

Bachelor Thesis

Unlinkability of Verifiable Credentials in a practical approach

3. Juli, 2024

Joel Robles - Adv. Dr. Annett Laube, Dr. Reto Koenig - Exp. Dr. Andreas Spichiger | TI

Inhaltsverzeichnis

- ▶ Ziel
- ▶ Self-sovereign Identity
- ▶ Verifiable Credentials & Verifiable Presentations
- ▶ OpenID Connect for Verifiable Presentations
- ▶ Fazit
- ▶ Ausblick

Ziel

Was ist das Ziel?

Die Analyse, ob eine Implementierung von Verifiable Credentials mit dem BBS Signature Scheme in der realen Welt Unverknüpfbarkeit beibehält

Self-sovereign Identity

Self-sovereign Identity (SSI)

- Ist ein Konzept, bei dem eine Person (**Holder**) entscheiden kann, wer was über sie wissen darf
- Holder dürfen wählen, was sie offenbaren und was nicht (**selective disclosure**)
- Keine Verbindung zwischen Präsentationen (**unlinkability**)
- Heutiger Stand - Holder haben keine Kontrolle über ihre Attribute
- Zukünftiger Stand dank SSI - Holder haben volle Kontrolle über ihre Attribute

Trust Triangle

- Der Holder zeigt einem Verifier eine staatliche ID
- Diese beinhaltet mehrere Attribute, z.B. Vorname
- Wie weiss ein Verifier, dass eine Menge von Attributen (**Credential**) valid ist?
- Er vertraut dem Issuer!
- Beispiel: Schweizer ID hat Hologramme

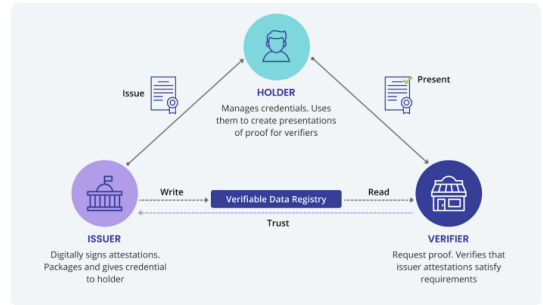


Abbildung: Trust triangle

Verifiable Credentials & Verifiable Presentations

Verifiable Credentials (VC)

- Verifiable Credentials sind eine digitale Repräsentation von physischen Credentials

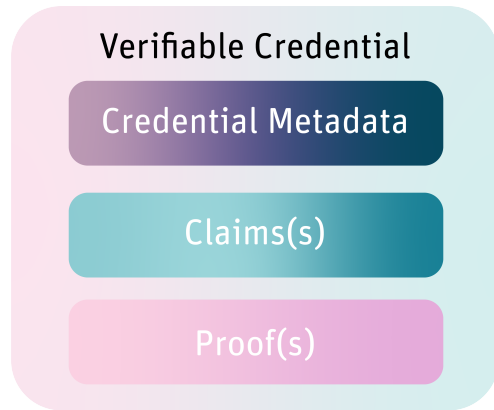


Abbildung: VC Aufbau

Verifiable Credentials (VC)

- Verifiable Credentials sind eine digitale Repräsentation von physischen Credentials
- JSON-LD repräsentiert Attribute als **key-value pairs**
- Beispiel:
 - ▣ Vorname auf einer ID
 - ▣ Repräsentiert als {first_name: "John"}
 - ▣ "first_name" ist der key und "John" ist der value

```
1 {
2   "@context": [
3     "https://www.w3.org/ns/credentials/v2",
4     "https://w3id.org/security/data-integrity/v2",
5     "https://raw.githubusercontent.com/robl...
6   ],
7   "type": [
8     "VerifiableCredential"
9   ],
10  "credentialSubject": {
11    "first_name": "John",
12    "last_name": "Doe",
13    "birth_date": "1.1.1970"
14  },
15  "proof": {
16    "type": "DataIntegrityProof",
17    "cryptosuite": "bbs-2023",
18    "created": "2023-08-15T23:36:38Z",
19    "verificationMethod": "https://example.com/publicKey",
20    "proofPurpose": "assertionMethod",
21    "proofValue": "u2V0C..."
22  }
23 }
```

Abbildung: Beispiel VC

Probleme von digitalen Credentials

Problem 1:

- Holder zeigt ein Credential einem Verifier
- Der Verifier, sieht alle Attribute
- Bricht **selective-disclosure**

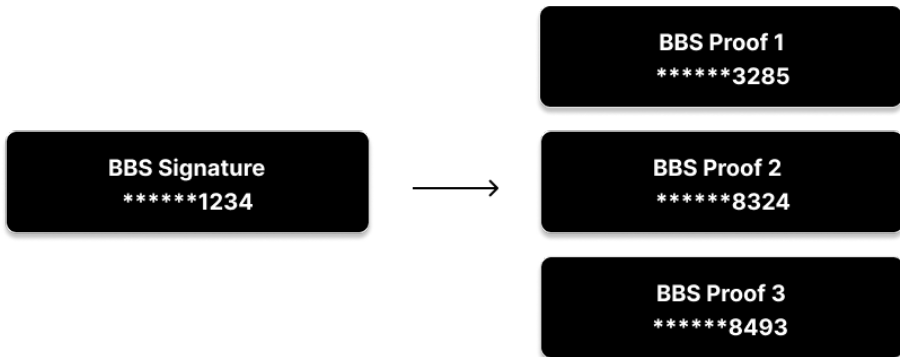
Problem 2:

- Holder zeigt ein Credential einem Verifier
- Holder zeigt das gleiche Credential einem zweiten Verifier
- Der Holder kann ge-linked werden (Metadaten)
- Bricht **unlinkability**

VCs and BBS

- Warum werden sie **Verifiable** Credentials genannt?
- Der Verifier kann ein VC, welches ihm präsentiert wurde (**Verifiable Presentation**), aufgrund kryptographischer Signaturen verifizieren
- Diese zeigen, dass das Credential seit der Ausstellung nicht verändert wurde
- Wir nutzen das BBS Signature Scheme (**BBS**)
- Dieses Schema bietet **selective disclosure** und **unlinkability**
- Aber wie unlinkability? - Der Verifier braucht die Signatur
- BBS kann **proofs** generieren
- Diese beweisen, dass der Holder die Signatur kennt, ohne diese zu offenbaren
- Fungieren als neue Signatur für das selectively disclosed VC
- Weiter sind proofs unverknüpfbar zwischen jeder Generierung

BBS proofs



BBS proofs

```
1 {
2   "@context": [
3     "https://www.w3.org/ns/credentials/v2",
4     "https://w3id.org/security/data-integrity/v2",
5     "https://raw.githubusercontent.com/robl..."
6   ],
7   "type": [
8     "VerifiableCredential"
9   ],
10  "credentialSubject": {
11    "first_name": "John",
12    "last_name": "Doe",
13    "birth_date": "1.1.1970"
14  },
15  "proof": {
16
17
18
19
20
21
22
23 }
```

BBS Signature

Abbildung: VC mit BBS Signatur

```
1 {
2   "@context": [
3     "https://www.w3.org/ns/credentials/v2",
4     "https://w3id.org/security/data-integrity/v2",
5     "https://raw.githubusercontent.com/robl..."
6   ],
7   "type": [
8     "VerifiableCredential"
9   ],
10  "credentialSubject": {
11    "first_name": "John",
12    "Blinded"
13    "birth_date": "1.1.1970"
14  },
15  "proof": {
16
17
18
19
20
21
22
23 }
```

BBS Proof

Abbildung: VC mit BBS Proof

Verifiable Presentation (VP)

- Ein Holder würde gerne ein VC präsentieren
- Dafür werden **Verifiable Presentations** genutzt
- BBS kann nur Statements signieren
- Der **RDF** canonicalization Algorithmus, welcher Statements aus key-value pairs generiert

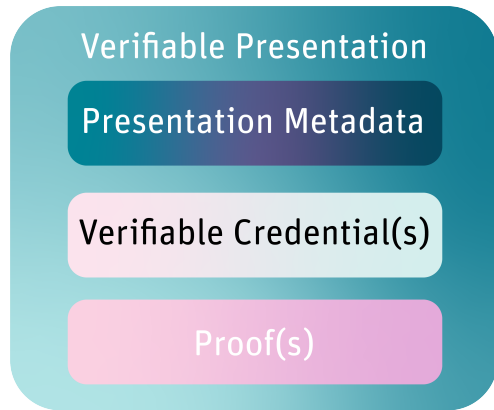


Abbildung: VP Aufbau

Der RDF Algorithmus

JSON zu Statements

```
1 {
2   "@context": [
3     "https://www.w3.org/ns/credentials/v2",
4     "https://w3id.org/security/data-integrity/v2",
5     "https://raw.githubusercontent.com/robl...
6   ],
7   "type": ["VerifiableCredential"],
8   "credentialSubject": {
9     "first_name": "John",
10    "last_name": "Doe",
11    "birth_date": "1.1.1970"
12  }
13 }
```


Abbildung: Beispiel VC

```
1 _:_1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_0 <https://schema.org/
  birthDate> \"1.1.1970\" .\n
2 _:_1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_0 <https://schema.org/
  familyName> \"Doe\" .\n
3 _:_1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_0 <https://schema.org/
  givenName> \"John\" .\n
4 _:_1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_1 <http://www.w3.org/1
  999/02/22-rdf-syntax-ns#type> <https://www.w3.org/2018/credent
  ials#VerifiableCredential> .\n
5 _:_1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_1 <https://www.w3.org/
  2018/credentials#credentialSubject> _:_1c6e5c99-c7a8-4e76-a5ab
  -3ac3074926e3_0 .\n
```

Abbildung: Beispiel Statements

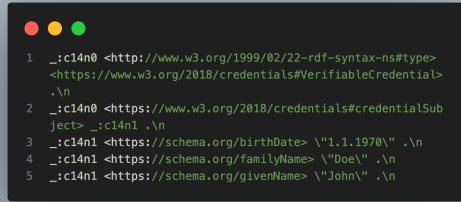
Der RDF Algorithmus

Statements sortieren



```
1  _:1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_0 <https://schema.org/
  birthDate> \"1.1.1970\" .\n
2  _:1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_0 <https://schema.org/
  familyName> \"Doe\" .\n
3  _:1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_0 <https://schema.org/
  givenName> \"John\" .\n
4  _:1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_1 <http://www.w3.org/1
  999/02/22-rdf-syntax-ns#type> <https://www.w3.org/2018/credent
  ials#VerifiableCredential> .\n
5  _:1c6e5c99-c7a8-4e76-a5ab-3ac3074926e3_1 <https://www.w3.org/
  2018/credentials#credentialSubject> _:1c6e5c99-c7a8-4e76-a5ab
  -3ac3074926e3_0 .\n
```

Abbildung: Beispiel Statements



```
1  _:c14n0 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type>
  <https://www.w3.org/2018/credentials#VerifiableCredential>
  .\n
2  _:c14n0 <https://www.w3.org/2018/credentials#credentialSub
  ject> _:c14n1 .\n
3  _:c14n1 <https://schema.org/birthDate> \"1.1.1970\" .\n
4  _:c14n1 <https://schema.org/familyName> \"Doe\" .\n
5  _:c14n1 <https://schema.org/givenName> \"John\" .\n
```

Abbildung: Beispiel Sortierung

Permutation von Statements

- Holder bekommt ein VC, welches unter anderem ein Zivilstands-Attribut beinhaltet
- Holder präsentiert ein VP mit verborgenen Zivilstands-Attribut
- Holder heiratet, bekommt ein neues VC mit geändertem Zivilstand
- Holder präsentiert das aktualisierte VP mit verborgenen Zivilstand
- **Datenleck:** Der Verifier kann herausfinden, dass sich der Zivilstand geändert hat
- Aber nur wenn Attribute, welche nach dem Zivilstand kommen, präsentiert werden (RDF Algorithmus)
- Damit das nicht passieren kann, muss der Issuer immer die Statements zufällig permutieren
- Der Issuer muss die Permutation dem Holder bekannt geben, aber **nie** dem Verifier

Verknüpfbarkeit von Identifikatoren & Metadaten

- VCs können Metadaten wie Ablaufdatum beinhalten
- Falls das Ablaufdatum sehr genau ist (z.B. auf die Sekunde), führt dies zu Verknüpfbarkeit
- Ablaufdatum auf einen Tag genau, um die Verknüpfbarkeit zu umgehen
- VCs und VPs können auch IDs für z.B. Entziehung beinhalten, welche zu Verknüpfbarkeit führen
- Dafür kan man Zero-Knowledge proofs verwenden, um zu zeigen, dass man nicht Teil einer Entzugs-Liste ist

OpenID Connect for Verifiable Presentations

Transport zwischen Holder und Verifier

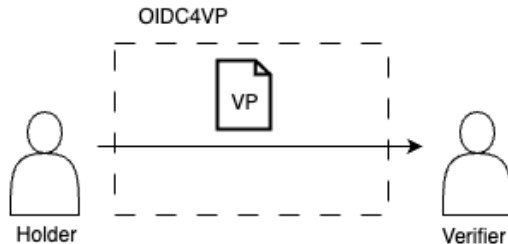


Abbildung: OpenID connect for Verifiable Presentations

OIDC4VP Fluss

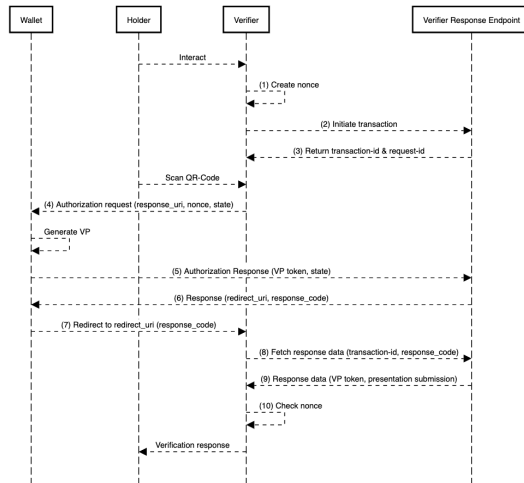


Abbildung: OpenID connect for Verifiable Presentations Sequenz

Replay attack

- Der Holder sendet dem Verifier ein VP
- Ein *Man-in-the-Middle* speichert die Vorstellung
- Der *Man-in-the-Middle* kann das gespeicherte VP wiederverwenden
- Um dieses Problem zu umgehen, wird eine Zufallszahl genutzt (challenge-response)

Fazit

Fazit

- Wie kombiniert man BBS und VCs?
 - ▣ **Durch den RDF Algorithmus**
- Was sind die Probleme mit dieser Kombination und wie kann man diese lösen?
 - ▣ **Determinismus des RDF Algorithmus und Verknüpfbarkeit von Metadaten und Identifikatoren**
 - ▣ **Lösung: Permutierung der Statements und Gebrauch von zero-knowledge proofs**
- Wie werden die VPs von Holder zu Verifier gesendet und was sind die Probleme?
 - ▣ **OpenID connect for Verifiable Presentations**
 - ▣ **Replay-Attacke, gelöst mit der nonce**

Es funktioniert!

Ausblick

Ausblick

- Link Secrets und Blind BBS Signatures für linkability und selective disclosure analysieren
- Implementieren und testen