

# Bachelor Thesis

## Unlinkability of Verifiable Credentials in a practical approach

June 5, 2024

Joel Robles | TI

---

# Inhaltsverzeichnis

- ▶ Ziel
- ▶ Self-sovereign Identity
- ▶ Verifiable Credentials
- ▶ Verifiable Presentations
- ▶ Sicherheitsüberlegungen von VC/VPs
- ▶ OpenID Connect for Verifiable Presentations
- ▶ Sicherheitsüberlegungen von OIDC4VP
- ▶ Fazit
- ▶ Ausblick

Ziel

# Was ist das Ziel?

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

# Self-sovereign Identity

---

# Self-sovereign Identity (SSI)

- Ist ein Konzept wo eine Person (**holder**) entscheiden kann, wer was über sie wissen darf
- Holders dürfen wählen was sie offenbaren und was nicht, auch bekannt als **selective disclosure**
- Erstes Problem:
  - Holder zeigt eine Staatliche ID
  - Ist eine Menge von Daten oder eine Menge von **Attributen**
  - Die Person welche verifiziert sieht alle Attribute
- Zweites Problem:
  - Holder zeigt Attribute einer Person die diese verifizieren will, bekannt als **verifier**
  - Holder zeigt die gleichen Attribute einem zweiten **verifier**
  - Der holder kann ge-**linked** werden
- Heutiger stand - Holder haben keine Kontrolle über ihre Attribute
- Zukünftiger stand dank SSI - Holder haben volle Kontrolle über ihre Attribute

# Trust Triangle

- Wie weis ein verifier das eine Menge von Attributen (**credential**) valid ist?
- Er vertraut dem issuer!
- Beispiel: Schweizer ID hat Hologramme

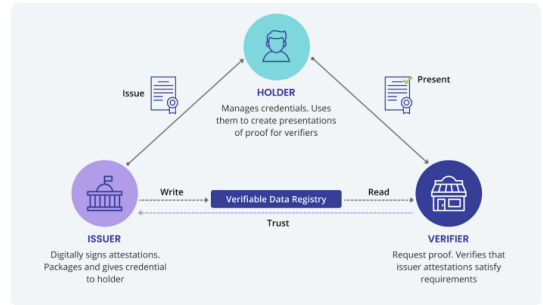


Abbildung: Trust triangle

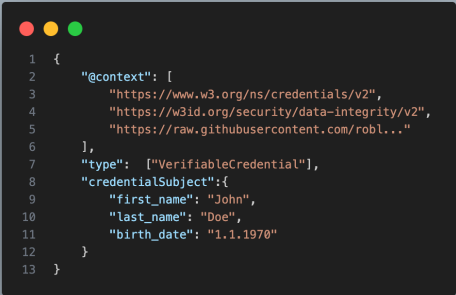
# Verifiable Credentials

---



# 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": ["VerifiableCredential"],
8    "credentialSubject": {
9      "first_name": "John",
10     "last_name": "Doe",
11     "birth_date": "1.1.1970"
12   }
13 }
```

Abbildung: Beispiel VC

# VCs and BBS


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

# Verifiable Presentations

---

# 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



```
1 _:b0 <https://schema.org/birthDate> \"1.1.1970\" .\n2 _:b0 <https://schema.org/familyName> \"Doe\" .\n3 _:b0 <https://schema.org/givenName> \"John\" .\n4 _:b1 <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <https://www.w3.org/2018credentials#VerifiableCredential> .\n5 _:b1 <https://www.w3.org/2018/credentials#credentialSubject> _:b0 .\n
```

**Abbildung:** Beispiel canonicalized VP

# Der RDF Algorithmus

## JSON zu staments

```
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 statemetns

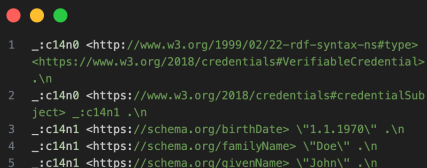
# Der RDF Algorithmus

## 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 statemetns



```
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

# Sicherheitsüberlegungen von VC/VPs

---

# Permutation von statements

- Holder präsentiert ein VP mit verborgenen Zivilstands-Attributen
- Holder heiratet bekommt ein neues VP mit geändertem Zivilstand
- Holder präsentiert das aktualisierte VP mit verborgenem Zivilstand
- **Datenleck:** Der verifier kann herausfinden, dass sich der Zivilstand geändert hat
- 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



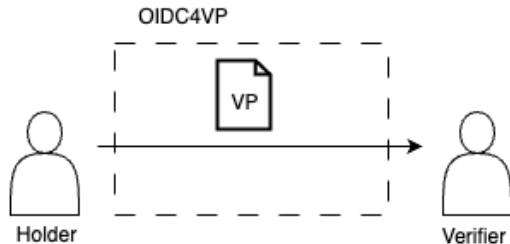
# Verknüpfbarkeit von Identifikatoren & Metadaten

- 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 das man nicht Teil einer Entzugs-Liste ist

# OpenID Connect for Verifiable Presentations

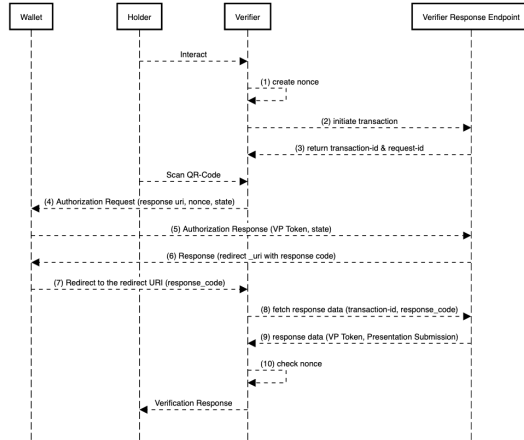
---

# Transport zwischen holder und verifier



**Abbildung:** OpenID connect for Verifiable Presentations

# OIDC4VP Fluss



**Abbildung:** OpenID connect for Verifiable Presentations Fluss

# Sicherheitsüberlegungen von OIDC<sub>4</sub>VP

---

# 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 zufalls Nummer genutzt (challenge-response)

# Fazit

---

# Fazit

- Was hat BBS für Vorteile?
  - Selective-disclosure und unlinkability
- Wie funktionieren VCs/VPs?
- Wie kann man VCs/VPs und BBS verbinden?
  - Kanonisierung durch RDF Algorithmus
- Wie werden die VPs von holder zu verifier gesendet?
  - OpenID connect for Verifiable Presentations



**Es funktioniert!**

Ausblick

# Ausblick

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