Requirements and Secure Serialization for Selective Disclosure Verifiable Credentials

Vasilis Kalos[1] and George C. Polyzos[1]

Mobile Multimedia Laboratory
Department of Informatics, School of Information Sciences and Technology
Athens University of Economics and Business, 104 34 Athens, Greece
{kalos20, polyzos}@aueb.gr

Abstract. The emergence of the Verifiable Credentials recommendation from W3C allows the adoption of credential systems in a much wider range of user-centric applications and use cases. With this shift to user-centric credential systems, Selective Disclosure has been proposed and used to cryptographically secure user privacy. Although much work has been undertaken in creating selective disclosure supporting cryptographic protocols, those schemas are not directly applicable for credentials. Implementations rely on canonicalization algorithms to transform a credential to the necessary data format, which will be used by the cryptographic layer. Those algorithms are often used without the necessary cryptographic and security considerations, leading to insecure implementations. In this work we define three necessary security properties for the canonicalization algorithms. We also propose a mathematical model for JSON credentials, which we use to prove the security of a proposed canonicalization algorithm.

Keywords: Self-Sovereign Identity (SSI) · privacy · JSON · canonicalization · linked data · JSON LD · Anonymous Credentials

1 Introduction

Credentials systems play a key role in the management of user identities on the Internet, with systems like Open ID Connect (OIDC) [16] widely used to this day.[1] In the credentials ecosystem, there are three main entities: the Issuer that creates and signs the credential (e.g., an airline company that issues a ticket), the Holder that controls the credential (e.g., a traveller that bought the ticket) and the Verifier that checks the validity of that credential (e.g., the airport agent that validates the ticket). A limitation of existing credentials systems, like OIDC, is that the issuing and verification procedures are tightly coupled, which allows for the tracking of the user by the identity provider (Idp) [4]. Furthermore, the Idp must always be online for the process to be successful.

[1] OpenID. Market Share & Web Usage Statistics: https://www.similartech.com/technologies/openid

As an answer to the above limitations, anonymous credentials [13, 15, 19, 5] have been proposed, which decouple the "identity issuance" procedure from the verification process, with W3C's Verifiable Credentials (VCs) standard [17] being the first step towards a standardized data model. In general, a VC is a data structure, usually either in JSON or JSON Linked Data (JSON-LD) [18] format, containing various metadata and claims as key-value pairs. The W3C specification allows, and many of the envisioned use cases [14, 12] expect, the creation and usage of long-lived credentials, which in many cases would contain the user's personal information (e.g., age and address, medical records, security information like passwords, bank account numbers etc.).

An important problem that rises as a result, concerns the disclosure of the user's private data. If the included cryptographic proofs are created with "traditional" digital signatures (like RSA etc.), the entire credential must be presented to the Verifier for the proof 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 VCs at best or be quite dangerous at worst. Furthermore, not all the information in a VC will be always needed by all the potential Verifiers. From these, and many other use cases[2], emerges the need to hide personal or sensitive information from the VC and still be able to convince a Verifier regarding the ownership, correctness, and integrity of the revealed information.

To meet those requirements, the use of cryptographic protocols supporting "selective disclosure" has been proposed [3, 15, 6, 1]. Selective disclosure protocols work by signing a list of messages and giving the Holder the ability to choose what messages they want to reveal in each interaction with a Verifier. In this work, we will consider selective disclosure as being comprised of two different properties. Firstly, the "selective showing" part, which allows the Holder to prove the integrity, ownership, and authenticity of any subset from a set of signed messages, and secondly, the "zero-knowledge" part [9], that protects against any information about the un-disclosed messages being leaked. As we mentioned though, in practice a credential will be in some structured data format (e.g., JSON, JSON-LD etc.). Turning these formats to a list of messages, to be signed by a selective disclosure supporting cryptographic protocol, is not trivial. In consequence, many implementations and standards will use over-complicated or insecure "data canonicalization" algorithms, or make some over-simplifications [11]. Most importantly however, those algorithms might be overlooked during security analysis as a potential point of exploit, with some being used without the necessary security audits.

In this work we draw attention to the need for rigorous security assessments of the canonicalization algorithms when used to enable the selective disclosure of a Verifiable Credential. To achieve that, we first present a threat model for those algorithms. Then, we introduce a novel modelling of a JSON credential that we use to give an example of a canonicalization algorithm and demonstrate its security in the context of the presented threat model. Finally, we evaluate the performance of our canonicalization algorithm.

[2] Further cases: https://identity.foundation/bbs-signature/draft-bbs-signatures.html

2 Related Work

There is a long line of work around anonymous credentials [10, 3, 15, 19, 6], first envisioned by Chaum [8]. Different proposals introduce different properties for a wide range of use-cases. For example, proposals like Microsoft's U-Prove [15] offer the ability to efficiently revoke a credential at any time, but to do so, inserts correlatable elements to the credential. On the other hand, El-Passo [19] offers two factor authentication and correlation protection, but does not support credential revocation (it can optionally support anonymity revocation, though in this case, the user will be correlatable by the same Verifier). Those systems apply various cryptographic protocols ([5, 9, 7]) to achieve anonymity and user privacy. Those cryptographic protocols offer a vast range of features like range proofs, delegating signatures etc. Almost all the anonymous credentials proposals though use selective disclosure to protect the user's privacy. For this reason, in our work we are mainly focusing on the selective showing and zero-knowledge (of the undisclosed credential) properties, although our results are generic enough that may also apply to other properties as well (like range proofs etc.).

For canonicalization algorithms, perhaps the most notable example is the URDNA algorithm [2], mainly used in the case of JSON-LD credentials (but could also be applied to a JSON format). URDNA works by modelling a credential as a graph. Then it creates labels for each node and edge of that graph. It then returns the edges of the knowledge graph as the messages corresponding to the credential. That said, URDNA does suffer from the third vulnerability defined in our threat model (Definition 6, in Section 3), i.e., it compromises the zero-knowledge of the underlying cryptographic protocol's selective disclosure property [11]. Note that we have presented that result to the relevant W3C working group and an efficient solution has already been derived. Other canonicalization algorithms like JSON Web Proofs[3] and Termwise Canonicalization[4] (of which the first version also suffered from the second vulnerability of our threat model but has been patched since then) have been proposed and are currently developed.

3 Background and Definitions

3.1 Credentials

For simplicity, in this work we define a credential to be any JSON or JSON-LD data structure (unless otherwise stated), containing attributes and metadata as key-value pairs (for example credentials that are compliant with W3C's Verifiable Credential Data Model specification). Let $\mathcal{CR}[K, V]$ be the space of all credentials with keys from K and values from V. Let also $C \xleftarrow{R} \mathcal{CR}[K, V]$ denote a randomly sampled credential from the $\mathcal{CR}[K, V]$ space.

[3] https://github.com/json-web-proofs/json-web-proofs
[4] https://github.com/yamdan/jsonld-signatures-bbs

Before continuing, we will need to define the equality between two credentials. We will also need to define when a credential C_1 is a "sub-credential" to another credential C_2, meaning that every key and value of C_1 is also present in C_2 and with the same structure. The two definitions follow,

Definition 1 (Sub-Credentials). *Let S and C be two credentials. S is a sub-credential of C iff Algorithm 1 on input (S, C) returns true. In that case we write that $S \lhd C$. We also define $\mathcal{UP}(S) = \{C \in \mathcal{CR}[K,V] : S \lhd C\}$ and $\mathcal{SU}(C) = \{S \in \mathcal{CR}[K,V] : S \lhd C\}$.*

Definition 2 (Credentials Equality). *Let C_1 and C_2 be two credentials. If both C_1 and C_2 are in JSON-LD format, then $C_1 = C_2$ iff $URDNA(C_1) = URDNA(C_2)$. If the credentials are in JSON format, then $C_1 = C_2$ iff $C_1 \lhd C_2$ and $C_2 \lhd C_1$.*

Algorithm 1 Recursive algorithm that checks if a credential is contained by another credential

```
procedure SUB-CREDENTIAL(C1, C2)
    for key ∈ C1 do
        if key ∉ C2 then
            return false
        else if C1(key) is a JSON object then
            SUB-CREDENTIAL(C1(key), C2(key))
        else
            if C1(key) ≠ C2(key) then
                return false
    return true
```

3.2 Selective Disclosure and Canonicalization Algorithms

As a canonicalization algorithm we define any algorithm that on input of a credential, returns a list of messages (or claims, or attributes as they have been often called in research). Note that we don't restrict the nature of those messages and don't request anything about their semantic meaning, though most applications will use canonicalization algorithms that map to messages conveying similar information with the credential. More formally, we give the following definition.

Definition 3 (Canonicalization Algorithm). *Let M be the space of messages and $\mathbb{P}(M)$ the space of all the finite sets with elements from M. Given a credential $C \in CR[K,V]$ we define the canonicalization algorithm, indexed by C as,*

$$Can_C : CR[K,V] \to \mathbb{P}(M)$$

We also define $\mathcal{O}^{Can}(\cdot,\cdot)$ *to be an Oracle that on query* (C, S) *will answer with,*

$$\mathcal{O}^{Can}(C,S) = \begin{cases} Can_C(S) \; if \; S \lhd C \\ null \qquad otherwise \end{cases}$$

We define canonicalization algorithms to be indexed by a credential for a multiple of reasons. Consider the case where an Issuer canonicalizes a credential C to create a signature, which it sends, along with C, to a Holder. Let the Holder choose a sub credential S of C to present to the Verifier, along with a proof of knowledge of the signature (which they create by also canonicalizing S). Depending on the credential that was originally signed (i.e., C), the canonicalization's algorithms result on S may need to differ, for the proof generation and verification process to succeed. That said, the Holder cannot just send C to the Verifier, resulting to the need to send $Can_C(\cdot)$ instead. We stress that this is just a convention to simplify notation. In practice instead of $Can_C(\cdot)$, the Holder will send the necessary information (that is still depended on the original credential, i.e., C), encoded in an appropriate format. The Verifier will use that information to canonicalize the presented credential correctly. For an example of this procedure, see Section 4.3.

The general structure of the cryptographic protocols, for which the canonicalization algorithms are intended can be seen in Figure 1. We stress that the Verifier will not get a list of messages from the Holder but a credential (a sub credential of the original, that contains only the information the Holder wants to disclose). The reason being, that the Verifier will be interested in the semantic and structural information that this credential provides, which may not translate exactly to the messages corresponding to the canonicalized result (aside from the fact that this is the way that has been standardized by W3C). That said, the cryptographic layer awaits a list of messages. It is this mismatch that enables the possible exploits, defined in the next section.

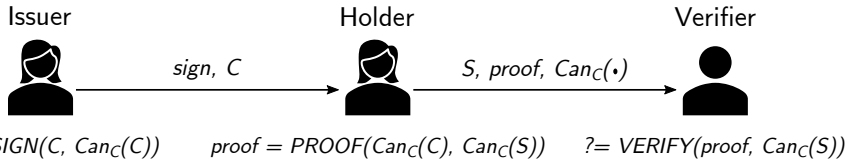

Fig. 1. The general structure of a credentials system using selective disclosure. The $SIGN$, $PROOF$ and $VERIFY$ functions are generalisations of the signing, proof generation and verification functions, of the selective disclosure cryptographic protocols.

3.3 Threat Model

In this section we formally define the considered threat model. Usual security properties (e.g., unforgeability of a signature, security of a private key etc.) are

not considered, as we assume that the underlying cryptographic protocols are secure. We are mainly considering threats that could lead to the malicious misuse of the canonicalization algorithm by the Holder or the Verifier.

To that end, we define two properties (Definitions 4 and 5) that if not met, could allow the Holder to find a different credential (or sub-credential for when they don't disclose part of their VC) than the one created and signed by the Issuer, that after canonicalized, results to the same messages (or subset of those messages) with the ones that the Issuer signed, effectively allowing the Holder to cheat the Verifier. The definitions follow,

Definition 4 (Collision resistant). *For credentials C and canonicalization algorithm Can_C, $\nexists$ different credentials S_1 and S_2, so that $Can_C(S_1) = Can_C(S_2)$.*

Definition 5 (Forgery resistant). *For credential C and canonicalization algorithm Can_C $\nexists$ subset $m \subset Can_C(C)$ so that there is credential S' with $S' \not\lhd C$ and $Can_C(S) = m$*

Note that we don't consider the case where $\exists\, C_1, C_2, S$ with $C_1 \neq C_2$, $S \lhd C_i$, $i = 1, 2$ and $Can_{C_1}(S) = Can_{C_2}(S)$. Although the Holder could technically "cheat" by revealing $Can_{C_2}(\cdot)$ instead of $Can_{C_1}(\cdot)$, they don't actually reveal any information that is not signed by the Issuer (they only reveal S).

To protect the zero-knowledge property of the underlying selective disclosure capable system, we need the output of $Can_C(S)$ to not reveal information about the not disclosed part of C to the Verifier. Following we formalize that requirement in two different levels.

Definition 6 (Hiding). *Let adversary $\mathcal{A}^{\mathcal{O}^{Can}}$ be a probabilistic polynomial time (PPT) running algorithm $\mathcal{A}$ with access to the $\mathcal{O}^{Can}(\cdot\,,\cdot)$ oracle.*

- *we will denote the canonicalization algorithm as being "total hiding" if the advantage $Adv(S)$ of $\mathcal{A}^{\mathcal{O}^{Can}}$ for credential S defined as,*

$$Adv(S) = \Big| Pr(\mathcal{A}^{\mathcal{O}^{Can}}(C, S, Can_C(S)) = 1 : C \in \mathcal{UP}(\mathcal{S})) - Pr\Bigg(\mathcal{A}^{\mathcal{O}^{Can}}(C', S, Can_C(S)) = 1 : \begin{matrix} C \in \mathcal{UP}(S), C' \neq C \\ C' \xleftarrow{R} \mathcal{UP}(S), \end{matrix}\Bigg)\Big|,$$

  is negligibly above semantic interpretation $\forall\, S \in \mathcal{CR}[K, V]$.
- *we will denote the canonicalization algorithm as being "values hiding" if the advantage $Adv(S)$ of $\mathcal{A}^{\mathcal{O}^{Can}}$ for credential S defined as,*

$$Adv(S) = \Big| Pr(\mathcal{A}^{\mathcal{O}^{Can}}(C, S, Can_C(S)) = 1 : C \in \mathcal{UP}(\mathcal{S})) - Pr\Bigg(\mathcal{A}^{\mathcal{O}^{Can}}(C', S, Can_C(S)) = 1 : \begin{matrix} C \in \mathcal{UP}(S), C' \neq C \\ C' \xleftarrow{R_{val}} C/S, \end{matrix}\Bigg)\Big|, \tag{1}$$

  is negligible above semantic interpretation $\forall\, S \in \mathcal{CR}[K, V]$.

All probabilities are defined over the coin flips of Can_C and $\mathcal{A}^{\mathcal{O}^{Can}}$ as well as the choices of C and C'.

By $C' \xleftarrow{R_{val}} C/S$ we define a credential C' that is the same as C with all the values that don't also belong to the sub-credential S of C being random. By "negligible above semantic interpretation" we define a PPT algorithm $\mathcal{A}$ that has no more advantage in distinguishing the two distributions, than it would have by just semantically examining the credentials (note that all our proofs are in the standard model). Formally, $\mathcal{A}$ will have "negligible above semantic interpretation" advantage if given $Can_C(S)$ and access to the oracle $\mathcal{O}^{Can}(\cdot\,,\cdot)$, it results to a negligible rise in the advantage it would have, if it was only given the two credentials (i.e., (C, S) and (C', S)). Intuitively, consider the following example. Let the Holder having a credential $C = \{age : 25, profession : JuniorDev\}$, presenting the sub-credential $S = \{profession : JuniorDev\}$ ($S \triangleleft C$). An adversary $\mathcal{A}$ that sees S, after semantically examining the credentials, may be able to deduce that the Holder is more likely to possess C rather, for example, the credential $C' = \{age : 90, profession : JuniorDev\}$.

As a result, the "total hiding" property defines that given credential $S \triangleleft C$, the output of the canonicalization algorithm $Can_C(S)$ will not give any additional advantage, to an adversary trying to extract information about the rest of the credential that is not part of S (i.e., C). On the other hand, the "values hiding" property defines that the adversary will not gain any additional advantage, when trying to extract any information regarding the values of C not in S (but they may get some information on the keys or the structure of C, that does not appear in S).

The reason we define a stronger ("total hiding") and a weaker ("values hiding") version of the same property, is that for many implementations the weaker level of security will be enough, while leading to simpler and more efficient algorithms. Furthermore, in many applications, information regarding the structure of the credential will be either publicly available, as to enable certain protocols (e.g., Holder, Verifier VC negotiations etc.) or will be leaked through some other way (perhaps through some requirement of the cryptographic protocol). Those applications could use faster algorithms, by opting for the weaker hiding property.

We can easily show now that given the output of a "values hiding" canonicalization algorithm $Can_C(S)$, C cannot be retrieved and that different outputs of Can_C cannot be linked together. Obviously, the same result holds for "total hiding" algorithms.

Theorem 1. *Let a "values hiding" canonicalization algorithm Can.*

- *there is no PPT algorithm $\mathcal{A}_1$ that given $Can_C(S)$ will return C with non-negligible probability.*
- *there is no PPT algorithm $\mathcal{A}_2$ that given, $Can_{C_i}(S_i)$ for $C_i \in \mathcal{CR}[K, V]$ and $S_i \triangleleft C_i$, $i = 1, 2$, can decide with non-negligible advantage if $C_1 = C_2$.*

Proof. For the first property, let $\mathcal{A}$ be a PPT that on input $(C', Can_C(S))$, for $S \in \mathcal{CR}[K, V]$ and $C, C' \in \mathcal{UP}(S)$ returns 1 if $\mathcal{A}_1(Can_C(S)) = C'$ and 0

otherwise. Obviously $\mathcal{A}$, will be able to distinguish between the distributions $\{(C, Can_C(S)) : C \in \mathcal{UP}(S)\}$ and $\{(C', Can_C(S)) : C', C \in \mathcal{UP}(S), C \neq C'\}$ with non-negligible advantage, breaking the "values hiding" property.

For the second property we will take advantage that we allow $\mathcal{A}$ in Definition 6 to call the $\mathcal{O}^{Can}(\cdot, \cdot)$ oracle. As such, we define a PPT algorithm $\mathcal{A}^{\mathcal{O}^{Can}}$ that on input $(C_2, Can_{C_1}(S_1))$ query's the oracle to get $z = \mathcal{O}^{Can}(C_2, S_2)$. It holds, $S_2 \lhd C_2 \Rightarrow z \neq null$. Let $\mathcal{A}^{\mathcal{O}^{Can}}(C_2, Can_{C_1}(S_1))$ return J with $J = \mathcal{A}_2(Can_{C_1}(S_1), z) = \mathcal{A}_2(Can_{C_1}(S_1), Can_{C_2}(S_2))$. If $C_1 = C_2$, then $J = 1$ with non-negligible advantage (equivalently $J = 0$ if $C_1 \neq C_2$). It is trivial to see that $\mathcal{A}^{\mathcal{O}^{Can}}$ breaks the "values hiding" property.

We also define an canonicalization algorithm to be pregnable against "vulnerabilities 1, 2" and "3" if it is not "collision resistant", "forgery resistant" or "hiding" correspondingly.

4 Proposed Algorithm and Security Analysis

4.1 JSON modelling

When considering JSON-LD or JSON credentials, the modelling was mainly done using graphs and more specifically, the knowledge graph representing the claims of the VC. That modelling, although flexible, requires complicated, not intuitive canonicalization algorithms. As a novelty of our paper, we propose an alternative modelling, that will represent the credential in a way closer to the one required by the cryptographic algorithms, i.e., as a set of bit-arrays. To do that, we will use finite functions, that can be naturally (i.e., by definition) transformed into a set. We will then demonstrate how to use our modelling to prove the security properties of a proposed canonicalization algorithm. Our hope is that this representation of a credential, will make it easier for other algorithms to be proven secure, and make the security analysis of credential systems supporting selective disclosure more formal.

The basic observation is 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. We first define a non-zero positive integer $n \in \mathbb{N}$. Let $[n] = \{1, 2, ..., n\}$.

Structure:
Let K be the set of all possible "keys" that can appear in the JSON and K^* be the set of finite tuples with elements from K. We define the **structure** of a JSON to be K along with an injective function ϕ,

$$\phi : [n] \to K^* \tag{2}$$

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, along with a function g,

$$g : [n] \to V \tag{3}$$

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 7. *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 structure as $J = (n, \phi, g)$.*

The intuition behind the proposed modelling 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 the corresponding actual literal value.

As an example of our modelling, consider the credential of Figure 2, with each place that a literal value can go mapped to an integer in $[6] = \{1, 2, ..., 6\}$.

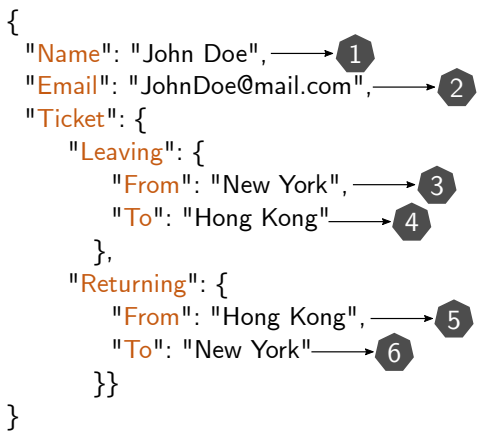

Fig. 2. modelling a JSON data-structure example.

The values of the ϕ and g functions can be seen at Table 1. Note that $\forall\, i_0, i_1 \in [6]$, with $i_0 \neq i_1 \Rightarrow \phi(i_0) \neq \phi(i_1)$, while $g(3) = g(6)$ and $g(4) = g(5)$.

Table 1. Values of the ϕ and g functions of the Credential on Figure 2.

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

From the above example it can be seen that the order with which we map each literal value position in the JSON to an integer, should not matter, hence the following definition of JSON equality,

Definition 8. *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 $J_1 = J_2$ iff $n_1 = n_2 = n$ and there is a permutation σ of $[n]$, such that $\phi_1 = \phi_2(\sigma)$ and $g_1 = g_2(\sigma)$.*

Similarly, we get the following definition for when a JSON credential is a sub-credential to another credential,

Definition 9. *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 $J_1 \lhd J_2$ iff $n_1 \leq n_2$ and there is a permutation σ of $[n_2]$, such that $\forall i \in [n_1]$, $\phi_1(i) = \phi_2(\sigma(i))$ and $g_1(i) = g_2(\sigma(i))$*

It is easy to see that Definitions 8 and 9 are consistent with Definitions 2 and 1 respectively. As an example, if C_1 and C_2 credentials with $C_1 \not\lhd C_2$, then from Definition 1, there must be either a value in C_1 that does not appear in C_2 or a key in C_1 (or a nested object of C_1) that does not appear in C_2 (or a nested object of C_2). If we represent those credentials using our modeling, i.e., as $C_1 = (n_1, \phi_1, g_1)$ and $C_2 = (n_2, \phi_2, g_2)$ where ϕ_1, ϕ_2, g_1, g_2 as in Table 1 for the example of Figure 2, we can conclude that, for some $i \in [n_1]$ either $\phi_1(i) \notin [\phi_2(1), ..., \phi_2(n_2)]$ or $g_1(i) \notin [g_2(1), ..., g_2(n_2)]$ which means that $C_1 \not\lhd C_2$ per Definition 9 as well.

An important part in Definition 7 of JSON structures is that the function ϕ is injective. However, if the credential contains a list, that may not be the case, given the definitions above. For example, consider the credential $C = \{\text{“key”} : [\text{“v}_1\text{”}, \text{“v}_2\text{”}]\}$. Using the same method as in the example of Figure 2, we will get $\phi(1) = \phi(2) =$ “key”. To eliminate that problem, we consider a subset of $\mathcal{CR}[K, V]$ which we will call “simple credentials” and that will not contain any lists. We will denote the set of those credentials as $\mathcal{SCR}[K, V]$. We stress that our definition is not as restrictive as it may seem, since there is a simple mapping between any credential from $\mathcal{CR}[K, V]$ to a credential in $\mathcal{SCR}[K, V]$, using the following transformation; $\psi([a_1, a_2, ..., a_L]) = \{I_1 : a_1, I_2 : a_2, ..., I_L : a_L\}$ where $I_i, i \in \mathbb{N}$ is an index reserved for this use (for example “_lid#i”). By applying the transformation ψ to all the lists of a credential (i.e., transforming a list to a mapping between the index of the list element and that element) in $\mathcal{CR}[K, V]$ we get a credential in $\mathcal{SCR}[K, V]$. This will allow us to define the proposed canonicalization algorithm (see Section 4.2) for any credential in $\mathcal{CR}[K, V]$.

Let credential $C \in \mathcal{CR}[K, V]$ and $C' \in \mathcal{SCR}[K, V]$ with C' be the credential C after ψ is applied to all its lists. We define a mapping $\psi_C : \mathcal{SC}[C] \rightarrow \mathcal{SC}(C')$ such that, on input $S \lhd C$, $\psi_C(S)$ will be the credential S after all lists of S are transformed the same way the lists of C did (note that since $S \lhd C$, for every list L of S, there will be a list L' of C with $L \subseteq L'$). As an example,

$$C = \{\text{“key”} : [\text{“v}_1\text{”}, \text{“v}_2\text{”}]\} \rightarrow \psi_C(C) = \{\text{“key”} : \{I_1 : \text{“v}_1\text{”}, I_2 : \text{“v}_2\text{”}\}\}$$
$$S = \{\text{“key”} : [\text{“v}_2\text{”}]\} \rightarrow \psi_C(S) = \{\text{“key”} : \{I_2 : \text{“v}_2\text{”}\}\}$$

It is trivial to show that $\psi_{C_1}(S_1) = \psi_{C_2}(S_2) \Rightarrow S_1 = S_2$ and that if $S_1 \lhd S_2 \Leftrightarrow \psi_C(S_1) \lhd \psi_C(S_2)$ (note that the transformation ψ is injective). A drawback of our modelling is that, following Definitions 7 and 8, if the JSON contains a list and we change the order of the elements in that list we will get a different JSON (per the Definition 8 of equality). Although that caveat does not seem to have any significance in practice, especially for cryptographic applications where the signed data should not be able to change in any way, additional work could be done to extend the above modelling to also account for that case (defining broader classes of equality etc.). For the intended applications however, those definitions will suffice.

4.2 Canonicalization Algorithm

Let credential $C = (n, \phi, g) \in \mathcal{CR}[K, V]$. The proposed canonicalization algorithm $JCan_C$ (Algorithm 2) on input S, transforms S to a simple credential $S_C = (n_S, \phi_{S_C}, g_{S_C})$ using the mapping ψ_C and returns $\{\phi_{S_C}(i), g_{S_C}(i)\}_{i \in [n_S]}$.

Algorithm 2 Canonicalize a JSON credential

```
function JCan_C(S)
    S_C ← ψ_C(S)
    Messages ← []
    Claim ← None
    procedure RECURSE(S_C)
        for key ∈ S_C do
            if typeof S_C[key] = JSONobject then
                Claim ← Claim + key + "."
                RECURSE(S_C[key])
            else
                Claim ← Claim + " : " + S_C[key]
                Messages.push(Claim)
                Claim ← None
    return Messages
```

The following 2 Lemmas will be used for the security proofs of the $JCan$ algorithm. Lemma 1 provides a natural way to check equality between credentials using our Definition 7.

Lemma 1. *Let* $J_1 = (n_1, \phi_1, g_1)$ *and* $J_2 = (n_2, \phi_2, g_2)$. *Then* $J_1 = J_2$ *iff* $n_1 = n_2 = n$ *and* $\{(\phi_1(i), g_1(i))\}_{i \in [n]} = \{(\phi_2(i), g_2(i))\}_{i \in [n]}$.

Proof. We will first prove the ($\Rightarrow$) direction. Lets assume that $n_1 = n_2 = n$ and that $\{(\phi_1(i), g_1(i))\}_{i \in [n]} = \{(\phi_2(i), g_2(i))\}_{i \in [n]}$. Then $\forall\ i \in [n],\ \exists\ i' \in [n]$ so that $(\phi_1(i), g_1(i)) = (\phi_2(i'), g_2(i'))$. We define σ as,

$$\sigma : [n] \rightarrow [n]$$
$$\sigma(i) = i' \Leftrightarrow (\phi_1(i), g_1(i)) = (\phi_2(i'), g_2(i'))$$

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 $i_0 = i_1 \Rightarrow \phi_1(i_0) = \phi_1(i_1) \Rightarrow \phi_2(i'_0) = \phi_2(i'_1) \Rightarrow i'_0 = i'_1$ since ϕ_2 injective. As a result σ is a function.

2) Let now $i_0 \neq i_1$. Lets assume $\sigma(i_0) = \sigma(i_1) \Rightarrow i'_0 = i'_1$ meaning that $\phi_2(i'_0) = \phi_2(i'_1) \Rightarrow \phi_1(i_0) = \phi_1(i_1) \Rightarrow i_0 = i_1$ 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 $\phi_1(i') = \phi_2(i) \Rightarrow \sigma(i') = i$. 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 ($\Leftarrow$) 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.

Lemma 2. *Given $J = (n, \phi, g)$ and $S \subset \{\phi(i), g(i)\}_{i\in[n]}$, $\nexists\ J' = (n', \phi', g')$, with $J' \not\lhd J$ and $\{\phi'(i), g'(i)\}_{i\in[n']} = S$*

Proof. Let $J' = (n', \phi', g')$, where ϕ', g' as in the example of Figure 2, with $J' \not\lhd J$ and $\{\phi'(i), g'(i)\}_{i\in[n']} = S$. This is with no loss of generality since, given Definition 8 of equality between two JSON in our modeling, if $J' = (n, \phi, g)$ then $n' = n$ and ϕ will just be a permutation of ϕ' (similarly g will be a permutation of g'). From Definition 9 we have that there must be $j \in [n']$ so that $\phi'(j) \notin \{\phi(i)\}_{i\in[n]}$ and $g(j) \notin \{g(i)\}_{i\in[n]}$. As a result, $S = \{\phi'(i), g'(i)\}_{i\in[n']} \not\subset \{\phi(i), g(i)\}_{i\in[n]}$ which is a contradiction.

Following we prove that *JCan* is secure against Vulnerability 1 and 2 of our threat model, i.e., that it is "coalition" and "forgery resistant".

Lemma 3. *Let credentials $C_1, C_2 \in \mathcal{CR}[K, V]$, with $C_1 \neq C_2$.*

- *$\nexists$ credentials $S_i \lhd C_1, i = 1, 2$ with $S_1 \neq S_2$ and $JCan_{C_1}(S_1) = JCan_{C_1}(S_2)$.*
- *$\nexists$ credentials $S_1 \lhd C_1$ and $S_2 \lhd C_2$ with $S_2 \not\lhd C_1$ such that, $JCan_{C_2}(S_2) \subset JCan_{C_1}(S_1)$*

Proof. For the first property. Let $\psi_{C_1}(S_i) = S'_i = (n_{S_i}, \phi_{S'_i}, g_{S'_i}), i = 1, 2$. Since, $JCan_{C_1}(S_1) = JCan_{C_1}(S_2) \Rightarrow \{\phi_{S'_1}(i), g_{S'_1}(i)\}_{i\in[n_{S_1}]} = \{\phi_{S'_2}(i), g_{S'_2}(i)\}_{i\in[n_{S_2}]}$. From Lemma 1 we got that $\psi_{C_1}(S_1) = \psi_{C_1}(S_2)$ and from the construction of ψ_{C_1} that $S_1 = S_2$.

For the second property, let again $\psi_{C_i}(S_i) = S'_i = (n_{S_i}, \phi_{S'_i}, g_{S'_i}),\ i = 1, 2$. Let $m \subset JCan_{C_1}(S_1) = \{\phi_{S'_1}(i), g_{S'_1}\}_{i\in n_{S_1}}$ and $m = JCan_{C_2}(S_2)$. From Lemma 2, we get that $\psi_{C_2}(S_2) \lhd \psi_{C_1}(S_1) \Rightarrow S_2 \lhd S_1 \lhd C_1$ which is a contradiction.

For "hiding" (Definition 6), we cannot prove *JCan* to be "total hiding", but we can prove it to be "values hiding". Intuitively, that is because we are considering the indexes of a list in a credential as keys which can give away some information

of the credential's structure. For example, "*employees._lid#2: Jane*" reveals the information that likely there is a hidden message "*employees._lid#1: X*". The above problem has an easy solution, which is to transform every list in the credential to a mapping between a random key and the elements of the list (i.e., instead of $\_lid\#i$ use $\_lid\#r_i$ with r_i random). However, we deemed the gained simplicity of the algorithm to be warranted.

Lemma 4. *There is no PPT algorithm $\mathcal{A}^{\mathcal{O}^{JCan}}$ that for credential C, and a sub credential $S \lhd C$ on input $(C, S, JCan_C(S))$ will return 1 with non negligible advantage over semantic interpretation. The advantage of $\mathcal{A}^{\mathcal{O}^{JCan}}$ is defined in Equation 1.*

Proof. From the definition of $\mathcal{A}^{\mathcal{O}^{JCan}}$'s advantage for S of Equation 1, we can see that essentially is the advantage that $\mathcal{A}^{\mathcal{O}^{JCan}}$ has in distinguishing the distributions

$$\left\{(C, S, JCan_C(S)) : C \in \mathcal{UP}(S)\right\} \text{ and } \left\{(C', S, JCan_C(S)) : \begin{array}{l} C, C' \in \mathcal{UP}(S) \\ C \neq C' \\ C' \xleftarrow{R_{val}} C/S \end{array}\right\}$$

Note also that we give $\mathcal{A}^{\mathcal{O}^{JCan}}$ access to the $\mathcal{O}^{JCan}(\cdot, \cdot)$ oracle. Next we will construct a credential $C' \xleftarrow{R_{val}} C/S$ and prove that $\mathcal{O}^{JCan}(C, S) = \mathcal{O}^{JCan}(C', S)$. Let $C = (n, \phi, g)$ and $S \subset C$ with $S = (n_S, \phi_S, g_S)$. We create a random credential $C' = (n, \phi_{C'}, g_{C'})$ with $\phi_{C'} = \phi_C$ and,

$$g_{C'}(i) = \begin{cases} g_S(i) & i \in [n_S] \\ r_i & i \in [n] \setminus [n_S], r_i \text{ random value} \end{cases}$$

It is trivial to show that $S \lhd C'$ and that $\psi_C(S) = \psi_{C'}(S)$. For the last equality note that if $[a_1, a_2, ..., a_L]$ and $[b_1, b_2, ..., b_{L'}]$ are lists of C and C' correspondingly, if there is some indexes I_S for which the elements of those lists also appear in a list of S then

$$[a_i]_{i \in I_S} = [b_i]_{i \in I_S} \Rightarrow$$
$$\psi_C([a_i]_{i \in I_S}) = \{I_i : a_i\}_{i \in I_S} = \{I_i : b_i\}_{i \in I_S} = \psi_{C'}([b_i]_{i \in I_S})$$

All other elements of S (i.e., not lists) will remain constant. We can see that since $\psi_C(S) = \psi_{C'}(S) \Rightarrow JCan_C(S) = JCan_{C'}(S) \Rightarrow \mathcal{O}^{JCan}(C, S) = \mathcal{O}^{JCan}(C', S)$. Note that $\mathcal{A}^{\mathcal{O}^{JCan}}$ can use only its internal state, coin flips or the random values of C' to decide on what other queries to send the Oracle, which means that those will not help distinguish the 2 distributions (since it will be the same in both cases). As a result $\mathcal{A}^{\mathcal{O}^{JCan}}$ will have 0 advantage above semantic interpretation, in distinguish the two distributions above.

4.3 Real Life Example

In practice, as we mentioned, there will be only one algorithm and the Holder will send to the Verifier the necessary information to canonicalize the derived credential correctly. In effect, $Can_C(S)$ will be $Can(S, Info_C)$ or $Can(S_C)$, where the necessary information is directly encoded in the credentials. That said, all the security requirements defined in Section 3 (i.e., Definitions 4, 5 and 6) apply exactly, with the only difference that $Info_C$ should be passed as an input to the adversary $\mathcal{A}^{\mathcal{O}^{Can}}$ in Definition 6. Accordingly, we define $JCan$ to be Algorithm 2 without the transformation by $\psi_C(\cdot)$ in the first step (i.e., by replacing $\psi_C(\cdot)$ with the Identity function).

Consider the example of Figure 3. The Issuer will start with the credential $C \in \mathcal{CR}[K, V]$. Then, they will transform that credential (using ψ) to the credential $C' \in \mathcal{SCR}[K, V]$ and finally they will canonicalize it to get the messages that they will sign. Next, they will send credential C' along with the signature to the Holder. The Holder will then choose the credential $S \lhd C'$ that they want to disclose and canonicalize it, using again $JCan$. From the messages returned from $JCan(S)$ and $JCan(C')$ (note that $JCan(S) \subseteq JCan(C')$) they will derive a proof (for example a zero-knowledge proof of knowledge of the signature and the rest of the messages) that they will send to the Verifier, together with the credential S. Finally, the Verifier will get the messages $JCan(S)$ and use them to validate the Holder's proof. Note from Figure 3, that the messages returned from $JCan$ depend on the credential C (hence the reason we index $JCan$ with C in our modelling), but they do not reveal any information about the undisclosed values of C (as proven in Lemma 4).

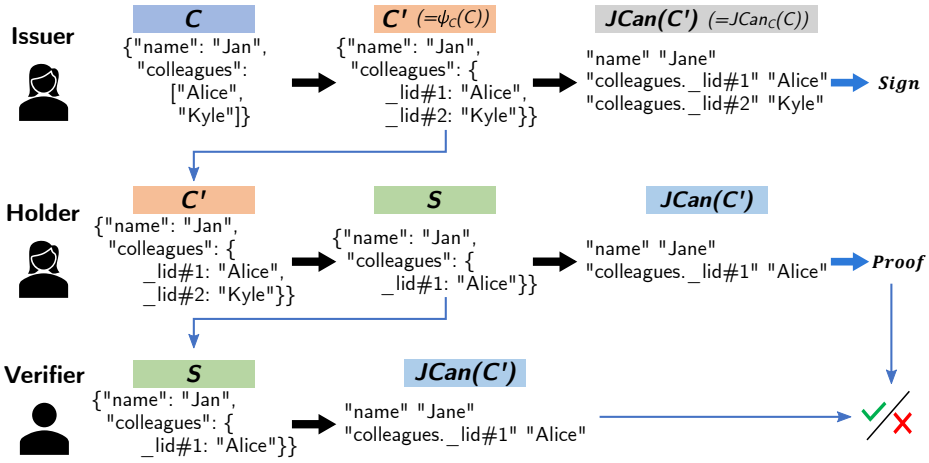

Fig. 3. Real life flow example for the signing, proof derivation and verification of a credential and sub-credential process.

5 Evaluation and Performance

We benchmarked and compared the URDNA canonicalization algorithm and the *JCan* algorithm (Algorithm 2). The benchmarking procedure was executed on a personal computer using an Intel i5 with 3.2 GHz clock and 16 GBs of RAM. Also note that a JSON-LD credential before it is canonicalized it must first be expanded. The expansion process brings the credential to the proper form, so it can be correctly canonicalized. In the results below we do not include the expansion algorithm as part of the process of canonicalizing a credential, to get a more accurate performance measurement of just the URDNA algorithm.

5.1 Benchmarking Results

For the benchmarking procedure[5], we generated credentials of max depth (maximum number of continuously nested objects) 2, 4, 8 and 16. For each different JSON depth, we created credentials with 10, 20, 40 and 80 claims and benchmarked the 2 algorithms multiple times on each credential. Finally, we measured the mean canonicalization time of each algorithm, over the credentials with different number of claims, for each different JSON depth value. The results are shown in Figure 4. Note that *JCan* runs 5× to 9× times faster than the URDNA algorithm. The examples for the benchmarks were generated purposefully to aid URDNA and hinder *JCan*, so we can get an accurate estimation of the performance differences.

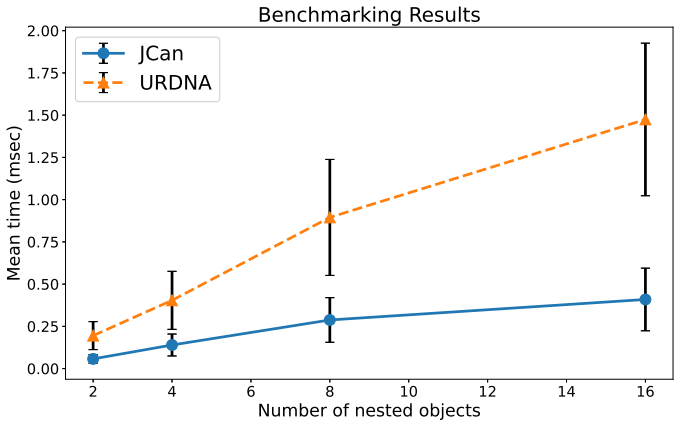

Fig. 4. Benchmarking results comparing the URDNA to *JCan* (Algorithm 2). The credentials used have only one chain of nested objects (i.e., max depth = number of nested objects). This creates a worst case scenario in favor of URDNA over *JCan*.

[5] Code used: https://github.com/BasileiosKal/JCan

6 Conclusion

In this work we considered the use of anonymous credentials using W3C's credentials data model to preserve privacy in user-centric applications. We noted that the cryptographic solutions developed do not consider what the interface with the higher layers will be (i.e., those of JSON and JSON-LD credentials etc.). We then proposed the key security properties that all canonicalization algorithms should possess for the serialization to be secure. We also presented a novel modelling of a JSON credential that we used to propose an efficient algorithm and formally analyse its security. For future work, we want to use our threat model and JSON modelling to analyse and construct security proofs for other canonicalization algorithms as well. We also plan to use those algorithms to construct a complete anonymous credentials system and formally analyse the security of all its aspects. Finally, we want to apply our model on canonicalization algorithms intended to be used by credentials systems supporting additional properties beyond selective disclosure (e.g., range proofs, etc.).

Acknowledgements. This work has been funded in part by subgrant *Securing Content Delivery and Provenance (SECOND)* of EU H2020 project *NGIatlantic.eu*, under grant agreement No 871582.

References

1. Alpár, G., van den Broek, F., Hampiholi, B., Jacobs, B., Lueks, W., Ringers, S.: IRMA: Practical, Decentralized and Privacy-friendly Identity Management using Smartphones. HotPETs 2017 (2017)
2. Arnold, R., Longley, D.: RDF Dataset Canonicalization. https://lists.w3.org/Archives/Public/public-credentials/2021Mar/att-0220/RDFDatasetCanonicalization-2020-10-09.pdf (2020), Accessed: 2022-03-06
3. Bauer, D., Blough, D.M., Cash, D.: Minimal Information Disclosure with Efficiently Verifiable Credentials. In: Proceedings of the 4th ACM Workshop on Digital Identity Management. pp. 15–24. Association for Computing Machinery (2008)
4. Brands, S.: The Problem(s) with OpenID. https://web.archive.org/web/20110516013258/http://www.untrusted.ca/cache/openid.html (2007), Accessed: 2021-09-15
5. Camenisch, J., Lysyanskaya, A.: Signature Schemes and Anonymous Credentials from Bilinear Maps. In: Annual International Cryptology Conference. pp. 56–72. Springer (2004)
6. Camenisch, J., Van Herreweghen, E.: Design and Implementation of the idemix Anonymous Credential System. In: Proceedings of the 9th ACM Conference on Computer and Communications Security. pp. 21–30 (2002)
7. Chaum, D.: Blind Signatures for Untraceable Payments. In: Advances in Cryptology. pp. 199–203. Springer (1983)
8. Chaum, D.: Security without Identification: Transaction Systems to Make Big Brother Obsolete. Communications of the ACM **28**(10), 1030–1044 (1985)
9. Goldwasser, S., Micali, S., Rackoff, C.: The Knowledge Complexity of Interactive Proof-Systems. In: Proceedings of the Seventeenth Annual ACM Symposium on Theory of Computing. p. 291–304. STOC '85, ACM New York, NY, USA (1985)

10. Hanzlik, L., Slamanig, D.: With a Little Help from My Friends: Constructing Practical Anonymous Credentials. In: Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security. pp. 2004–2023 (2021)
11. Kalos, V., Polyzos, G.C.: Verifiable Credentials Selective Disclosure: Challenges and Solutions. https://mm.aueb.gr/master_theses/polyzos/2021-Kalos.pdf (October 2021), M.Sc CS Thesis, Accessed: 2022-02-01
12. Lagutin, D., Kortesniemi, Y., Fotiou, N., Siris, V.A.: Enabling Decentralised Identifiers and Verifiable Credentials for Constrained IoT Devices using OAuth-based Delegation. In: Proceedings of the Workshop on Decentralized IoT Systems and Security (DISS 2019), in Conjunction with the NDSS Symposium, San Diego, CA, USA. vol. 24 (2019)
13. Neira, B., Queern, C.: Introduction to Azure Active Directory Verifiable Credentials. https://docs.microsoft.com/en-us/azure/active-directory/verifiable-credentials/decentralized-identifier-overview (2021), Accessed: 2021-09-15
14. Otto, N., Lee, S., Sletten, B., Burnett, D., Sporny, M., Ebert, K.: Verifiable Credentials Use Cases. Working Group Note, W3C (September 2019), https://www.w3.org/TR/vc-use-cases/
15. Paquin, C., Zaverucha, G.: U-Prove Cryptographic Specification V1.1. Revision 3. Technical Report, Microsoft Corporation (December 2013), https://www.microsoft.com/en-us/research/wp-content/uploads/2016/02/U-Prove20Cryptographic20Specification20V1.1.pdf
16. Sakimura, N., Bradley, J., Jones, M., Medeiros, B.D., Mortimore, C.: OpenID connect core 1.0. https://openid.net/specs/openid-connect-core-1_0.html (2014), Accessed: 2022-02-01
17. Sporny, M., Longley, D., Chadwick, D.: Verifiable Credentials Data Model 1.0. Recommendation, W3C (November 2021), https://www.w3.org/TR/vc-data-model/
18. Sporny, M., Longley, D., Kellogg, G., Lanthaler, M., Champin, P.A., Lindström, N.: A JSON-based Serialization for Linked Data. Recommendation, W3C (July 2020), https://www.w3.org/TR/json-ld11/
19. Zhiyi, Z., Michal, K., Alberto, S., Lixia, Z., Etienne, R.: EL PASSO: Efficient and Lightweight Privacy-Preserving Single Sign On. In: Proceedings on Privacy Enhancing Technologies. pp. 70–87. Sciendo (2021)