελεγκτής, αλλά και οποιοσδήποτε επιτιθέμενος που μπορεί να παρακολουθεί την επικοινωνία μεταξύ του Κατόχου και του Ελεγκτή, θα λάβουν πρόσβαση σε όλες τις πληροφορίες που περιέχονται σε αυτό το διαπιστευτήριο, είτε ο Ελεγκτής τις χρειάζεται είτε όχι.

Προκειμένου να αντιμετωπιστεί το παραπάνω πρόβλημα, ώστε να προστατευτεί η ασφάλεια των χρηστών και να κερδηθεί η εμπιστοσύνη τους, έχει προταθεί η χρήση

κρυπτογραφικών υπογραφών BBS+. Οι υπογραφές BBS+ παρέχουν τα μέσα για επιλεκτική αποκάλυψη μόνο ενός μέρους του διαπιστευτηρίου. Με αυτόν τον τρόπο, ο Κάτοχος μπορεί να παρέχει στον Ελεγκτή τις πληροφορίες που χρειάζονται, διατηρώντας ταυτόχρονα κρυφά τα προσωπικά του στοιχεία. Τελευταία, η χρήση των υπογραφών BBS+ έχει τραβήξει μεγάλο μέρος της προσοχής, με πολλές εφαρμογές να υιοθετούν τα αντίστοιχα πρωτόκολλα που ορίζονται από την υπεύθυνη ομάδα της W3C. Σε αυτήν την εργασία θα παρουσιάσουμε δύο νέα κενά ασφάλειας αυτών των πρωτοκόλλων, μαζί με μερικές προτεινόμενες λύσεις. Και τα δύο αυτά προβλήματα παρουσιάστηκαν και συζητήθηκαν με την αρμόδια ομάδα της W3C, ενώ ορισμένες από τις προτεινόμενες λύσεις έχουν ήδη αρχίσει να εφαρμόζονται. Στη συνέχεια, θα προτείνουμε έναν εναλλακτικό, πιο αποτελεσματικό αλγόριθμο, για τη χρήση των BBS+ υπογραφών με σκοπό την ασφάλιση των διαπιστευτηρίων, που αποφεύγει αυτά τα προβλήματα και μας επιτρέπει να αποδείξουμε την ασφάλεια των ανάλογων πρωτοκόλλων. Η πρόταση μας βασίζεται στη χρήση των JSON αρχείων, για τα οποία προτείνουμε μια μοντελοποίηση, η οποία μας επιτρέπει να κατασκευάζουμε αποδείξεις ασφάλειας, ανάγοντας απλά στις ανάλογες ιδιότητες των BBS+. Επιπλέων, όπως θα δούμε, ο αλγόριθμός που προτείνουμε για την κανονικοποίηση ενός Verifiable Credential (διαδικασία που το φέρνει στην κατάλληλη μορφή ώστε να υπογραφτεί με BBS+) είναι από 5 έως και 25 φορές ταχύτερος από αυτόν που χρησιμοποιείται στην πράξη.

Acknowledgements

First of all, I would like to thank my thesis supervisor, prof. Gerogios Polyzos, for their invaluable guidance and patience. I would also like to extend my gratitude to prof. Vasilios Siris and senior researchers Nikos Fotiou as well as the rest of the Mobile Multimedia Laboratory members for their constant input and guidance. Lastly I would like to thank my sister and my good friend and colleague Thomas Tsouparopoulos for their priceless help and guidance as well the rest of my friends and family for their support and patience.

Contents

Abstract vi

Acknowledgements vii

1 Introduction 1
- 1.1 Applications . . . 1
- 1.2 Motivation and Problem Statement . . . 2
- 1.3 Contributions . . . 3
- 1.4 Thesis Structure . . . 3

2 Background and Related Work 5
- 2.1 Verifiable Credentials . . . 5
  - 2.1.1 Ecosystem . . . 5
  - 2.1.2 Data Model . . . 7
  - 2.1.3 Life-cycle Example . . . 8
- 2.2 BBS+ Digital Signatures . . . 10
  - 2.2.1 Mathematical Preliminaries . . . 10
  - 2.2.2 Security Notions . . . 11
  - 2.2.3 Public Data and Key Generation . . . 12
  - 2.2.4 Signing and Verifying . . . 13
  - 2.2.5 Selective Disclosure . . . 13
- 2.3 Related Work . . . 14

3 Linked Data Signatures 17
- 3.1 Linked Data Introduction . . . 17
- 3.2 LD BBS+ Proofs . . . 18
  - 3.2.1 Linked Data Canonicalization . . . 19
  - 3.2.2 Holder Authentication . . . 20
- 3.3 Protocol Exploits . . . 23
  - 3.3.1 URDNA2015 Security Vulnerability . . . 23
  - 3.3.2 Holder Authentication Security Vulnerability . . . 28

4 JSON BBS+ Signatures 33
- 4.1 Modeling JSON . . . 33
- 4.2 Signing JSON Credentials . . . 37

4.3 Experimental Results 38
4.3.1 Set Up 38
4.3.2 Benchmarking Results 39

5 Conclusions **43**

Bibliography **45**

List of Acronyms **48**

List of Figures **49**

List of Tables **50**

List of Algorithms **51**

Appendix A **52**

Appendix B **54**

1 Introduction

Verifiable Credentials (VCs) are a W3C specification [SLC19a] that provides a way for entities (users, applications, devices etc.) to prove claims about a subject. In general, a VC is a file, usually either in JSON or JSON Linked Data (see Section 3), containing various metadata and the claims as key/value pairs, like,

$$\{\text{"}name\text{"} = \text{"}Joe\ Doe\text{"},$$
$$\text{"}age\text{"} = 30\}$$

The easiest way for VCs to be understood is to draw comparisons between them and real life "paper" credentials, like passports and driver licenses. Much like their physical counterparts, VCs are meant to be used to prove statements about the subject of that credential. The main difference however is that Verifiable Credentials are in machine-readable form, allowing for much more convenient and efficient interactions.

1.1 Applications

The first applications for VCs that could come to mind are as a complete replacement of paper credentials like government IDs, licenses, and degrees [Ott+19]. Even in cases where there is a digital equivalent of a paper credential, like airplane tickets, using VCs to represent them ensures interoperability, ease of use and security. One of the main advantages of VCs is the ability to use the same credential for different applications. For example, a government issued VC can be used to open a bank account from home, for remote voting, for easily paying taxes etc. A VC given to a student could be used to get access to the Universities Wi-Fi network, take a test remotely, or prove that they are entitled to a discount given only to students. Similarly, a security badge given to a data-center's employee could be used to prove to a guard that they have the right to enter the area and can also be used to prove that they have the right to log-in to one of the servers.

Verifiable Credentials can also be used to secure existing infrastructures. For example, a lot of security risks in 5G arise from the fact that when a device initially connects to a network, it has no-way of knowing if it is communicating with a legitimate base station or not [PM19]. VCs could be used from the network provider to secure this communication,

by configuring the base station to send a VC to a new connected device. The device then by validating that credential can check the validity of the base-station it communicates with (analogous to how a user will verify the security of a Web Site using a Web certificate).

Other applications include smart home or IoT device management. For example, each device can have their own credential and the resident of the smart home or the operator of the IoT network will have its own. The user can then assign permissions that indicate which devices can communicate with each other, based on those credentials. Moreover, the IoT device manufacturers or the Certifications Testing Lab could assign their own credentials to a device. Then the user or another device could check those credentials, both during installation of the device and periodically, to ensure the security of the network. Lastly, if the user wants to update the software of the device, the device can ensure that only authorized updates will be installed and that only the authorized user will install those updates.

1.2 Motivation and Problem Statement

An important problem concerns the disclosure of the user's private data. For example, if the cryptographic proofs contained in a credential are created with Rivest–Shamir–Adleman (RSA) signatures [RSA78], the entire credential must be presented to the Verifier for the signature to be validated. As a result, anyone could gain access to all the information in that VC, something that can be proven to reduce the usability and flexibility of credentials at best or be quite dangerous at worst. As an example, one can consider VCs that contain personal information like age and address, medical records, or security information like passwords, bank account numbers etc. Consider for example again a VC for a security badge given to an employee of a data-center that contains an ID proving authorization to enter the road gate and an adminID used to allow the employee to log-in to a server. Most likely, the data-center would not want their employees showing their adminID's to the guard on the road gate. From these use cases emerges the need to hide personal or sensitive information from the VC and still be able to convince a relying party regarding the ownership, correctness, and integrity of the revealed information.

Another issue is the correlation of the user through unique information (like IDs) contained in the credential. Generally, we want VCs to be secure against non-trusted authorities, their coalition and even adversaries that through different attacks (like Man-in-the-Middle attacks) can monitor the use of the user's credential. If the user reveals elements in their VC that are unique (like their public cryptographic key or the Issuer's signature), an adversary can correlate all the different uses of that VC, deriving potentially private information and dangerous metadata. For example, if a student has a Verifiable Credential from their University, even if their place of resident is not mentioned to that VC, if they use it to get

some discount from a local cinema, library etc., an adversary, through the credentials ID, can monitor the uses of that VC and narrow down the student's location.

In this thesis we will provide ways to both allow a user to choose what information they want to reveal from their credential and to use their credential without needing unique ID's or other correlatable information.

1.3 Contributions

In the following we give a short description of the contributions we made with this thesis in the area of privacy preserving Verifiable Credentials.

Discovered Exploits The first of our contributions are two discovered protocol exploits of the proposed by the W3C working group specification for using BBS+ with Verifiable Credentials. By taking advantage of the first vulnerability, an adversary could reveal information the Holder wants to keep secret (something that we managed to reproduce), while the second exploit gives the ability to the adversary to forge a credential with any claims they want. We reported both vulnerabilities to the working group, and we discussed possible solutions, which we will also present here.

Alternative pipeline We designed and implemented an alternative solution based on the JSON representation of Verifiable Credentials using BBS+ cryptographic signatures. The simplicity of our scheme allows for well-defined security properties, which we also prove mathematically (something not done for the current specifications), while also improving performance.

Evaluation and Benchmarking Finally, through a series of benchmarks, we evaluated the performance of our construction compared to the algorithms proposed by the W3C specification for BBS+ signatures with Verifiable Credentials (which are also used in practice).

1.4 Thesis Structure

Chapter 2

In the second chapter we will give the necessary background on the Verifiable Credentials ecosystem and data model, as well as for the BBS+ signatures. We will also give an overview of the related work in the area of privacy preserving identity management on the Web.

Chapter 3

In the third chapter we will look at the linked data Verifiable Credentials and signatures proposal. We will also present the two vulnerabilities we discovered in the corresponding proposed specification.

Chapter 4

In this chapter we will present and evaluate our proposal for privacy preserving Verifiable Credentials, based on the JSON representation.

Chapter 5

In the final chapter we conclude our work and propose some future work in the area.

2 Background and Related Work

2.1 Verifiable Credentials

2.1.1 Ecosystem

An overview of the Verifiable Credentials Ecosystem [SLC19b] is shown in Figure 2.1. The four main entities are the Issuer, the Holder, the Subject, and the Verifier. In the following, these entities and their roles are discussed in more detail,

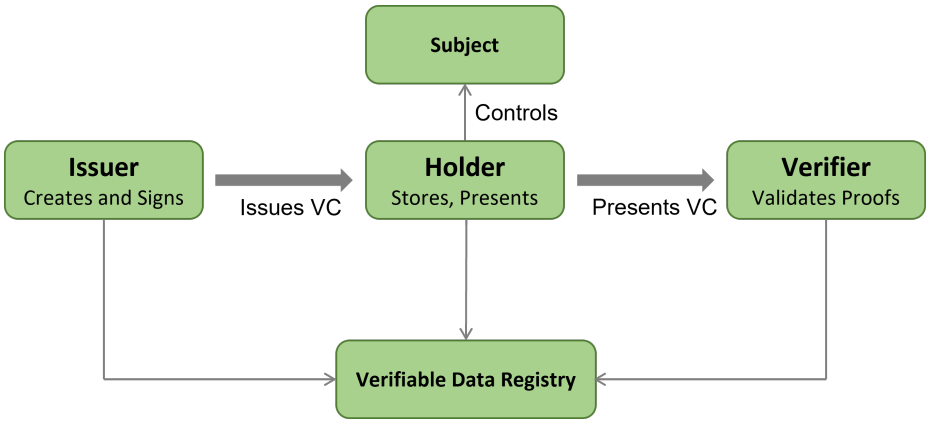

Fig. 2.1: The Verifiable Credential's Ecosystem. Figure taken from: https://www.w3.org/TR/vc-data-model/

Subject The Subject is the entity for which claims in a Verifiable Credential are made. A VC can contain claims about one or more subjects. A subject could be a Human, a thing, like an application, a resource etc.

Issuer The Issuer is the entity that will originally create the credential by combining claims about one or more Subjects. The Issuer will also act as the guarantor of the integrity and truthfulness of these claims. As a result, they will be commonly tasked with authenticating the Holder/Subject and creating the necessary (usually cryptographic) proofs (see Section 2.1.2). Examples of entities that could act as an Issuer are governments creating citizen ID's and passports, universities issuing student ID's, or companies issuing employee security badges, all in the form of a Verifiable Credential. Examples of proofs are RSA digital signatures, or message authentication codes (MACs) combined with the Issuer's cryptographic key [KBC97].

Holder The Holder is the designated (by the Issuer) controller of the credential. They are responsible for safely storing and presenting the VC. When presenting a credential, the role of the Holder is to prove integrity of the claims and the protected metadata, prove that they are the rightful controller of the VC and prove that the designated Issuer created the credential. To do that the Holder, like the Issuer, must generate their own cryptographic proofs. Note that in many cases the Holder and the Subject will be the same entity (e.g., a undergraduate in possession of a student's ID, is both the Subject and the Holder of that credential). However, this is not always the case. For example, the Holder could be a parent in control of a credential for one of their children, or a manufacturer in control of a VC for one of their devices. In those cases, the proofs created by the Holder, must also prove that they have the right to represent the Subject.

Verifier The last recipient of the Verifiable Credential is the Verifier. The Verifier will receive the credential from the Holder along with all the necessary cryptographic material and proofs. These proofs will typically include the ones created by the Holder, but it is not necessary to also include the ones created by the Issuer (like in the case of BBS+ signatures, see Section 2.2). In any case, the Verifier must be able to validate the integrity of the presented claims and metadata, to confirm who is the Issuer of the credential and to authenticate the Subject and the Holder. Example of Verifiers include airport personal receiving tickets as VCs, bookstores receiving student ID's etc.

Aside from the aforementioned entities, many implementations, as well as the W3C specification, also include a verifiable data registry as part of the ecosystem. That entity is usually responsible with maintaining identifiers (e.g., Issuer/Holder ID's, decentralized identifiers etc.), credential schemas, revocation registries, metadata like expiration dates etc. Example data registries include decentralized ledgers, trusted databases (like private enterprise databases), government ID databases etc. The reason that we do not include the verifiable data registry as a main part of the ecosystem is that their use and utilization varies among applications, with many use-cases using multiple types (like government databases to store official ID's and decentralized ledgers to store revocation lists) while others may not use them at all (like verifiable credentials using BBS+ signatures).

From the above we can conclude that there are only two trust relationships necessary for the ecosystem to function; The Issuer must trust the Holder and the Verifier must trust the Issuer. If the ecosystem also includes a verifiable data registry, all other entities (Issuer, Holder, Verifier) should also trust that registry. The above trust model is one of the big advantages of Verifiable Credentials. In contrast with other "certificate authority" trust models [Mye+99] or systems like Open ID Connect [Sak+14], there is no central third-party authority that establishes trust between the entities of the ecosystem. That is

not to say that these authorities will not be useful in some cases where is not possible to establish one of the necessary trust relationships otherwise. That being said, in the VCs ecosystem, these central authorities are optional and not a main part of the specification, while in many use cases may even be obsolete. For example, most likely any Verifier will trust a passport issued by a government, a client (Verifier) will trust a receipt issued by a manufacturer (Issuer), or a company will trust the security badge of an employee (The company here is both the Issuer and the Verifier).

2.1.2 Data Model

At a high level, the verifiable credential's data model, seen in Figure 2.2, consists of three main parts, the claims, the metadata, and the proofs. In the following we discuss each of them.

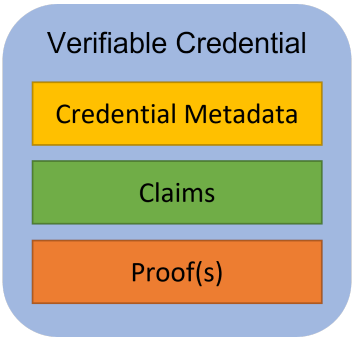

Fig. 2.2: The Verifiable Credential's Data Model. Figure adapted from: https://www.w3.org/TR/vc-data-model/

Claims Claims are usually represented as key/value pairs like <"first name": Joe> and <"age": 20>. More generally, claims represent statements about a subject and can be modeled as a subject-property-value relationship. To model a set of more than one claims, a popular and widely used approach (proposed by the W3C VC spec), is to merge them together in a common "knowledge graph". This approach seems to be especially useful -or even necessary- when using linked data to represent VCs (Section 3.1). As an example, the knowledge graph of Figure 2.3, corresponds to the claims that Pat (the subject) is alumni (the property) of AUEB (the value), and also knows Sam who is a professor (notice how each edge and node in the knowledge graph has a label).

Metadata Metadata are highly application dependent; however, some common occurrences include the credential's Issuer identifier, the issuance and expiration dates, cryptographic material etc. Those metadata may also be signed by the Issuer. An important part of the metadata is the context of the credential. The context is used to map human readable keys and values to longer Resource Identifiers or objects, containing machine readable information about those keys and values. That infor-

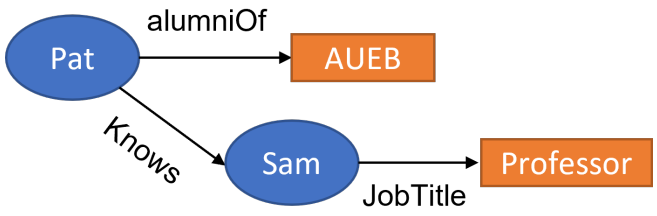

Fig. 2.3: A set of claims as a knowledge graph. Figure adapted from: https://www.w3.org/TR/vc-data-model/

mation could for example, be a URL mapping to a schema.org page (e.g., "name": "https://schema.org/givenName"), or an object specifying the *version*, *value*, *type* etc., of a term. Contexts have many uses apart from making long URLs human readable. They allow linking complex structured information with a single term, giving the ability to easily create arbitrary large knowledge graphs. They also give the ability to update the definition of a term in a credential, without needing to update every credential with that term. That last advantage, however, could also be the cause of various security risks.

Proofs The purpose of the proofs in a VC may vary, but at least they always should provide proof of the integrity of the claims (and the protected metadata, if those exists). They must also provide insurance about the creator of the proof. As an example, a proof could be created by the Issuer of a credential using RSA Digital Signatures. Then anyone could use the Issuer's public key to verify the integrity of the claims. An important distinction to make here is between proofs provided by the Issuer and proofs provided by the Holder. In addition to the base requirements, the Issuer's proof should identify them as the creator of the credential. On the other hand, proofs provided by the Holder should designate them as the rightful controller of the VC.

2.1.3 Life-cycle Example

The following is a simple example of using a verifiable credential as an airplane ticket. The necessary proofs are created using RSA digital signatures. The credential itself is encoded using JSON. Figure 2.4, shows the verifiable credential originally created by the airline company (Issuer), containing all the necessary claims and information regarding the traveler (Holder). The airline also wants to authenticate the Holder. For that purpose, they require from the traveler to create a public/private RSA key pair. After the airline creates the JSON file, they sign it using RSA digital signatures using the company's RSA private key, creating the JSON file of Figure 2.5. That is also the file that the company will send to the traveler. Note that one of the signed claims is the public RSA key of the

```
{ // Metadata
 "@context": "https://www.w3.org/2018/credentials/v1",
 "Issuer": "https://www.example.airline.com"
 "issuanceDate": "2010-01-01T19:73:24Z"
 // Claims
 "credentialSubject": {
                "RSAkey": "dasd343fsd54523bdt7234"
                "name": "John Doe",
                "email": "JahnDoe@bestMail.com",
                "address": "City Zip",
                "escorts": [{"name": "Kale", "age": "9"},
                            {"name": "Kaiya", "age": "12"}],
                "Ticket": {"Gate": "D 12",
                           "Flight": "A2321",
                           "From": "New York",
                           "To": "Hong Kong"}}
}
```

Fig. 2.4: An airline ticket represented as a Verifiable Credential.

traveler. In the airport, when the traveler is required to present their ticket, they will also use their RSA private key to sign the credential, creating the JSON file of Figure 2.6.

```
{  // Metadata
 "@context": "https://www.w3.org/2018/credentials/v1",
 "Issuer": "https://www.example.airline.com"
 "issuanceDate": "2010-01-01T19:73:24Z"
  // Claims
  "credentialSubject": {
                "RSAkey": "dasd343fsd54523bdt7234"
                "name": "John Doe",
                "email": "JahnDoe@bestMail.com",
                "address": "City Zip",
                "escorts": [{"name": "Kale", "age": "9"},
                            {"name": "Kaiya", "age": "12"}],
                "Ticket": {"Gate": "D 12",
                           "Flight": "A2321",
                           "From": "New York",
                           "To": "Hong Kong"}}
 // The Issuer's Proof
 "proof":  { "type": "RSAsignature"
             "signature": "asda837478us987987qwey....."
              }
}
```

Fig. 2.5: The VC from Figure 2.4 signed by the Issuer/ Airline (green part).

After the airport staff (Verifier) receive the credential, they will have to first validate the Issuer's proof and next the Holder's. To verify the Issuer's proof, the airport will contact their private database containing all the airline companies registered to fly though the airport and their cryptographic public keys. Using the Issuer field from the credential, the airport will retrieve the airline's RSA public key and verify the Issuer's signature. Verifying that signature proves that the specific airline company truly created that ticket, and that the claims it contains (travel information and Holder's public RSA key) are correct. Next the airport will use the Holder's public RSA key contained in the credential, to verify the Holder's signature. If that signature is legitimate, it means that the traveler is in possession

of the secret key corresponding to the public RSA key, inserted in the credential by the Issuer. As a result, the traveler must be the rightful owner of that ticket. In the end, the airport personnel can be sure that the ticket is authentic and that the traveler is its rightful owner.

```
{ "Ticket":
   {"@context": "https://www.w3.org/2018/credentials/v1",
    "Issuer": "https://www.example.airline.com"
    "issuanceDate": "2010-01-01T19:73:24Z"
    "credentialSubject": {
                "RSAkey": "dasd343fsd54523bdt7234"
                "name": "John Doe",
                "email": "JahnDoe@bestMail.com",
                "address": "City Zip",
                "escorts": [{"name": "Kale", "age": "9"},
                            {"name": "Kaiya", "age": "12"}],
                "Ticket": {"Gate": "D 12",
                           "Flight": "A2321",
                           "From": "New York",
                           "To": "Hong Kong"}}
   // The Issuer's Proof
   "proof": { "type": "RSAsignature"
              "signature": "asda837478us987987qwey....."
              }
        }
// The Holder's Proof
 "proof": { "type": "RSAsignature"
            "signature": "ajf828340fsdf9kjdfjsfu654....."
   }}
```

Fig. 2.6: The VC from Figure 2.5 also signed by the Holder (green part).

2.2 BBS+ Digital Signatures

BBS+ signatures were originally proposed as a group signatures protocol by Boneh, Boyen, and Shachum [BBS04] (from where they take their name) and have been refined by Au Man Ho et.al. [Ho+12] and later by Jan Camenisch, Manu Drijvers and Anja Lehmann [CDL16] (where the + comes from). BBS+ signatures have many advantages, having keys and signature sizes of 4-12 times shorter than RSA and only 2 times larger than ECDSA, while at the same time being as efficient as ECDSA (one of the more efficient digital signature protocols [DK18]). Furthermore, in contrast with many cryptosystems, like RSA, that work by encrypting a single message (byte array), BBS+ works on a list of messages, giving the ability to create proofs of knowledge, use selective disclosure and keep a zero-knowledge property all at the same time (we will discuss those terms in more detail below).

2.2.1 Mathematical Preliminaries

The basic mathematical constructions used in BBS+ are finite algebraic groups and pairings between those groups. A finite group is a finite set of elements, on which an operation is

well defined (usually either addition or multiplication) and has certain properties. More specifically, the operation in a finite group must be associative, have an identity element (in the group) and for each element of the group its inverse must also be an element of the group. We will denote a finite group as $(\mathbb{G}, f)$, where f the operation of the group $\mathbb{G}$. Examples include the additive group of integers $modulo\ n$; $(\mathbb{Z}_n.+)$, and the permutation group of length n; $(S_n, \circ)$ with group operation the composition of such permutations.

In a finite group $(\mathbb{G}, f)$, an element or a set of elements are said to "generate" the group $\mathbb{G}$, if every element of that group can be expressed as a combination (using the group's operation f) of the generators or their inverses. In this work we will mainly consider groups of prime cardinality (order) p. Those groups have only one generator and every element can be considered as one.

For finite groups $\mathbb{G}_1, \mathbb{G}_2$ and $\mathbb{G}_T$, a pairing is a function, $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_T$ so that two properties hold,

$$\text{Bi-linearity} : \forall x \in G_1,\ y \in G_2\ and\ a,\ b \in Z_p,\ \ e(x^a, y^b) = e(x, y)^{ab} \tag{2.1}$$

$$\text{Non-degeneracy} : e \neq 1 \tag{2.2}$$

Pairings, initially used as an "attack" against a large variety of cryptosystem, have been proven to have many interesting functions and uses in cryptography. See Appendix A for more information on pairings in cryptography.

BBS+ are also based on the q-strong Diffie-Helman assumption for their security properties. That assumption states that given two finite groups $\mathbb{G}_1$ and $\mathbb{G}_2$ of prime cardinality p, and elements $g_0 \in \mathbb{G}_1$ and $h_0 \in \mathbb{G}_2$, it is hard to find an algorithm that on input $g_0, h_0, h_0^x, h_0^{x^2}, ..., h_0^{x^q}$ produces an output (A, x) such that $A^{\frac{1}{x+c}} = g_0$ for some $c \in \mathbb{Z}_p$. BBS+ signatures base their security on the truth of that assumption.

2.2.2 Security Notions

As already mentioned, aside from efficiency, the biggest advantage of BBS+ signatures is that they provide three very strong privacy preserving security properties; zero knowledge, proof of knowledge and selective disclosure.

Zero knowledge refers to the inability of an adversary to gain any information regarding a plain text or a secret cryptographic key from the corresponding cipher text. In other words, an adversary trying to extract the plain text or the secret key from a cipher text, cannot find a better method than brute force. Commonly, a protocol is said to be zero knowledge if there is a way to "simulate" the cipher texts, meaning that without

knowledge of the secret key or the plain text, there is a way to produce cipher texts of the same probability distribution as the ones produced when a legitimate plain text and secret key are used. Under those conditions, an adversary observing the cipher texts will gain the same amount of knowledge regarding the plain-text and secret key from both the "true" cipher-text and the simulated one. In the second case though, there is no knowledge of those elements to be gained and as a result, neither in the first.

Proof of Knowledge means that through the completion of the protocol, the Verifier can be convinced that the Prover is in possession of a secret. The challenge is of-course to keep the zero-knowledge property, meaning that the Verifier will still gain no information about that secret. To do that, the protocol must be shown to have an "extractor", meaning a hypothetical machine with special capabilities (like rewinding the Prover to a previous state), that can interact with the Prover. If the Prover manages to complete the protocol successfully, the extractor must be able to derive the Prover's secret information (plain-text or keys). The protocol is then said to provide "proof of knowledge", in that on completion, the Verifier can be sure that the Prover knows the secret, since if they didn't, the extractor would not succeed and as a result, the protocol should also have failed.

Selective disclosure is the ability to only reveal parts of a secret to a Verifier, and still be able to prove integrity and possession of the revealed information, as well that the revealed information is part of the original larger secret. In the case of VCs, there are two types of information that the Holder would not want to disclose. The first is values from the credential itself, like their age or address, to protect their private data. The second is the signature of the Issuer, the reason being that through that signature, since it will most likely be unique, the user could be correlated and tracked. Additionally, revealing the Issuer's signature, gives an adversary an advantage when trying to brute-force the hidden claims of the credential, breaking that way the zero-knowledge property.

2.2.3 Public Data and Key Generation

The public data of the BBS+ public key infrastructure (PKI) contain the finite groups $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ of prime order p, a common pairing $e : \mathbb{G}_1 \times \mathbb{G}_2 -> \mathbb{G}_T$ and elements, $g_0, g_1, ... \in \mathbb{G}_1$ and $h_0, h_1, ... \in \mathbb{G}_2$. The number of those elements is arbitrary and depends on the Issuer. Also $h_0, h_1, h_2, ...$ can either be considered as part of the public key or as public data. Furthermore, we will reserve the elements g_1 and h_0 as the "base" generators of $\mathbb{G}_1$, $\mathbb{G}_2$, or more generally, we will consider $\mathbb{G}_1$ as the group generated from

g_1 and $\mathbb{G}_2$ as the group generated from h_0. To create the keys, the Issuer chooses a random $x \in_R Z_p$. The Issuer's secret key (SK) and public key (PK) are,

$$SK_{Issuer} = x \tag{2.3}$$

$$PK_{Issuer} = h_0^x \tag{2.4}$$

The choice of $\mathbb{G}_1$, $\mathbb{G}_2$ and $\mathbb{G}_T$ is very important as it affects both security and performance. The BBS+ PKI uses groups or subgroups of elliptic curves and more specifically of the BLS12-381 elliptic curve family [Bow17]. For more information see Appendix B.

2.2.4 Signing and Verifying

As already mentioned, BBS+ signatures create signatures for a list of messages (bit arrays). To sign a block of messages, $m_1, m_2, ..., m_L$ the Issuer chooses random $\epsilon, s \in_R Z_p$ and then computes the messages "credential" A as, e

$$A = [g_1 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_L^{m_L}]^{\frac{1}{\epsilon+x}} \tag{2.5}$$

The signature then on $m_1, m_2, ..., m_L$ is $\sigma = (A, \epsilon, s)$. To verify the signature a Verifier only needs to check the equality,

$$e(A, PK_{Issuer} \cdot h_0^{\epsilon}) = e(g_1 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_L^{m_L}, h_0) \tag{2.6}$$

where e the public pairing. That equality is a direct result of the bi-linearity property of pairings (meaning that $e(x^a, y^b) = e(x, y)^{ab}$) since,

$$\begin{aligned} e(A, PK_{Issuer} \cdot h_0^{\epsilon}) &= e([g_1 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_L^{m_L}]^{\frac{1}{\epsilon+x}}, h_0^x h_0^{\epsilon}) = \\ &= e([g_1 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_L^{m_L}], h_0)^{\frac{1}{\epsilon+x}(\epsilon+x)} = \\ &= e(g_1 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_L^{m_L}, h_0) \end{aligned} \tag{2.7}$$

The signature can be proven to be secure under adaptive chosen message attacks.

2.2.5 Selective Disclosure

Let the Holder be in possession of the messages $m_1, m_2, ..., m_L$ and a BBS+ signature on those messages $\sigma = (A, \epsilon, s)$, created by the Issuer. For the Holder to prove possession of the signature σ in zero knowledge, while only disclosing a subset of the messages $\{m_i\}_{i \in D}$, where D the indexes of the disclosed messages, there are two high level steps. First, randomize the values of the signature σ to "blind" them (we don't want to reveal σ). Second, follow a zero knowledge proof of knowledge protocol to prove some relations

between the blinded and un-blinded signature as well as between the disclosed and un-disclosed messages (while of course only revealing the blinded signature and the disclosed messages).

More formally, the Holder chooses random values $r_1, r_2 \in_R \mathbb{Z}_p$ and sets $r_3 = \frac{1}{r_1}$ and $s' = s - r_2 r_3$. Then they calculate $A' = A^{r_1}$, $\bar{A} = A'^{-\epsilon} b^{r_1}$ and $d = b^{r_1} h_0^{-r_2}$ where, $b = g_1 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_L^{m_L}$. The elements $(A', \bar{A}, d)$ are the blinded elements of the BBS+ signature σ. Let π be a zero knowledge proof (using any suitable zero knowledge protocol) that the following hold,

$$\begin{aligned} \bar{A}/d &= A'^{-\epsilon} h_0^{r_2} \\ g_1 \prod_{i \in D} h_i^{m_i} &= d^{r_3} h_0^{-s'} \prod_{i \notin D} h_i^{-m_i} \end{aligned} \tag{2.8}$$

The derived proof of knowledge that the Holder will send to the Verifier is $(A', \bar{A}, d, \pi)$. Note that the left parts of the above equations can be computed by the Verifier but not the right ones. A rough intuition about the equations 2.8 is that they have two purposes. First, to allow the Holder to prove that they posses an element A that has the "correct" form, meaning that $A = b^{\frac{1}{\epsilon + x}}$ and $b = g_1 h_0^s \prod_{i \in \{1,2,\ldots,L\}} h_i^{m_i}$, without revealing any information about A or b. Secondly, that $D \subseteq \{1, 2, \ldots, L\}$ meaning that the disclosed messages are part of the originally signed (by the Issuer) messages. Both of these requirements can be met by proving the two equations below,

$$\begin{aligned} A' = A^{r_1} &= b^{\frac{r_1}{\epsilon + x}} \\ b = g_1 \cdot h_0^s \cdot \prod_{i \in D} h_i^{m_i} &\cdot \prod_{i \in \{1,2,\ldots,L\} \setminus D} h_i^{m_i} \end{aligned} \tag{2.9}$$

By using these equations for A' and b, and moving all the elements that the Holder will reveal at the left side while keeping all the secret elements to the right, we can see that the equations (2.9) are equivalent to the equations of (2.8).

The Verifier now must verify π and that the equality, $e(A', PK_{Issuer}) = e(\bar{A}, h_0)$ holds. As already mentioned, proving the two equations 2.8 (meaning verifying π) proves that $(A', \bar{A}, d)$ have been created using elements of a BBS+ signature, while checking that $e(A', PK_{Issuer}) = e(\bar{A}, g2)$, binds the signature to the Issuer's public key.

2.3 Related Work

Perhaps the most notable example of identity management on the Web are single sign on systems like Open ID Connect (OIDC) [Sak+14], which is still used by more than a million web sites today [Sim21]. OIDC enables the use of a single identity, provided by an identity authority, improving this way security (for example by allowing 2 factor authentication),

efficiency, by neutralizing the need to manage different passwords etc. That being said, OIDC has a lot of drawbacks. Each of the user's logins to a reliable party requires the communication between that party and the identity provider. These interactions allow the tracking of the user by the identity provider, leading to concerns about their privacy [Bra07]. Furthermore, the identity provider must always be online for the process to be successful. As a result, only "tech giants" like Google or Amazon can realistically act as identity providers, while it is harder for other organizations like smaller companies or universities to be trusted with this responsibility.

The answer to the above limitations are anonymous credentials ([NQ21], [PZ13], [Zhi+21], [CL04]), that decouple the "identity issuance" procedure from the verification process. These kinds of credentials usually use an Issuer-Holder-Verifier ecosystem, like the one proposed by the W3C specification and with similar functionalities (Section 2.1.1). There are various proposals and implementations of anonymous credentials which present different trade-offs. For example, some of the solutions like Microsoft's Azure Active Directory Credentials and Decentralized Identity with ION [NQ21] use popular standards (like RSA) but do not support selective disclosure or tracking protection. Others may support selective disclosure but will also introduce some other trade-off. For example, Microsoft's U-Proove [PZ13] offers the ability to revoke a credential at any time, but to do so, it inserts to the credential correlatable elements. On the other hand, El-Passo [Zhi+21] offers two factor authentication and protection from the tracking of the user's activity on different reliant parties but not from the same one (i.e., if a Holder sends the same credential multiple times to the same Verifier, those uses could be correlated). Additionally, many of those proposals do not strictly apply any specification and for that reason end-up (in various degrees) to be application specific. As a result, they are suffering from poor interoperability and limited functionalities, which is counter intuitive to the use of selective disclosure.

Furthermore, most of the anonymous credential proposals supporting selective disclosure ([BBC08], [PZ13], [Zhi+21]), consider the credentials as already a set of attributes that can be signed with a supporting cryptographic scheme (like BBS+). However, in practice those credentials will be in some structured format like JSON or XML. Turning these formats to a list of attributes is not trivial (see Section 3.3.1). As a result, many implementations and standards will use over-complicated or insecure algorithms (like the URDNA2015 algorithm [AL20], see Section 3.2.1), or they will make over-simplifications, for example by only considering the most "outer" attributes in the credential and not give the ability to use selective disclosure on nested values [MW21].

In this work, we will look at the most recent and complete proposal of the W3C BBS+ linked data proofs spec [LS21], for credentials that follow the W3C specification and allow selective disclosure and user correlation protection. We will examine their proposed algorithm for extracting a list of attributes from a credential and authenticating the Holder. Then we will analyze some security vulnerabilities and exploits of those algorithms. Finally,

we will propose our own algorithm for extracting the list of attributes and show that it offers enhanced security and performance.

3 Linked Data Signatures

Currently, the state-of-the-art in using and signing Verifiable Credentials, is to use the JSON Linked Data (JSON LD) format [Spo+20]. The use of JSON LD is recommended by the W3C specification, which itself is highly focused on the linked data functionalities instead of the simple JSON format (e.g the use of contexts, id links etc). Furthermore, many Verifiable Credential's implementations[1] have opted for the use of JSON LD formats. Similarly, the only W3C standardization body and specification for the use of BBS+ with Verifiable Credentials until now is targeted exclusively to linked data signatures [LS21]. That being said, a lot of criticism against the standard has stemmed from worries regarding the security of the linked data approach [Har20]. Up until now however, these worries were mainly based on the apparent complexity of some of the linked data processing algorithms, and not on some known security vulnerability. In this chapter, we will discuss the advantages that made linked data the format of choice and some of the algorithms used in practice. Finally, we will present two discovered security vulnerabilities of the linked data BBS+ proposed specification.

3.1 Linked Data Introduction

Linked data is a standards-based way to use structured interlinked machine-readable data on the Web [SAM15]. The basic idea is that every resource, piece of data or "thing" will be bound to a Uniform Resource Identifier (URI). Other pieces of structured data can use these URIs to mention (or link to) other resources, creating a large graph of linked data. The whole process is analogous to the use of URLs in the Web. Instead however, of users "clicking" on URLs to fetch Web Sites that contain links to other Web Sites etc., machines and applications can use resources that link to machine readable structured data (which then themselves may contain links to other resources etc.).

Linked data are in the core of the Semantic Web vision, first proposed by W3C [BHL01]. In the Semantic Web, the entirety of the World Wide Web and the Internet will act as a big -machine readable- database. The first big advantage of linked data and the Semantic Web is that machines will be able to "semantically query" a linked data graph (like the

[1] Examples of companies adopting the Linked Data approach for VCs;
Mattr: https://mattr.global/
DigitalBazzaar: https://www.digitalbazaar.com/
Transmute: https://www.transmute.industries/

Semantic Web), to get and semantically analyze the information they need. Another big advantage of linked data is their updatability. In most cases, the URI associated with a resource will remain constant even if something in this resource changes or is updated. As a result, other resources "connected" to the updated resource through that URI, will not need to also be updated.

To use linked data with VCs, the dominant approach is to use JSON LD (JSON linked data) to represent the credential. JSON LD are essentially JSON files, meaning they follow the same rules and syntax, but with some additional functionalities. In a JSON LD file, the value corresponding to a key starting with "@" will be considered as a link to some other resource (e.g. {"@WebSite": "https://www.myWebSite.com"}). Additionally, keys themselves can be links to other resources containing (mostly) semantic information about that key/value pair. For example {"https://schema.org/familyName": "Joe"} will associate the name "Joe" with the semantic information available at "https://schema.org/familyName", meaning that a machine parsing that credential will "know" that "Joe" is a text value, is associated with a "Person" item etc.

3.2 LD BBS+ Proofs

As we mentioned in Section 2.2, BBS+ signatures work on lists of bit arrays (messages). However, it is not trivial to convert a JSON file, and especially a JSON LD file, in a list of messages. Since our ultimate goal is to give the Holder at each time the ability to choose the parts of their credential they want to reveal, each message (or set of messages) should correspond to a claim in the credential. Given that, JSON LD credentials are modeled using knowledge graphs (e.g., Figure 2.3 in Section 2.1.2), the specification uses the edges of those graphs as the messages signed with BBS+. Then the Holder can reveal any part of the knowledge graph they want, with different parts of the graph corresponding to different claims.

```
{"@context": "https://www.w3.org/2018/credentials/v1",
  "Issuer": "https://www.example.com/Issuers/9876"
  "Subject": {
      "@id": "did:example:abc12345"
      "name": "Jane",
      "residence": {
            "@owner": "did:example:abc12345"
            "address": "213"},
}
```

Fig. 3.1: An example of a Verifiable Credential in JSON LD format.

A problem that rises is that not all nodes in a linked data graph have a label with which to represent them when creating the list of all the edges. To see how those un-named, or

-blank- nodes, may appear in a knowledge graph, let's consider the following example. The JSON LD credential of Figure 3.1 has the corresponding knowledge graph of Figure 3.2. It is easy to check that both the JSON LD and the graph representation convey exactly

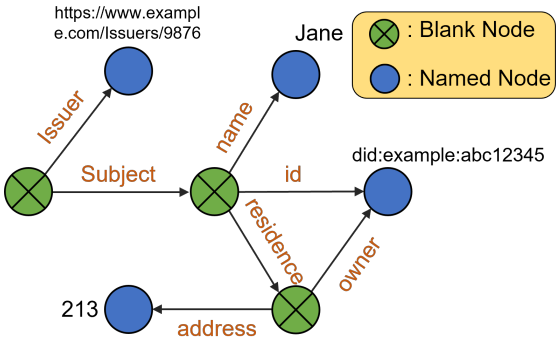

Fig. 3.2: The corresponding Knowledge Graph of the credential from Figure 3.1.

the same information. Note that in the JSON LD credential, each value, either a nested object or a literal, are mapped to a node in the knowledge graph. If the value is a literal (e.g., "Jane", "213" etc.), then the node will get that value as a name. If the value is a nested object however, in most cases, there is not a way to immediately name the corresponding node and it will be represented as a blank node in the knowledge graph.

3.2.1 Linked Data Canonicalization

To label the blank nodes of the knowledge graph of a JSON LD credential, the best known algorithm is the Unified RDF Canonicalization algorithm (URDNA2015) [AL20]. The URDNA2015 algorithm was first proposed as a way to check if two knowledge graphs were isomorphic (meaning they would convey the same information), a problem also hard if the two graphs contained un-labeled nodes. The algorithm produces "canonical" labels for each blank node, meaning labels that will remain the same between two isomorphic graphs. Then, if the two graphs were truly isomorphic, their lists of edges should be identical. The same functionality is clearly useful in the case of JSON LD BBS+ signatures. For those reasons, along with its surprisingly good performance and the fact that the algorithm was already used by many (for other applications), the URDNA2015 algorithm became ideal to be used with BBS+. In the following, we will give an short overview of the URDNA2015 algorithm.

The basic idea of the algorithm is to create labels for each blank node using the information that is linked with that node in the knowledge graph. To do that, the algorithm will call its "First-Degree-Hash" sub-algorithm that checks the information that is directly linked to the node (meaning its neighbors in the graph). If there are blank nodes in the graph that cannot be characterized from only their neighbors, the algorithm will call the "N-degree-Hash" sub-algorithm to label them. Bellow follows a discussion on both.

First-Degree-Hash

The First-Degree-Hash algorithm is called for every blank node in the knowledge graph. For each node that the algorithm visits, it gives to that node the temporal label "_:a" and to every other blank node the temporal label "_:z". Then the algorithm will create a set of all the neighbors (both in and out, the knowledge graphs are directed) of that node, called reference set. To represent the edges of the graph a special format is used called "n-quad" or just "quad", that also includes the label (or type) of the edge (predicate) and is of the form; {$Node_1$, *Edge Label*, $Node_2$}. In practice, the graph itself will be represented as a collection of quads called Resource Description Framework (RDF) [CWL14]. Quads will most likely also contain additional information aside the *Object* (i.e., $Node_1$), *Predicate* (i.e., *Edge Label*) and *Subject* (i.e., $Node_2$), like value types, graph names etc., but that simplification will suffice here.

After the First-Degree-Hash algorithm constructs the reference sets of each blank node, it will calculate the cryptographic hash of the concatenated elements of each reference set separately. That hash, called the "first-degree-hash" of that blank node, essentially "encodes" in one number the information directly linked to each blank node. Finally, it will assign labels to the blank nodes with a unique first-degree-hash, in the same succession as the alphabetical order of the base64 representation of those hashes. All the blank node labels will have the prefix "_:c14n", added to an integer indicating the order of that label. For example, the blank node with the smallest (alphabetically) first-degree-hash (the hash of its reference set) will get the label "_:c14n0", the one with the second smallest (unique) first-degree-hash will get the label "_:c14n1" etc.

N-Degree-Hash

The N-Degree-Hash sub-algorithm is called when there are blank nodes in the graph that have the same first-degree-hash. The algorithm will start traversing the graph, searching for information that will distinguish those blank nodes from each other. As a result, the "N-Degree-Hash" algorithm is the most demanding when it comes to performance.

3.2.2 Holder Authentication

We have mentioned in Section 2.1.2, that the proof created by the Issuer must identify them as the author of the credential, prove the integrity of the claims and provide a way for the Holder to prove ownership of that VC. BBS+, like any other digital signature, can give an answer to the first two requirements. However, up until now we haven't tackled the last one. Since anyone in possession of a VC and a BBS+ signature on that VC could create a Holder's zero knowledge proof of knowledge to send to any Verifier, the question that rises is "How do we bind a Verifiable Credential to only one designated Holder?".

Commonly, in cryptographic applications binding a "thing" to a "person" means binding that thing to the person's public/private key pair. This will also be our goal in order to answer the above question. A first solution, applied with other cryptographic schemes like RSA, is to include the public key of the Holder as part of the credential that will later be signed by the Issuer (See proofs paragraph in the 2.1.2 Section). An important drawback of the above method is that a public key is most likely unique, meaning it could be used to correlate the Holder and the use of their credential. Using BBS+ signatures we can avoid including correlatable components to authenticate the Holder, by taking advantage of the commitments mechanic.

Commitments

A commitment is essentially a Holder's supplied message to be signed by the Issuer, without them (the Issuer) knowing what that message is. There are many commitment cryptographic protocols. The one used in the case of BBS+ is a simplification of the Pedersen's commitment scheme that is also a zero-knowledge protocol [Ped91], meaning that the Issuer will not be able to get any information about the Holder's committed message. Let $m_1, m_2, ..., m_n$ be the messages that the Issuer would sign normally and m_c be the message that the Holder wants to commit. To create the committed message, they will calculate the commitment C as,

$$C = h_{n+1}^{m_c} \tag{3.1}$$

With no commitment from the Holder, the BBS+ signature would be (A, e, s) with $A = b^{\frac{1}{e+x}}$ and $b = g_0 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_n^{m_n}$ (Section 2.2.5). To include the Holder's commitment C, the Issuer instead of $A = b^{\frac{1}{e+x}}$ will calculate $A_c = (C \cdot b)^{\frac{1}{e+x}}$. From the equation 3.1 and the formula of A above we get the following,

$$\begin{aligned} A_c = (C \cdot b)^{\frac{1}{e+x}} = (h_{n+1}^{m_c} g_0 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_n^{m_n})^{\frac{1}{e+x}} => \\ => A_c = (g_0 h_0^s h_1^{m_1} h_2^{m_2} \cdots h_n^{m_n} h_{n+1}^{m_c})^{\frac{1}{e+x}} \end{aligned} \tag{3.2}$$

It is straightforward now to check that $\sigma_c = (A_c, e, s)$ is a valid BBS+ signature on the messages $m_1, m_2, ..., m_n$ and m_c (note that A_c is of the exact same form as A in equation 2.5).

Another key aspect of the BBS+ signatures is that when the Holder wants to generate their proof for the Verifier (ZKP POK) they must know all the messages that were originally signed by the Issuer. Observe that the equations 2.8 in Section 2.2.5, that the Holder must use to apply selective disclosure, include all the messages, not just the ones disclosed. Based on the above, the basic idea of the Holder's authentication mechanism is to create a secret message and submit a commitment to that message to the Issuer. The Issuer will sign it along with the other claims of the VC and return the signature to the Holder.

Then, if the Holder wants to create a proof, they must know all the signed messages, meaning they must also know the secret message they committed to the Issuer. Anyone else that does not possess that secret message will not be able to create a valid proof for that credential, essentially, authenticating the Holder. A drawback of the above method is that we essentially allow the Holder to "insert" any message they want to the signature, something obviously problematic.

Message Indexes

To circumvent that problem, we will draw again attention to the selective disclosure Section 2.2.5 and more precisely to the Equations 2.8. As we have mentioned there, the Verifier will receive the proof $(A', \bar{A}, d, \pi)$ and the messages $\{m_i\}_{i \in D}$ where D the indexes of the disclosed messages. Then they will attempt to verify the Equations 2.8 by validating the proof π. To do that, first the Verifier will calculate the parts of the two equations 2.8 that contain the elements known to them. More precisely, the Verifier will calculate V_1 and V_2 where,

$$\begin{aligned} V_1 =& \bar{A}/d \\ V_2 =& g_1 \prod_{i \in D} h_i^{m_i} \end{aligned} \tag{3.3}$$

Then using π, they should confirm if the equations 2.8 hold or not, or more specifically, if the following equations hold,

$$\begin{aligned} V_1 =& A'^{-\epsilon} h_0^{r_2} \\ V_2 =& d^{r_3} h_0^{-s'} \prod_{i \notin D} h_i^{-m_i} \end{aligned} \tag{3.4}$$

The key observation here is that the Verifier will not receive the value V_2 (equations 3.3) but they would calculate it themselves. To do that correctly however, they should know the exact mapping between the elements h_i and the messages m_i for $i \in D$ so that they can compute $h_i^{m_i}$. That mapping will be also supplied by the Holder.

Authentication Method

Going back to the problem of the Holder been able to submit and have signed any message they want as part of their authentication process, the basic idea of the solution is to reserve an element g_{auth} in $\mathbb{G}_1$ and force the Holder to submit their secret message m_{auth} using only g_{auth} (meaning that the Holder's commitment can be only of the form $C = g_{auth}^{m_{auth}}$). Then, since the Verifier must always know the mapping between the messages m_i and the elements they will raise to the power of m_i, we could warn them (e.g., make it part of the spec) to not accept any messages that are mapped to that reserved for the Holder's authentication element g_{auth}. What we achieve with this approach, is that even though

the Holder could choose whatever message m_{auth} they want to be committed and signed, they can't take advantage of it, since if they try to reveal it and map it to the element g_{auth} the Verifier will dismiss the proof and if they map it to a different element the proof's validation will fail.

The last missing piece of the solution is a way to force the Holder to use g_{auth}, or more precisely, a way for the Issuer to verify that the commitment C they received from the Holder, is made using g_{auth}. There are many cryptographic protocols that could be used here, however perhaps the most efficient way (proposed by the BBS+ LD proofs spec) is to use cryptographic BLS keys and signatures [BLS01]. The basic concept is to use a public BLS key ($g_{auth}^{m_{auth}}$) as the commitment, with the secret message m_{auth} being the corresponding secret key. The Holder, besides the commitment C will also send a random message r signed with their private BLS key m_{auth}. The Issuer will verify the signature on r assuming that the Holder used g_{auth} to create the commitment. The signature will be successfully validated, only if the Holder truly used g_{auth} to create their commitment (public BLS key).

3.3 Protocol Exploits

As already mentioned, we managed to discover two security vulnerabilities in the specification of the linked data based BBS+ proofs for Verifiable Credentials. The first vulnerability is in the URDNA2015 canonicalization algorithm used to "break" a credential to a list of messages (see section 3.2.1). The second is in the proposed Holder authentication procedure (section 3.2.2). Both vulnerabilities were presented in the W3C working group of the BBS+ LD proofs spec. Here we will also present some of the solutions that we discussed with the members of the working group as well with some implementers of the spec, some of which have already been implemented, after we presented the two vulnerabilities below.

3.3.1 URDNA2015 Security Vulnerability

Up until now, any criticism exercised on the URDNA2015 algorithm was mainly the result of the algorithm's complexity and there were not any known security vulnerabilities. Here, we will show that when the Holder uses selective disclosure to hide some of the claims on a credential, an adversary under certain conditions could take advantage of the blank node labels produced by the URDNA2015 algorithm to either reduce the possible range for some of the Holder's hidden claims or, in some cases, even reveal them completely. Furthermore, the adversary in this attack is completely passive, meaning that they only need to observe the credential and not interact with either the Holder or the Issuer in any

way. After we managed to reproduce the exploit for various examples[2], we presented the vulnerability to the W3C working group[3].

Let's assume that the Holder has a Credential in their possession which they want to send to a Verifier. However, they only want to disclose some of the claims on that credential and keep others private. To create a proof of knowledge, the Holder will use the URDNA2015 algorithm to calculate the blank node labels and retrieve the edges of the graph as the messages to use to create the proof (the ones also signed by the Issuer). Observe that this procedure is possible because the knowledge graph corresponding to the credential signed by the Issuer and the knowledge graph of the credential that the Holder possesses are isomorphic, meaning that the URDNA2015 algorithm will assign the same blank node labels to both (clearly both the Issuer and the Holder must get the same messages from the VC).

The graph of the credential that the Verifier will receive on the other hand will not be isomorphic to either one of the Holder's or Issuer's knowledge graphs. That is because the Holder will use selective disclosure, meaning that in the credential the Verifier will get, some of the claims it originally had, will be missing (kept secret by the Holder). As a result, the Holder should inform the Verifier about the blank node labels they computed, since the Verifier will not be able to compute them themselves. That information will be entered in the VC in the form of "id's" in the nested objects corresponding to the blank nodes which the Holder wants to "publish" their labels. As an example consider the following credential and the corresponding knowledge graph of Figure 3.3. Figure 3.4 shows the credential and knowledge graph the Verifier would receive, if the Holder hides their first name.

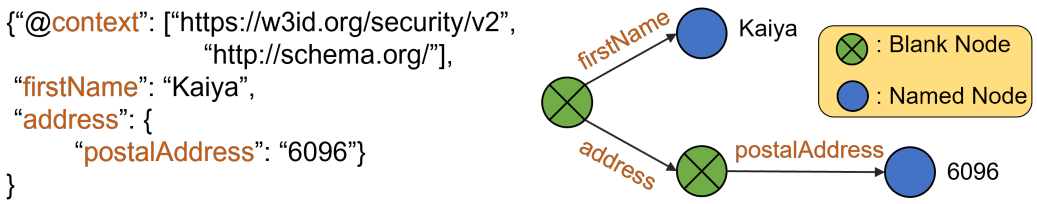

Fig. 3.3: JSON LD credential example and corresponding knowledge graph.

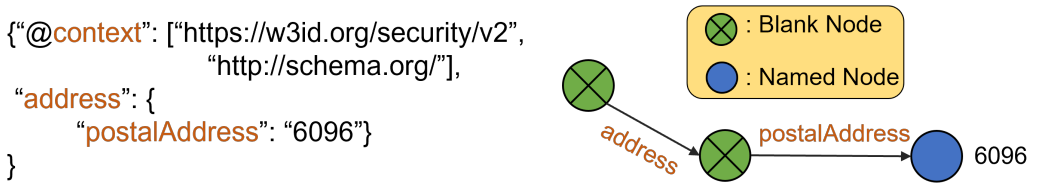

Fig. 3.4: JSON LD credential and corresponding knowledge graph the Verifier will receive if the Holder uses selective disclosure to hide their "firstName" from their credential of Figure 3.3

[2]Three examples of our exploit, in node.js: https://github.com/BasileiosKal/CanonizeVulnerability
[3]URDNA2015 exploit report: https://github.com/w3c-ccg/ldp-bbs2020/issues/60

The messages that the Issuer and the Holder will get from the credential of Figure 3.3 as well as the messages that the Verifier will get from the credential they will receive from the Holder (Figure 3.4) are shown in Table 3.1.

Messages Returned from URDNA2015	
Holder/Issuer	Verifier
< _:c14n1> <firstName> <Kaiya>	< _:c14n0> <address> <_c14n1>
<_:c14n1> <address> <_c14n0>	<_:c14n1> <postalAdress> <6096>
<_:c14n0> <postalAdress> <6096>	

Tab. 3.1: Messages returned from the URDNA2015 algorithm when is used from the Holder/Issuer and the Verifier with the credential of Figure 3.3 and Figure 3.4 correspondingly

Notice that the messages the Verifier will get from URDNA2015 are different than the ones the Issuer and the Holder had. The result of that miss-match will be that the verification process will fail, since the Verifier will try to validate a signature on messages that are not signed by the Issuer. To solve that problem, instead of the credential of Figure 3.4, the Holder will send to the Verifier the credential of Figure 3.5 (notice the addition of the "id" fields). The Verifier, seen that "id", will know which is the correct blank node label they should use (for example that the blank node label of the "address" nested object is "_:c14n0" and not "_:c14n1").

```
{"@context": ["https://w3id.org/security/v2",
              "http://schema.org/"],
"@id": "urn:bnid:_:c14n1",
 "address": {
      "@id": "urn:bnid:_:c14n0",
      "postalAddress": "6096"}
}
```

Fig. 3.5: JSON LD credential example and corresponding knowledge graph.

As already mentioned, the order of those labels corresponds to the order of the "first-degree-hashes" of the blank nodes in the graph. Let $h(Q)$ be the first-degree-hash of the reference set Q (neighbors of a blank node), where h is a cryptographic hash function (usually sha256). Each first-degree-hash encodes the information directly linked with a blank node which may contain claims that the Holder wants to keep private. If that hash or some information about that hash is revealed, it could lead to information about the hidden claim being leaked ass well (note that the whole procedure was supposed to be zero-knowledge). Of course, the "first-degree-hash" $h(Q)$ itself will not be completely revealed when the Holder conceals some of the claims included in the corresponding reference set Q. If however the Holder reveals some of the canonical blank node labels, as in the example above, they could also reveal the order of that "first-degree-hash", relevant to some other known value (e.g., another "first-degree-hash" that the claims in its reference set are revealed, meaning that the hash will be known). This fact, combined with the

commonly very small range of possible claims (e.g., first names, ages etc.) and some additional external knowledge an adversary may possess about the credential (e.g., that it contains the Subject's first name, without knowing that name), leads to the vulnerability.

Reference Sets Elements	
Q_1	< _:a, firstName, Kaiya>, <_:a, address, _:z>
Q_2	<_:z, address, _:a>, <_:a, postalAddress, "6096">

Tab. 3.2: Reference sets Q_1 and Q_2.

As an example, consider again the credential of Figure 3.3. The reference sets of each blank node are shown in Table 3.2. Observe that the first reference set Q_1 is only defined from the first claim {"firstName": "Kaiya"} while the second set Q_2 is only defined from the second claim {"postalAddress": "6096"}. Let's also assume that the adversary will know that the credential contains the subjects first name (in many cases in practice the credential's structure will be known, like government IDs, student IDs, passports etc.). Let $Q_1(name)$ be the first reference set as a function of the name ($Q_1(Kaiya) = Q_1$ of Table 3.2). Since we assume that the Holder will reveal only their address and not their first name, the Verifier will receive the credential of Figure 3.5. From that they will learn that address = 6096 which will reveal all the elements of the reference set Q_2 of Table 3.2. As a result, the Verifier (or any adversary observing the credential of Figure 3.5) will be able to compute the "first-degree-hash" of the second blank node; $h(Q_2)$. Since the adversary doesn't know the Subject's first name, they will not know all the elements of Q_1 meaning they can't directly compute the "first-degree-hash" of the first blank node. They can observe however that the second blank node's label is "_:c14n0" from the "id" field inserted in the "address" object of the credential (Figure 3.5). From that, although they can't calculate the "first-degree-hash" of the first blank node $h(Q_1)$ directly, they can deduce that whatever the "name" of the subject is, it must be one so that $h(Q_1(name)) > h(Q_2)$ holds.

From there it is relatively trivial for the adversary to check all the possible English names to find the ones for which $h(Q_1(name))$ is larger than $h(Q_2)$. That search took only 0.18 seconds on an Intel i5 with 3.2 GHz of clock speed and 16GB of RAM. For our example, the only name for which that inequality holds is the name "Kayia", which means that even though the Holder wanted to keep their first name private, the adversary will manage to reveal it completely.

In the above example we assumed that the adversary would have some information about the structure of the credential. There are many ways someone could obtain this kind of information. First, the credential's structure is likely to be standard and known (e.g., government IDs, passports etc.) or revealed from other users that use a similar credential.

Additionally, even if there is no way to get any information about the credential's structure, there is still information being leaked. In the previous example, if the adversary didn't know that the secret claim was the Subject's first name, they would still get the information that $h(Q_1) > h(Q_2)$ where Q_1 contains -some- secret claims, which breaks the advertised zero-knowledge property. Lastly, up-until now we have assumed that the adversary is passive. However, that may not always be the case and by interacting with the Holder or the Issuer in some way, an adversary could get the necessary information about the structure of the credential. For example, consider the following credential of Figure 3.6, which contains the names and ages of the Holder's two children. Assume a situation, where the Holder wants to reveal the name and age of their oldest child (Ioanna) (perhaps to attend Ioanna's Thesis presentation) but also wants to keep secret the name and age of their youngest child (Kale). From the message indexes (see Section 3.2.2) the adversary will know that there are two claims in the credential that are not revealed (and where those claims would go in the JSON LD structure). If from some other source or by interacting with the Holder they learn that the Holder has two kids, the same method could be used to completely reveal both the name and the age of the Holder's youngest child.


```
{
 "@context": [
      "http://schema.org/",
      "https://w3id.org/security/v2"],
 "issuer": "did:example:489398593",
 "id": "https://issuer.com/credentials/123456",
 "credentialSubject": {
      "@id": "did:example:654321",
      "givenName": "Jane",
      "familyName": "Doe",
      "email": "jane.doe@example.com",
      "kids": [
           {"givenName": "Kale",
            "age": "9"},
           {"givenName": "Ioanna",
            "age": "28"}]}
}
```


Fig. 3.6: JSON LD credential example. If the Holder wants to keep the name and age of their youngest kid the adversary will be able to unveil them

The above examples examine the case where the adversary will be able to completely reveal some of the Holder's secrets. Of course, that will not always be possible. In the first example, if the Holder's postal address was "2513" instead of "6069", there is not only one first English name for which $h(Q_1(name) > h(Q_2)$. However, there is still a leak. In that example, the adversary will still be able to reduce the space of possible English names to only 18 (from 19.000). Additionally, as noted by Mr. Dave Longley, (CTO at DigitalBazzat and author in the JSON-LD W3C Specification), the above method will most likely reveal the hidden claim if the corresponding value is Boolean. In general, the smaller the values range of the un-disclosed claims are, the better the method will work.

Proposed Solution

The best solution thus far, proposed by the W3C working group members, is to randomize the blank node labels. That can be done in various ways. After we presented the vulnerability to the W3C working group, Mattr (on of the main implements of BBS+ and authors of the specification) implemented an initial mitigation by adding a random value (nonce) in the outer level of the credential[4]. The effect of that nonce is to randomize some of the blank node labels, which will result in shuffling the order of the rest. One random value however is not enough, and in large credentials there will be the need for additional randomness. To achieve this, we proposed to use the nonce already added as a seed to generate random values that will be added in every nested object. Essentially, we will add a random neighbor to each blank node in the graph, effectively randomizing the reference sets and hence the "first-degree-hashes". Considering again the example of Figure 3.3, now $Q_1(name)$ will also contain a random unknown value, meaning that the adversary will not be able to calculate $h(Q_1(name))$ by simply iterating through the different names. Another discussed solution is to calculate the labels in the usual way and pass them through an irreversible randomization function, effectively hiding them.

3.3.2 Holder Authentication Security Vulnerability

The second vulnerability is concerning the proposed Holder Authentication method. That vulnerability does not lie to the basic method as presented in Section 3.2.2. It rises however, from the proposed implementation of that method, from the BBS+ LD proofs spec. That proposal is to use one common PKI for both the BBS+ and the BLS signatures. There are many reasons to use a common PKI. First, the two signature schemes are very similar, with both using the same BLS12-381 elliptic curves, calculating public/private keys in the same way, using pairings to validate a signature etc. As a result, a common PKI would extremely simplify any implementation. Additionally, using the same PKI mutes the need for separate specifications, reviews etc. For these reasons, the BBS+ spec proposed to use the same base generator of the finite group $\mathbb{G}_1$ (the first BLS12-381 elliptic curve) for both creating BBS+ signatures and the BLS public key (creating the common PKI). Doing this however, violates the -implicit- assumptions under which the BBS+ signatures have proven to be unforgeable against adaptive plain-text attacks. In the following we present a method (that we also reported to W3C [5]) to forge a BBS+ signature, taking advantage of the fact that both PKI's use the same base generator for $\mathbb{G}_1$. Again, that method could work with a passive adversary.

Notice from Section 2.2.4 and equation 2.5 that given the groups $\mathbb{G}_1$, the base generator g_1 of $\mathbb{G}_1$, the messages $\{m_1, m_2, ..., m_L\}$ and a signature on those messages (A, e, s), the

[4]URDNA2015 fix: https://github.com/mattrglobal/jsonld-signatures-bbs/tree/tl/blank-node-determinism-fix
[5]Holder authentication exploit report: https://github.com/w3c-ccg/ldp-bbs2020/issues/37

correct form for A has g_1 raised only to the power of 1. Lets assume that to authenticate the Holder, we use BLS keys in $\mathbb{G}_1$ with the same base generator g_1. Then, for a BLS secret key BLS_{SK} the corresponding public key will be $BLS_{PK} = g_1^{BLS_{SK}}$. Remember from section 3.2.2, that the BLS_{PK} will become a commitment which the Issuer will include to the signature, treating the committed message (in this case the BLS_{SK}) as any other signed claim. The new signature (with the commitment) (A_c, e_c, s_c) will have $e_c = e$, $s_c = s$ and,

$$A_c = [g_1 h_0^s \; g_1^{BLS_{SK}} h_1^{m_1} h_2^{m_2} \cdots h_L^{m_L}]^{\frac{1}{e+x}} \tag{3.5}$$

Note that in the equation 3.5 for A_c, g_1 appears twice, once to the power of 1 as before, and once to the power of BLS_{SK}. However, that itself does not make the signature (A_c, e_c, s_c) invalid, since, the verification Equation 2.6 from Section 2.2.4 continuous to hold.

Lets assume that an adversary observes the messages $\{m_1, m_2, ..., m_L\}$ and the signature (A_c, e_c, s_c). Let the adversary choose a $k \neq 1 \in \mathbb{Z}_p$. We can see that by calculating A_c^k the attacker can get a forged BBS+ signature since,

$$\begin{aligned} A_c^k &= ([g_1 h_0^s \; g_1^{BLS_{SK}} h_1^{m_1} h_2^{m_2} \cdots h_L^{m_L}]^{\frac{1}{e+x}})^k = \\ &= [g_1^k h_0^{k \cdot s} \; g_1^{k \cdot BLS_{SK}} h_1^{k \cdot m_1} h_2^{k \cdot m_2} \cdots h_L^{k \cdot m_L}]^{\frac{1}{e+x}} = \\ &= [g_1 g_1^{k-1} h_0^{k \cdot s} \; g_1^{k \cdot BLS_{SK}} h_1^{k \cdot m_1} h_2^{k \cdot m_2} \cdots h_L^{k \cdot m_L}]^{\frac{1}{e+x}} => \\ => A_c^k &= [g_1 h_0^{k \cdot s} \; g_1^{k \cdot BLS_{SK} + k - 1} h_1^{k \cdot m_1} h_2^{k \cdot m_2} \cdots h_L^{k \cdot m_L}]^{\frac{1}{e+x}} \end{aligned} \tag{3.6}$$

Observe from the above equation 3.6 that we can write A_c^k as

$$A_c^k = [g_1 h_0^{s'} \; g_1^{BLS'_{SK}} h_1^{m'_1} h_2^{m'_2} \cdots h_L^{k m'_L}]^{\frac{1}{e+x}} \tag{3.7}$$

Then A_c^k has the same form as A_c from the equation 3.5 for,

$$\begin{aligned} m'_i &= k \cdot m_i, \\ BLS'_{SK} &= k \cdot BLS_{SK} + k - 1, \\ s'_c &= k \cdot s_c \end{aligned} \tag{3.8}$$

As a result, if (A_c, e_c, s_c) is a valid signature on the messages $\{m_i\}_{i \in \{1,2,...,L\}}$ and the BLS key BLS_{SK}, then (A_c^k, e_c, s'_c) is also a valid signature on the messages $\{m'_i\}_{i \in \{1,2,...,L\}}$ and the secret BLS key BLS'_{SK}. We can formally prove the above by noticing that, assuming (A_c, e_c, s_c) is valid for the messages $\{m_i\}_{i \in \{1,2,...,L\}}$ and the BLS key BLS_{SK}, it holds that $e(A_c, PK_{Issuer}\, h_0^{e_c}) = e(g_1 h_0^{s_c} g_1^{BLS_{SK}} \prod_{i \in \{1,2,..,L\}} h_i^{m_i}, h_0)$ which means that,

$$\begin{aligned} & e(A_c^k, PK_{Issuer}\, h_0^{e_c}) = e(A_c, PK_{Issuer}\, h_0^{e_c})^k = \\ & = e(g_1 h_0^{s_c} g_1^{BLS_{SK}} \prod_{i \in \{1,2,..,L\}} h_i^{m_i}, h_0)^k = e((g_1 h_0^{s_c} g_1^{BLS_{SK}} \prod_{i \in \{1,2,..,L\}} h_i^{m_i})^k, h_0) \\ & => e(A_c^k, PK_{Issuer}\, h_0^{e_c}) = e(g_1 h_0^{s'_c} g_1^{BLS'_{SK}} \prod_{i \in \{1,2,..,L\}} h_i^{m'_i}, h_0) \end{aligned} \tag{3.9}$$

Which proves that $(A_c^k, e_c, k \cdot s_c)$ is a valid BBS+ signature on the messages $\{m'_i\}_{i\in\{1,2,...,L\}}$ with the committed secret BLS'_{SK}. Since $m'_i = k \cdot m_i \neq m_i$, we have achieved a forgery of a BBS+ signature.

The intuition for the above is that, after A_c is raised to the power of k, all elements of the signature -including g_1- are subsequently also raised to the power of k. Since, A in a correct BBS+ signature (see Equation 3.1 in Section 2.2.4) contains g_1 raised only to the power of 1, $A^k = (g_1^k \cdot h_0^{k\cdot s} \cdot h_1^{k\cdot m_1}...)^{(1/(e+x))}$ will not be a valid signature normally. However, if commitments are allowed to be made using g_1 (like the BLS key pairs), an adversary could just break g_1^k to g_1 and g_1^{k-1}. Then they could use the first part (g_1) as the standard part of the signature and the second part (g_1^{k-1}) as a commitment to $k-1$ (since we allow commitments to use g_1). As a result, raising A_c to any power k produces a valid -forged- signature.

Note also, that in the beginning k was chosen arbitrarily. As a result, the adversary can choose different integers k to get different forged BBS+ signatures on the messages $m'_i = k \cdot m_i$. For example assume that the adversary wants to get a BBS+ signature on the message m_{adv}, that was not originally signed by the Issuer ($m_{adv} \notin \{m_i\}_{i\in\{1,2,..,L\}}$). All they need to do is to find a message $m_s \in \{m_i\}_{i\in\{1,2,..,L\}}$ that was signed by the Issuer, and a integer $k \in \mathbb{Z}_p$ so that $m_{adv} = k \cdot m_s$. This can be done by using the binary Extended Euclidean algorithm to calculate m_s^{-1} (the inverse of m_s), which runs in time linear to $log_2(p)$ (the number of bits of the message m_s). Then the adversary can simply set $k = m_{adv} \cdot m_s^{-1} \in \mathbb{Z}_p$ and calculate A_c^k to get a valid forged BBS+ signature for the message m_{adv}.

Proposed Solution

As already mentioned, the vulnerability is caused by the fact that using a BLS public key that was created using the base generator g_1 of $\mathbb{G}_1$, we allow g_1 to appear multiple times in the equation of A of a BBS+ signature: (A, e, s). That being said, we don't want to dismiss the use of BLS keys to authenticate the Holder, since there are use cases where binding a VC to a public BLS key is useful, for example, a Holder could have purchased a device with a burned BLS key for which they want a credential or they want to use a second device in a 2-factors authentication scheme. To achieve a secure binding of a BLS key to a credential we propose to reserve a different generator g'_1 to be used with the BLS PKI. This essentially decouples the BBS+ and BLS PKI's, which means that we could face a greater implementation complexity, but we believe it will only be a minor inconvenience, since the only thing that changes is the base generator of the finite groups. Another solution could be to use as a commitment the BLS signature on a random message r; $H(r)^{BLS_{SK}}$, that the Holder sends to the Issuer to prove possession of the BLS_{SK}. Note that this way, we are still binding the credential to a BLS key pair. That being said, it may still be

best practice to use different PKIs for the two digital signature protocols, to avoid other potential vulnerabilities.

4 JSON BBS+ Signatures

In the previous chapters we considered the current state-of-the-art for privacy preserving Verifiable Credentials and the current implementations and specifications which are focused on the linked data approach. As we saw in previous chapters, linked data are suffering from various security and privacy issues. Aside from the security vulnerabilities we pointed out (Section 3.2.1 and 3.2.2), there are still worries regarding various parts of the existing specification. Furthermore, even the solutions we proposed for the protocol exploits we discovered, are essentially "patches" and they do not guarantee that there will not be any other security vulnerability. For those reasons we propose an alternative pipeline for signing Verifiable Credentials with BBS+. The main difference is that instead of using JSON LD (Linked Data) to represent a credential, we will use a standard JSON format. This will allow us to create a mathematical modeling of the JSON representation with which we can prove the end-to-end security of our scheme. Finally, we will show that we can get an improved performance with our algorithm for breaking a credential to a list of messages (to be signed with BBS+) compared to the one used thus far (URDNA2015).

4.1 Modeling JSON

When considering JSON LD credentials, the modeling was mainly done using graphs and more specifically, the knowledge graph representing the claims of the VC. Cryptographic signature protocols however, mainly work on a bit-array that represents the message being signed (or in the case of BBS+ a set of bit-arrays) and not on graphs. This mismatch can cause not only various security vulnerabilities, like the URDNA2015 exploit we discovered (Section 3.3.1), but also makes it very hard to create provable secure protocols. The goal of our modeling is to represent the credential in a way closer to the one expected from the BBS+ algorithms, i.e., as a set of bit-arrays. To do that, instead of graphs we will use finite functions, that can be naturally (i.e., by definition) transformed into a set.

The basic concept of our modeling is the observation that a JSON representation is comprised from two things: the **Structure** and the **Values**. We will model those two separately and consider a JSON data-structure to be a combination of both. As a public parameter we will first define a non-zero positive integer $n \in \mathbb{N}$. Let $[n] = \{1, 2, ..., n\}$, be the set of all integers from 1 to n.

Structure

Let K be the set of all possible "keys" that can appear in the JSON file and $\mathbb{P}(K)$ be the set of finite sets with elements from K. We define the **structure** of a JSON file to be K along with an injective function ϕ,

$$\phi(x) : [n] -> \mathbb{P}(K) \tag{4.1}$$

Values

Similarly, we define the **values** of a credential as a set V of all the possible literal values that can appear in the JSON file, along with a function g,

$$g(x) : [n] -> V \tag{4.2}$$

Note that we make no assumptions for g, in contrast with ϕ that we define to be injective. We now end up on the following definition for a JSON data-structure.

Definition 1. We define a JSON data-structure J to be $J = (K, V, n, \phi, g)$, or since K and V can be as extensive as we want, for simplicity we will define a JSON file as $J = (n, \phi, g)$.

The intuition behind the proposed modeling is that $[n]$ will map each place in the JSON data-structure where a literal value could appear to an integer. Then ϕ will use those integers to map each position in the JSON with a literal value to the set of keys that will lead to that literal value and g will map each literal value position to that literal value. For example, consider the credential of Figure 4.1 with each place that a literal value can go mapped to an integer in $[6] = \{1, 2, ..., 6\}$.

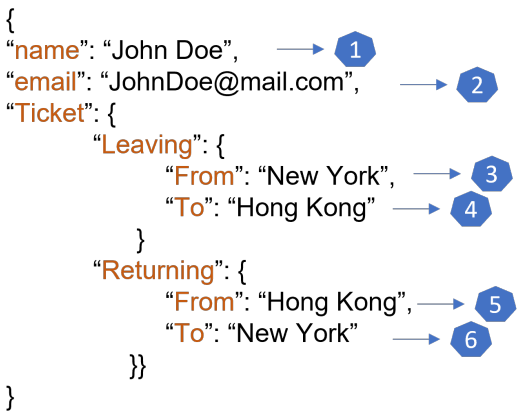

Fig. 4.1: Modeling a JSON data-structure example.

The values of the ϕ and g functions can be seen at Table 4.1. Observe that $\forall i_0, i_1 \in [6]$, with $i_0 \neq i_1 => \phi(i_0) \neq \phi(i_1)$, while $g(3) = g(6)$ and $g(4) = g(5)$.

n	ϕ	g
1	(name)	John Doe
2	(email)	JahnDoe@bestMail.com
3	(Ticket, Leaving, From)	New York
4	(Ticket, Leaving, To)	Honk Kong
5	(Ticket, Returning, From)	Honk Kong
6	(Ticket, Returning, To)	New York

Tab. 4.1: Values of the ϕ and g functions of the Credential 4.1

From the example above it can be seen that the order with which we map each literal value position in the JSON to an integer, should not matter. For example, we could map the "name" field to the integer 2 and the "email" field to the integer 1 and still have the same JSON file. For that reason, we define the equality between two JSON data-structures as following

Definition 2. Let $J_1 = (n_1, \phi_1, g_1)$ and $J_2 = (n_2, \phi_2, g_2)$ be two JSON data-structures. We define that $J_1 = J_2$ if and only if $n_1 = n_2 = n$ and there is a permutation σ of $[n]$, such that $\phi_1 = \phi_2(\sigma)$ and $g_1 = g_2(\sigma)$.

An important part in the Definition 1 of JSON is that the function ϕ is injective. However, from the Definition 2 above, it may seem that this is not always the case if the JSON file contains a list. For example, consider the JSON of Figure 2.4. We can observe that if we calculate the ϕ function as in the example JSON of Figure 4.1, we may get $\phi(3) = \phi(5) =$ (credentialSubject, escorts, name). For this reason, in cases that there is a list on the JSON, we consider the index of the list's elements as a key as well. We essentially, transform the list into a map between the list index and corresponding element. As a result, for the JSON of Figure 2.4 we will get that $\phi(3) =$ (credentialSubject, escorts, 1, name) and $\phi(5) =$ (credentialSubject, escorts, 2, name), essentially making ϕ again injective.

A drawback of our modeling is that following Definitions 1 and 2, if the JSON file contains a list and we change the order of the elements in that list we will get a different JSON file (per the Definition 2 of equality). Although that caveat does not seem to have any significance in practice, especially for cryptographic applications where the file should not be able to change in any way, additional work could be done to extend the above modeling to also account for that case (defining broader classes of equality etc.). For our applications however, those definitions will suffice.

Notice also that our modeling does not directly work with JSON LD. If for example we consider the credential of Figure 3.1, we can observe that, if we map the position of the value "did:example:abc12345" with the integer 3, we could define the structure with a function ϕ_1 with $\phi_1(3) =$ (Subject, @id) or with function ϕ_2 with $\phi_2(3) =$

(Subject, residence, @owner). Notice though that there is not a permutation σ so that $\phi_1 = \phi_2(\sigma)$ meaning that per our modeling ϕ_1 and ϕ_2 would lead to two different credentials. From the corresponding knowledge graph of Figure 3.2 we can see that the problem lies on the fact that there are two different paths leading to the value "did:example:abc12345". In a JSON credential, which in a graph representation will be a tree graph, something like that is not possible. That being said, we believe that our definitions could easily be extended to the case of linked data as well, by considering only the shortest paths or with some additional definitions, but that is out of scope for this work.

Continuing, we will prove that each JSON can also be defined as a set of different values by proving an alternative definition of equality. We will use this alternative definition (equivalent to Definition 2) later to sign a JSON file.

Lemma 1. *Given* $J_1 = (n_1, \phi_1, g_1)$ *and* $J_2 = (n_2, \phi_2, g_2)$, $J_1 = J_2$ *if and only if* $n_1 = n_2 = n$ *and* $\{(\phi_1(i), g_1(i))|\ i \in [n]\} = \{(\phi_2(i), g_2(i))|\ i \in [n]\}$.

Proof. Lets assume that $n_1 = n_2$ and that $\{(\phi_1(i), g_1(i))\}_{i\in[n]} = \{(\phi_2(i), g_2(i))\}_{i\in[n]}$. Since $\{(\phi_1(i), g_1(i))\}_{i\in[n]} = \{(\phi_2(i), g_2(i))\}_{i\in[n]}$, $\forall i \in [n]\ \exists i' \in [n]$ so that $\phi_1(i) = \phi_2(i')$ Lets consider a function $\sigma : [n] -> [n]$ with $\sigma(i) = i' <=> \phi_1(i) = \phi_2(i')$. We will show that σ is a permutation.

1) Let $i_0, i_1 \in [n]$ and $i'_0, i'_1 \in [n]$ with $\phi_1(i_0) = \phi_2(i'_0)$ and $\phi_1(i_1) = \phi_2(i'_1)$. If $i_0 = i_1 => \phi_1(i_0) = \phi_1(i_1) = \phi_2(i'_0) = \phi_2(i'_1) => i'_0 = i'_1$ since ϕ_2 injective. As a result σ is a function.

2) Let now $i_0 \neq i_1$. Lets assume $\sigma(i_0) = \sigma(i_1) => i'_0 = i'_1$ meaning that $\phi_2(i'_0) = \phi_2(i'_1) => \phi_1(i_0) = \phi_1(i_1) => i_0 = i_1$ since ϕ_1 is injective. We arrived in a contradiction and as a result, σ is injective.

3) Let an $i \in [n]$. Since $\{(\phi_1(i), g_1(i))\}_{i\in[n]} = \{(\phi_2(i), g_2(i))\}_{i\in[n]}$ we can conclude that $\exists i' \in [n]$ so that $(\phi_2(i), g_2(i)) = (\phi_1(i'), g_1(i'))$ and as a result $\phi_1(i') = \phi_2(i) => \sigma(i') = i$. As a result, the σ function is also bijective. We conclude then that σ is a permutation.

From the definition of the σ permutation it is easy to see that $\phi_1(i) = \phi_2(\sigma(i))$ and that $g_1(i) = g_2(\sigma(i))$ which means that $J_1 = J_2$.

The opposite direction, meaning that if $J_1 = J_2$ then $n_1 = n_2 = n$ and $\{(\phi_1(i), g_1(i))\}_{i\in[n]} = \{(\phi_2(i), g_2(i))\}_{i\in[n]}$ is trivial. □

4.2 Signing JSON Credentials

We can now easily define a BBS+ signature on a JSON credential using the proposed modeling. Let $J = (n, \phi, g)$ be a credential in JSON form. Let also $SIGN_{BBS+}$ to be the signing algorithm of the BBS+ signatures (Section 2.2.4). We define a BBS+ signature on that credential to be,

$$\text{Credential BBS+ Signature} = J_{sign} = SIGN_{BBS+}(\{(\phi(i), g(i))\}_{i\in[n]}) \tag{4.3}$$

Similarly, if POK_{BBS+} is a proof of knowledge (PoK) algorithm of the BBS+ schema (Section 2.2.5) then we define a PoK for J and the BBS+ signature J_{sign} as,

$$\text{Credential BBS+ POK} = J_{pok} = POK_{BBS+}(J_{sign}, \{(\phi(i), g(i))\}_{i\in[n]}) \tag{4.4}$$

Using the definitions above and the JSON modeling we proposed (Section 4.1), we can easily prove that a BBS+ signature on a credential J_{sign} is unforgeable and that a proof of knowledge J_{pok} is a zero-knowledge proof of knowledge, by simply reducing those properties to the corresponding properties of BBS+. Next, we will prove the unforgeability property under adaptive plain-text attacks. The zero-knowledge proof-of-knowledge is very similar and almost trivial.

Lemma 2. *JSON BBS+ signatures are unforgeable under adaptive plain-text attacks.*

Proof. Let an adversary that through an adaptive plain-text attack has acquire q JSON BBS+ signatures for the JSON files $J^1, J^2, ..., J^q$. From 4.3, we can see that essentially the adversary will have q BBS+ signatures J^k_{sign} on the messages $M_k = \{(\phi_k(i), g_K(i))\}_{i\in[n_k]}$ for $k = \{1, 2, ..., q\}$. Lets assume that using those signatures they can forge a signature J^{forge}_{sign} for a (forged) JSON file $J^{forge} \neq J^k \ \forall k \in \{1, 2, ..., q\}$. The J^{forge}_{sign} will be a BBS+ signature on the messages $M_f = \{(\phi_f(i), g_f(i))\}_{i\in[n_f]}$. Consider the following two cases,

First let the messages M_f not be part of the messages $\{M_k\}_{k\in\{1,2,..,q\}}$ (the ones signed by the Issuer). Then the adversary will have achieved a forgery of a BBS+ signature on the messages M_f, that are not previously signed by the Issuer using an adaptive plain-text attack, which is a contradiction.

Assume next that $M_f \in \{M_k\}_{k\in\{1,2,...,q\}}$. Then $\exists k_0 \in \{1, 2, ..., 1\}$ so that $M_f = M_{k_0}$ and $n_f = n_{k_0}$. From Lemma 1 we conclude that $J^{forged} = J^{k_0}$, which also is a contradiction. □

4.3 Experimental Results

The following results showcase the performance difference between dissecting a JSON-LD credential using a canonicalization algorithm, and a JSON credential using the proposed JSONdissect algorithm (Algorithm 1). As dissecting, we denote the mapping of a credential (either in JSON or JSON-LD form) to a list of messages that will later be signed with a BBS+ signature[1]. To dissect JSON-LD Verifiable Credentials (VCs), we used the URDNA2015 canonicalization algorithm. As already mentioned that algorithm is considered the state-of-the-art for dissecting JSON-LD VCs. For dissecting a JSON VC, we used the JSONdissect function, shown in Algorithm 1 below[2], that returns the set of values $\{(\phi(i), g(i))|i \in [n]\}$, in compliance with the modeling we proposed in Section 4.1.

Algorithm 1 Dissect a JSON credential

```
function JSONDISSECT(VC)
    Messages ← []
    Claim ← None
    procedure RECURSE(VC)
        for key ∈ VC do   ▷ If VC is a list, the keys will be the indexes of its elements.
            if typeof VC[key] == JSONobj then
                Claim ← Claim + "key."
                RECURSE(VC[key])
            else
                Claim ← Claim + "_VC[key]"
                Messages.push(Claim)
                Claim ← None
    return Messages
```

4.3.1 Set Up

To benchmark the two different algorithms, we generated credentials with various numbers of blank nodes, key/value pairs (which we will denote simply as claims) and maximum depth. More specifically, each figure below corresponds to a specific credential core (of set max depth and number of blank nodes) with an increasing number of claims. Each claim is randomly generated with a length of 40 bytes (20 bytes key with 20 bytes value). After the credentials are generated, we run the same benchmarking procedure for both algorithms and compared the results. The benchmarking procedure is as follows,

1. Each algorithm is executed as many times as possible for a duration of 50 msec.

2. During each run, the duration of the algorithm's execution is measured.

[1]The terms 'normalization' or 'canonicalization' are not used in the case of JSON credentials, since those terms entail additional meaning in the linked data context.
[2]Our implementation (node.js) and the benchmarking code: https://github.com/BasileiosKal/json-bbs

3. Finally, the mean duration is returned.

The above procedure was executed in a personal computer using an Intel i5 with 3.2 GHz clock speed and 16 GBs of RAM. Also note the following,

1. A JSON-LD credential before is canonicalized, it must first be expanded. The expansion process essentially brings the credential to the proper form, so it can be later turned to an RDF dataset (knowledge graph representation) and be canonicalized. In the results below, we don't include the expansion algorithm as part of the process of dissecting a JSON-LD VC (all the credentials were already expanded). The reason was to get a more accurate performance measurement of just the URDNA2015 algorithm. In practice, however, the expansion algorithm will most likely be used, and it will add an overhead of $\sim 0.2 - 1.4\ msec$ to the JSON-LD dissecting performance.

2. It is obvious that the performance of the JSONdissect algorithm depends on the max depth of the credential. On the other hand, the performance of the URDNA2015 algorithm depends on two separate conditions. First, in the number of blank nodes in the credential. Second, the place of these blank nodes in the knowledge graph. More specifically, the most expensive operation of the URDNA2015 algorithm is the N-degrees-hash sub-algorithm. That algorithm is called when there are blank nodes in the JSON-LD graph that cannot be distinguished from each other, by using only the information that is directly linked to them. For that reason, we performed the benchmarking twice. First, using credentials with a modest max depth. Then, using credentials specifically made to have a large max depth, that do not require from the URDNA2015 algorithm to call the N-degrees-hash procedure. That way, the results of the second benchmark correspond to a best-case scenario for URDNA2015 and a worst-case scenario for Algorithm 1.

3. The total size of the resulting claims in bytes (meaning the input of the BBS+ signing algorithm) are almost the same for both algorithms, with the JSONdissect algorithm (Algorithm 1) having a slight advantage in this regard.

4.3.2 Benchmarking Results

First Benchmarking Results

For the first benchmarking procedure, we generated credentials of max depth 1, 2 and 3 with 2, 3, 8 and 16 blank nodes. The results are shown in Figure 4.2 Note that Algorithm 1 runs $6\times$ to $25\times$ times faster than the URDNA2015 algorithm.

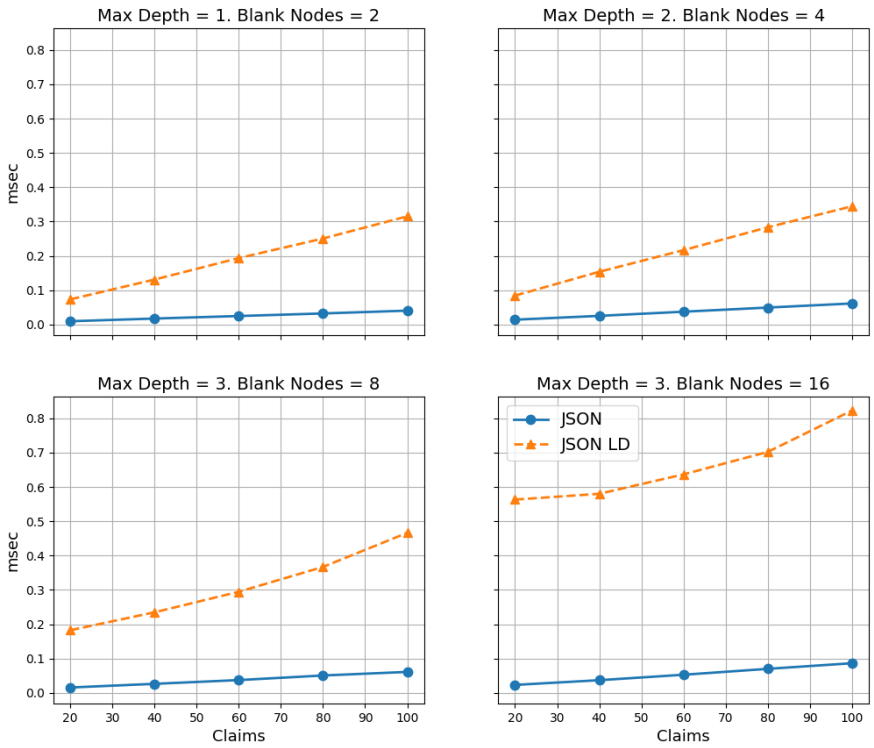

Fig. 4.2: First benchmarking results comparing the URDNA2015 for JSON LD to JSONdissect (Algorithm 1) for JSON. The credentials used have a modest depth, corresponding to more common use-cases.

Second Benchmarking

For the second benchmarking procedure, we generated credentials of max depth 2, 4, 8 and 16. Correspondingly the blank nodes are 2, 4, 8 and 16. We can observe from Figure 4.3, that even in the best-case scenario in favor of URDNA2015, JSONdissect runs $5\times$ to $9\times$ faster.

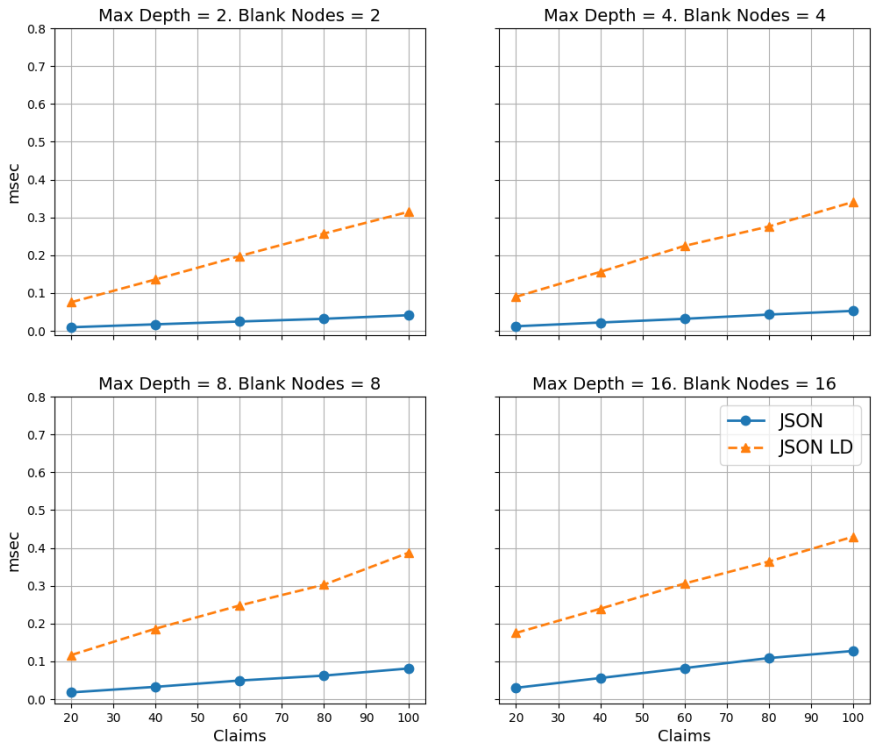

Fig. 4.3: Second benchmarking results comparing the URDNA2015 for JSON LD to JSONdissect (Algorithm 1) for JSON. The credentials used have large depth and blank nodes arranged in a way that will give the URDNA2015 algorithm the advantage and a handicap to the JSONdissect algorithm.

5 Conclusions

In this work we examined the current proposals for efficient privacy preserving management of a user's identity and their proposed trade-offs. We closely looked at the case of linked data verifiable credentials and their advantages. We also identified some vulnerabilities of the proposed specifications. Although there are already some proposed solutions for these exploits, there are worries that the resulting complexity of those algorithms may start to threaten implementability. As an answer, we proposed the use of a simpler and more limiting format (JSON instead of JSON Linked Data), for which we can create much simpler and efficient (not to mention secure) algorithms. To do that we propose a new mathematical modeling of the JSON format. We believe that, our proposed trade-off between efficiency, security and data extensibility and upgradeability is well suited for cryptographic and security/privacy applications and especially for the Verifiable Credential's use cases.

Although we discussed the basic operations of signing and creating proofs for a Verifiable Credential, most applications will need additional functionalities on top of the main ones, that we only alluded to here. Examples include mechanisms to revoke a credential, to retrieve the identity of a misbehaving user etc. As a future work, we plan to enrich our JSON VCs solution with these capabilities, in a privacy preserving manner (i.e., without introducing correlation elements etc.). We also plan to expand our current code-base and create samples and demos with which to further illustrate and evaluate the usability and efficiency of our approach.

Bibliography

[AL20] Rachel Arnold and Dave Longley. *RDF Dataset Canonicalization*. 2020. URL: https://lists.w3.org/Archives/Public/public-credentials/2021Mar/att-0220/RDFDatasetCanonicalization-2020-10-09.pdf (visited on Sept. 11, 2021).

[BBC08] David Bauer, Douglas M Blough, and David Cash. "Minimal Information Disclosure with Efficiently Verifiable Credentials". In: *Proceedings of the 4th ACM workshop on Digital Identity Management*. 2008, pp. 15–24.

[BBS04] Dan Boneh, Xavier Boyen, and Hovav Shacham. "Short Group Gignatures". In: *Annual International Cryptology Conference*. Springer. 2004, pp. 41–55.

[BHL01] Tim Berners-Lee, James Hendler, and Ora Lassila. "The Semantic Web". In: *Scientific American* 284.5 (2001), pp. 34–43.

[BLS01] Dan Boneh, Ben Lynn, and Hovav Shacham. "Short Signatures from the Weil Pairing". In: *International Conference on the Theory and Application of Cryptology and Information Security*. Springer. 2001, pp. 514–532.

[Bow17] Sean Bowe. *BLS12-381: New ZK-SNARK Elliptic Curve Construction*. 2017. URL: https://electriccoin.co/blog/new-snark-curve/ (visited on Sept. 11, 2021).

[Bra07] Stefan Brands. *The problem(s) with OpenID*. 2007. URL: https://web.archive.org/web/20110516013258/http://www.untrusted.ca/cache/openid.html (visited on Sept. 15, 2021).

[CDL16] Jan Camenisch, Manu Drijvers, and Anja Lehmann. "Anonymous Attestation Using the Strong Diffie-Hellman Assumption Revisited". In: *International Conference on Trust and Trustworthy Computing*. Springer. 2016, pp. 1–20.

[CL04] Jan Camenisch and Anna Lysyanskaya. "Signature Schemes and Anonymous Credentials from Bilinear Maps". In: *Annual International Cryptology Conference*. Springer. 2004, pp. 56–72.

[CWL14] Richard Cyganiak, David Wood, and Markus Lanthaler. *RDF 1.1 Concepts and Abstract Syntax*. 2014. URL: https://www.w3.org/TR/rdf11-concepts/ (visited on Sept. 11, 2021).

[DK18] Mahto Dindayal and Yadav Dilip Kumar. "Performance Analysis of RSA and Elliptic Curve Cryptography." In: *nternational Journal of Network Security* 20.4 (2018), pp. 625–635.

[Har20] Halpin Harry. "A Critique of Immunity Passports and W3C Decentralized Identifiers". In: *arXiv preprint arXiv:2012.00136* (2020).

[Ho+12] Au Man Ho, Willy Susilo, Yi Mu, and Chow Sherman S. M. "Constant-Size Dynamic K-Times Anonymous Authentication". In: *IEEE Systems Journal* 7.2 (2012), pp. 249–261.

[KBC97] Hugo Krawczyk, Mihir Bellare, and Ran Canetti. *HMAC: Keyed-Hashing for Message Authentication*. RFC 2104. RFC Editor, 1997.

[LS21] Tobias Looker and Orie Steele. *BBS+ Signatures 2020*. 2021. URL: https://w3c-ccg.github.io/ldp-bbs2020/ (visited on Sept. 11, 2021).

[MW21] Jeremie Miller and David Waite. *JSON Web Proof*. 2021. URL: https://hackmd.io/@quartzjer/JSON_Web_Proof (visited on Sept. 15, 2021).

[Mye+99] Michael Myers, Rich Ankney, Ambarish Malpani, Slava Galperin, and Carlisle Adams. *X.509 Internet Public key Infrastructure Online Certificate Status Protocol-OCSP*. RFC 2560. RFC Editor, 1999.

[NQ21] Barclay Neira and Caleb Queern. *Introduction to Azure Active Directory Verifiable Credentials*. 2021. URL: https://docs.microsoft.com/en-us/azure/active-directory/verifiable-credentials/decentralized-identifier-overview (visited on Sept. 15, 2021).

[Ott+19] Nate Otto, Sunny Lee, Brian Sletten, et al. *Verifiable Credentials Use Cases*. 2019. URL: https://www.w3.org/TR/vc-use-cases/ (visited on Sept. 11, 2021).

[Ped91] Torben Pryds Pedersen. "Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing". In: *Annual International Cryptology Conference*. Springer. 1991, pp. 129–140.

[PM19] Roger Jover Piqueras and Vuk Marojevic. "Security and Protocol Exploit Analysis of the 5G Specifications". In: *IEEE Access* 7 (2019), pp. 24956–24963.

[PZ13] Christian Paquin and Greg Zaverucha. *U-Prove Cryptographic Specification V1.1. Revision 3*. 2013. URL: https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/U-Prove20Cryptographic20Specification20V1.1.pdf (visited on Sept. 15, 2021).

[RSA78] Ronald Rivest, Adi Shamir, and Leonard Adleman. "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems". In: *Communications of the ACM* 21.2 (1978), pp. 120–126.

[Sak+14] Natsuhiko Sakimura, John Bradley, Mike Jones, Breno De Medeiros, and Chuck Mortimore. "Openid connect core 1.0". In: *The OpenID Foundation* (2014), S3.

[SAM15] Steve Speicher, John Arwe, and Ashok Malhotra. *Linked Data Platform 1.0*. 2015. URL: https://www.w3.org/TR/ldp/ (visited on Sept. 15, 2021).

[Sim21] Similartech.com. *OpenID. Market Share & Web usage Statistics*. 2021. URL: https://www.similartech.com/technologies/openid (visited on Sept. 15, 2021).

[SLC19a] Manu Sporny, Dave Longley, and David Chadwick. *Verifiable Credentials Data Model 1.0*. 2019. URL: https://www.w3.org/TR/vc-data-model/ (visited on Sept. 13, 2021).

[SLC19b] Manu Sporny, Dave Longley, and David Chadwick. *Verifiable Credentials Ecosystem Overview*. 2019. URL: https://www.w3.org/TR/vc-data-model/#ecosystem-overview (visited on Sept. 13, 2021).

[Spo+20] Manu Sporny, Dave Longley, Gregg Kellogg, et al. *JSON-LD 1.1*. 2020. URL: https://www.w3.org/TR/json-ld11/ (visited on Sept. 15, 2021).

[Zhi+21] Zhang Zhiyi, Król Michal, Sonnino Alberto, Zhang Lixia, and Rivière Etienne. "EL PASSO: Efficient and Lightweight Privacy-Preserving Single Sign On." In: 2. Sciendo. 2021, pp. 70–87.

List of Acronyms

PKI Public Key Infrastructure

VC Verifiable Credential

LD Linked Data

JSON JavaScript Object Notation

RSA Rivest–Shamir–Adleman Digital Signatures

POK Proof of Knowledge

ZKP Zero Knowledge Proof

List of Figures

2.1 The Verifiable Credential's Ecosystem. Figure taken from: https://www.w3.org/TR/vc-data-model/ 5
2.2 The Verifiable Credential's Data Model. Figure adapted from: https://www.w3.org/TR/vc-data-model/ 7
2.3 A set of claims as a knowledge graph. Figure adapted from: https://www.w3.org/TR/vc-data-model/ 8
2.4 An airline ticket represented as a Verifiable Credential. 9
2.5 The VC from Figure 2.4 signed by the Issuer/ Airline (green part). 9
2.6 The VC from Figure 2.5 also signed by the Holder (green part). 10

3.1 An example of a Verifiable Credential in JSON LD format. 18
3.2 The corresponding Knowledge Graph of the credential from Figure 3.1. . . . 19
3.3 JSON LD credential example and corresponding knowledge graph. 24
3.4 JSON LD credential and corresponding knowledge graph the Verifier will receive if the Holder uses selective disclosure to hide their "firstName" from their credential of Figure 3.3 24
3.5 JSON LD credential example and corresponding knowledge graph. 25
3.6 JSON LD credential example. If the Holder wants to keep the name and age of their youngest kid the adversary will be able to unveil them 27

4.1 Modeling a JSON data-structure example. 34
4.2 First benchmarking results comparing the URDNA2015 for JSON LD to JSONdissect (Algorithm 1) for JSON. The credentials used have a modest depth, corresponding to more common use-cases. 40
4.3 Second benchmarking results comparing the URDNA2015 for JSON LD to JSONdissect (Algorithm 1) for JSON. The credentials used have large depth and blank nodes arranged in a way that will give the URDNA2015 algorithm the advantage and a handicap to the JSONdissect algorithm. 41

List of Tables

3.1 Messages returned from the URDNA2015 algorithm when is used from the Holder/Issuer and the Verifier with the credential of Figure 3.3 and Figure 3.4 correspondingly 25
3.2 Reference sets Q_1 and Q_2. 26

4.1 Values of the ϕ and g functions of the Credential 4.1 35

List of Algorithms

1 Dissect a JSON credential 38

Appendix A

Pairings have many very interesting properties and uses. However before continuing we will need some additional definitions of the discreet logarithm (DL) problem, the decision Diffie-Hellman (DDH) problem and the computational Diffie-Hellman (CDH) problem, in a group G.

- **DL problem.** The DL (discreet logarithm) problem is the following. Given two elements $g, h \in G$, if it exits, find an integer n such that $h = g^n$.

- **CDH problem.** The CDH (computational Diffie-Hellman) problem is defined a following. Given three group elements $g, g^a, g^b \in G$, compute an element h of G such that $h = g^{ab}$.

- **DDH problem.** The DDH (decision Diffie-Hellman) problem is the following. Given four group elements $g, g^a, g^b, g^c \in G$, return true if $c = ab$ modulo the order of g and false otherwise.

These three problems are assumed to be hard (on various degrees and under certain conditions) and on this hardness allot of cryptosystems have based their security. Note that DDH is no harder than the CDH problem and CDH is no harder than the DL problem. Interestingly, pairings first use in cryptography was to attack cryptosystems based on the DL problem. That is because, using their bi-linearity property, pairings could reduce the DL problem from a large group G where the problem was hard, to a smaller group G_T where the DL problem was easier to solve. Another interesting fact is that if an easily computed pairing exists between a group G and another group G_T, the DDH problem becomes easy on G. Indeed, for given elements $g, g^a, g^b, g^c \in G$ and a pairing e, it holds that $e(g, g^c) = e(g, g)^c$ and $e(g^a, g^b) = e(g, g)^{ab}$. As a result to check if $c = ab$, one can simply check if $e(g, g^c) = e(g^a, g^b)$, solving that way the DDH problem. Because of those two reasons, initially pairings were treated as something to avoid, since they compromised the security of allot of cryptosystems that were based on the DL and DDH problems.

Since then however, pairings have been shown to be very useful in creating a variety of different cryptographic protocols. Those systems typically work on a specific type of finite groups called gap groups, where it exists an easily computed pairing meaning that the DDH problem is easy while the CDH and DL problems are equivalent and presumably hard. Usually, pairing based cryptosystems take advantage of the ease of solving the DDH problem, by employing it as part of the decryption or key/signature verification process. For example, given $g, g^a, g^b, g^c \in G$, where a, b and c some secrets, those systems will use the fact that it easy to decide if $c = ab$ or not as part of the verification process, and the fact that it is hard to actually calculate c, a or b (since CDH and DL are hard) to guarantee security.

BBS+ signatures work similarly, but instead of being based on the CDH or DL problem they use the more general and stronger q-Strong Diffie-Helman assumption. That assumption states that given two finite groups G_1 and G_2 of prime cardinality p, and elements $g_0 \in G_1$ and $h_0 \in G_2$, it is hard to find an algorithm that on input $g_0, h_0, h_0^x, h_0^{x^2}, ..., h_0^{x^q}$ produces an output (A, x) such that $A^{\frac{1}{x+c}} = g_0$ for some $c \in Z_p$. BBS+ signatures base their security on the truth of that assumption.

Appendix B

Groups of a very large order will have strong security properties (since exhaustive search in them will be hard) but will also have large elements, or time consuming operations, which will result on poor performance. Up until now, the finite groups to provide the best balance between performance and security are elliptic curves over finite fields. Elliptic curves over finite fields ($\mathbb{E}(\mathbb{F}_p)$) are collection of points (x, y) with $y = x^3 + ax + b$, where x takes values in the finite group $\mathbb{F}_p$. In general, the advantage of elliptic curves is that there are strong evidence that the DL problem in them is hard for relatively small p (?the best known attacks will take exponential time?). On the other hand, most operations take place in $\mathbb{F}_p$ where (if p is relatively small) they are really efficient.

In the case of BBS+ however we have the additional requirement that there needs to be an efficiently computed pairing between the groups we choose. Pairing-friendly elliptic curves that continue to have the desired security and efficiency properties are relatively rare and must be specifically constructed. Until today, perhaps the best pair of elliptic curves to satisfy all of these requirements are the BLS12-381 curves. Those curves are defined over $\mathbb{F}_q$ and $\mathbb{F}_{q^2}$, where q is an 381 bits prime, and have the following equations,

- $E(\mathbb{F}_q)$ with $E : y^2 = x^3 + 4$
- $E(\mathbb{F}_{q^2})$ with $E : y^2 = x^3 + 4(1 + i)$

The reasons these two specific pairings friendly elliptic curves where chosen are out of scope of this work, since they include various technical trade-offs between security and efficiency. Finally, the groups chosen are $\mathbb{G}_1 \subset E(\mathbb{F}_q)$, $\mathbb{G}_2 \subset E(\mathbb{F}_{q^2})$ and $\mathbb{G}_T \subset \mathbb{F}_{q^{12}}$.