



UNIVERSITY OF  
OXFORD

# Verifiable Credentials with expressive zero knowledge query

JESSE WRIGHT | FEB 2025





# DPhil @ Oxford

## Building neurosymbolic Web agents that

- Represent legal entities
- Support user autonomy
- Protect user data





## Solid Lead via Open Data Institute

Solid is an open standard for managing digital identities and storing personal data for re-use across applications on the Web. The goal of Solid is for people to have more agency over their data.



The logo for Inrupt, featuring the word "inrupt" in white lowercase letters on a blue-to-cyan gradient square background.

inrupt



## Formerly working on Solid @ Inrupt

- Enterprise software engineer and data architect
- Inrupt donated a Solid-based data wallet to the Open Wallet Foundation





# W3C

## Groups 14

- Decentralized Identifier Working Gr...
- Linked Web Storage Working Group
- Notation 3 (N3) Community Group
- RDF Dataset Canonicalization and ...
- RDF JavaScript Libraries Commun...
- RDF Surfaces Community Group
- RDF Test Suite Curation Communi...
- RDF-star Working Group
- SHACL Community Group
- Solid Community Group





UNIVERSITY OF  
OXFORD

What is a wallet?



UNIVERSITY OF  
**OXFORD**

**A way to prove someone said something**



UNIVERSITY OF  
OXFORD

*A generalisation* is to supply some evidence that  
you can **take something to be true**

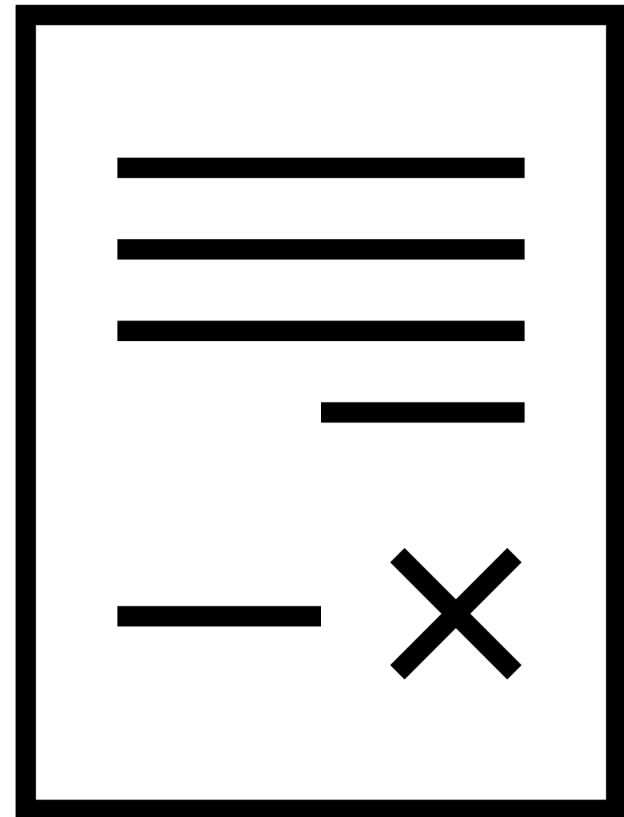
We call this provenance



The two core  
features (in my  
opinion) of  
(W3C) Verifiable  
Credentials



The two core  
features (in my  
opinion) of  
(W3C) Verifiable  
Credentials





UNIVERSITY OF  
OXFORD

But lets consider the use cases ...



Secondary Data Reuse

gaia-x





UNIVERSITY OF  
OXFORD

Give me an anonymised dataset of the **age** and **BMI**  
of all men in the UK



# On demand data integrity



UNIVERSITY OF  
OXFORD

Prove that you're eligible to hire a car in this country.



UNIVERSITY OF  
OXFORD

# Prove that you're eligible to hire a car in this country.

What if the company didn't need to write bespoke business logic integrating age, driving, and travel (e.g. visa) credentials?

Can the standards make this data integration easier?

Can the standards make this data integration more privacy preserving?





UNIVERSITY OF  
OXFORD

Can we support zero knowledge proof over  
arbitrary SPARQL query?



```
:UKDrivingAuthority :claims <<:Jesse :dob "06-00-2000"^^xsd:dateTime>> .
  :signature [...] .

:UKImmigrationAuthority :claims <<:Jesse :hasCitizenship :Australia>> .
  :signature [...] .

:UKImmigrationAuthority :claims <<:Australia a :CommonwealthCountry>> .
  :signature [...] .
```

I want to be able to execute the following SPARQL ASK query (API to be refined). I also want to be able to execute all other read-only SPARQL operations (SELECT and CONSTRUCT).

```
ASK {
  :Jesse :dob ?x .
  :hasCitizenship [ a :CommonwealthCountry ]

  FILTER (?x >= "03-01-2006"^^xsd:dateTime)
}
```

And have a zero-knowledge proof proving that the statement is true if you trust claims issued by :UKDrivingAuthority and :UKImmigrationAuthority .



UNIVERSITY OF  
OXFORD

# RDF and Query Overview



<https://eyereasoner.github.io/eye-js/example/>

<https://query.comunica.dev/>

---



```
SPARQL GraphQL-LD
1 CONSTRUCT {
2   ruben:me dbpedia-owl:sharesBirthPlace ?person ;
3   dbpedia-owl:olderThan ?person .
4 } WHERE {
5   ruben:me dbpedia-owl:birthPlace ?place;
6   dbpedia-owl:birthDate ?dateR.
7   ?person dbpedia-owl:birthPlace ?place;
8   dbpedia-owl:birthDate ?date.
9   FILTER(?date > ?dateR)
10 }
```

Press CTRL - <spacebar> to autocomplete

Execute query

18 results in 0.4s

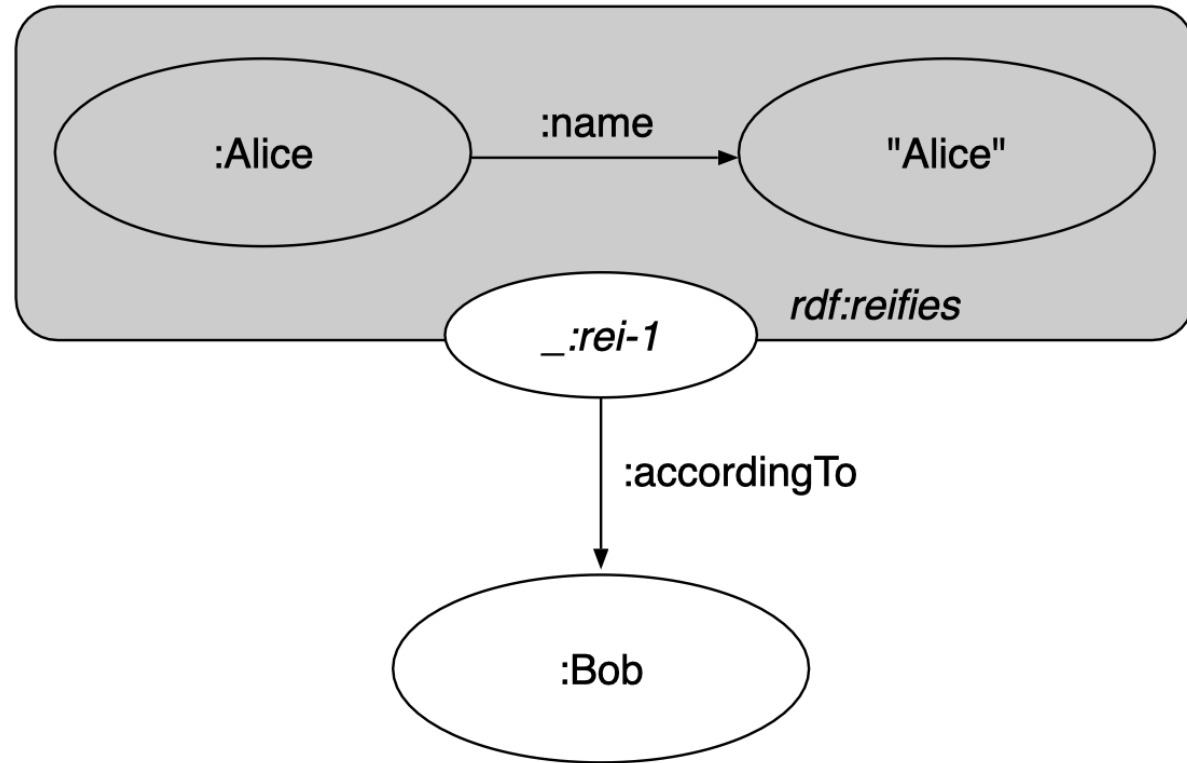
### Query results:

ruben:me dbpedia-owl:sharesBirthPlace dbpedia:Birger\_Verstraete;  
dbpedia-owl:olderThan dbpedia:Birger\_Verstraete;  
dbpedia-owl:sharesBirthPlace dbpedia:Brecht\_Capon;  
dbpedia-owl:olderThan dbpedia:Brecht\_Capon;  
dbpedia-owl:sharesBirthPlace dbpedia:Dieter\_Wittesaele;  
dbpedia-owl:olderThan dbpedia:Dieter\_Wittesaele;  
dbpedia-owl:sharesBirthPlace dbpedia:Divock\_Origi;  
dbpedia-owl:olderThan dbpedia:Divock\_Origi;  
dbpedia-owl:sharesBirthPlace dbpedia:Igor\_Vetokele;  
dbpedia-owl:olderThan dbpedia:Igor\_Vetokele;  
dbpedia-owl:sharesBirthPlace dbpedia:Jelle\_Florizoone;  
dbpedia-owl:olderThan dbpedia:Jelle\_Florizoone;  
dbpedia-owl:sharesBirthPlace dbpedia:Kimmer\_Coppejans;  
dbpedia-owl:olderThan dbpedia:Kimmer\_Coppejans;  
dbpedia-owl:sharesBirthPlace dbpedia:Niels\_Coussement;  
dbpedia-owl:olderThan dbpedia:Niels\_Coussement;  
dbpedia-owl:sharesBirthPlace dbpedia:Oliver\_Naesen;



UNIVERSITY OF  
OXFORD

# RDF 1.2 Overview





```
PREFIX : <file:///Users/jesght/Documents/GitHub/jeswr/rust-test/bar/example.sparql#>
PREFIX dbpedia-owl: <http://dbpedia.org/owl/>
PREFIX ruben: <https://ruben.verborgh.org/>

CONSTRUCT {
  [] :asserts <<ruben:me dbpedia-owl:sharesBirthPlace ?person; dbpedia-owl:olderThan ?person>> .
  | :proof ?derivedProof .

  BIND_PROOF(?derivedProof)
} WHERE {
  ?e1 :asserts <<ruben:me dbpedia-owl:birthPlace ?place; dbpedia-owl:birthDate ?dateR>> .
  | :proof ?proof1 .

  ?e2 :asserts <<?person dbpedia-owl:birthPlace ?place; dbpedia-owl:birthDate ?dateR>> .
  | :proof ?proof2 .

  FILTER(?date > ?dateR)
}
```





```
PREFIX : <file:///Users/jesght/Documents/GitHub/jeswr/rust-test/bar/example.sparql#>
PREFIX dbpedia-owl: <http://dbpedia.org/owl/>
PREFIX ruben: <https://ruben.verborgh.org/>

CONSTRUCT {
  [] :asserts <<ruben:me dbpedia-owl:sharesBirthPlace ?person; dbpedia-owl:olderThan ?person>> .
  | :proof ?derivedProof .

  BIND_PROOF(?derivedProof)
} WHERE {
  ?e1 :asserts <<ruben:me dbpedia-owl:birthPlace ?place; dbpedia-owl:birthDate ?dateR>> .
  | :proof ?proof1 .

  ?e2 :asserts <<?person dbpedia-owl:birthPlace ?place; dbpedia-owl:birthDate ?dateR>> .
  | :proof ?proof2 .

  FILTER(?date > ?dateR)
  FILTER(?e1 a :EUGov)
  FILTER(?e2 a :EUGov)
}
```



```
PREFIX : <file:///Users/jesght/Documents/GitHub/jeswr/rust-test/bar/example.sparql#>
PREFIX dbpedia-owl: <http://dbpedia.org/owl/>
PREFIX ruben: <https://ruben.verborgh.org/>

CONSTRUCT {
  [] :asserts <<ruben:me dbpedia-owl:sharesBirthPlace ?person; dbpedia-owl:olderThan ?person>> .
  :proof ?derivedProof
```

The derived proof should reveal **at most\*** which entities **stated** the facts used to **establish** the **derived facts**.



UNIVERSITY OF  
OXFORD

Now let's talk Zero Knowledge Proof



# Naïve approach

- Take a Zero Knowledge Virtual Machine (ZKVM)
- Add a SPARQL Query Engine
- Done (?)

*The RISC Zero zero-knowledge virtual machine zkVM (zkVM) lets you prove correct execution of arbitrary Rust code.*

*Anyone with a copy of the receipt can verify the guest program's execution and read its publicly shared outputs.*





# Naïve approach

<https://github.com/jeswr/risc0-sparql-poc/blob/main/core/src/lib.rs>



RISC  
ZERO





```
You, 3 weeks ago | 2 authors (You and one other)
CONSTRUCT {
  ?entity <https://example.org/ns#isAdult> ?adult
}
WHERE {
  ?entity <http://xmlns.com/foaf/0.1/age> ?age.
  BIND(?age >= 18 as ?adult)
}

PUT AZURE PROBLEMS TERMINAL TRUFFLE COMMENTS

> > ∨ TERMINAL
⊕ ⚙ Proving took 62.089464875s
Data hash: "95e173cfefe22794b617433c8adc6196856e5f7e34cca5a8aaa9226ef25fecc1"
Query hash: "84f9506febcb9d5cf176e365674d6b5a2be3c9121e92ab881e4e4ca40b984924"
Result hash: "8552537b8cd4bc9a68a365214bf5ab68d83f34a405337b48e00265e94ddf05f0"
Output result"<https://mypod.org/alice/profile/card#me> <https://example.org/ns#isAdult> \"true\"^^<http://www.w3.org/2001/XMLSchema#boolean> .\n"
Verification took 32.542208ms
```



# Gotcha's

- Proving time
- Proof is code dependent
- Hash and proof description is not part of SPARQL result

## RISC ZERO

```

      .: ^!777!~::~: ^?7JGG&@&5      .: ^!77!
      :7PGJ^~: YPJ5J7JB@@@@@#:      : YBP7:
      .!P&@5JP?5GGP5. 7@@@@@B^      :?J~^: J@@&P!
      .!Y@@@@@#5PG5JB@Y .P@@@&GGJ!~. .B@&B5!&@@@@@B7.
      .7JJP@@@@@#P!!YBGP 7@@@@5^ .!7? .?@@G:#&&@@@@@@@@#7
      ~Y~7@@@@@G?.:!GP?P~ .YB5      !! 7P5@5.7@@@@@@@@@
      ?J.7@@@@@&~ G@YGA      !GBP^ 7&GJ#@@@@@@@@@
      .Y7 Y@@@@@@@@@Y~G@@@@@G!      ~7J&GP@@@@@@@@@@@@
      5!^B@@@@@@@@@&&@&&B&G#^      ~G@@@@@&@GJY&&7#
      Y7. G@@@@@@@@@@@@@&&J!!5Y:      !BP#@P?!?YPG@#Y??5@B~
      7Y ^@@@@@@@@@@@@@B!^7:      7B&#5^7!~!:P!Y##@@B
      .P. ?@@@@@@@@@@@@P!:      .?#B#@P7:~:::J@@@
      7J Y@@@@@@@@@BY.      .5@@@@@@@@@&&@&&PB@@
      P~ 75@@@@5J5#P      .Y&@@@@@@@@@@@@@@@@@YGA
      .G. ^@@P ^?      7@@@@@@@@@@@@@@@@@@@@@75
      .G. Y@#~?P^^~J?^!:.      J@@@@@@@@@@@@@@@@@@@@@7
      .G. ^JGG! .:~J7^      7#@@@@@@@@@@@@@@@@@@@@@
      5~ ^Y@? ^~^..      :5&@@@@@B@@@@@@@@@@@@
      7J ~YY7?@@@&B!..      :!~::~. ~7B@@@@@@@@@@@@
      .G. .7@@@@@#B?      G@@@@@@@@@#!
      7Y .B@@@@@@@@@5Y7^^.      :B@@@@@@@@@?
      Y7 G@@@@@@@@@@@@@@@@@#J      G@@@@@@@@@!
      5! .P@@@@@@@@@@@@@@@@@G^      .P@@@@@@@@B7~G
      .Y7 ~#@@@@@@@@@@@@@~      5@@@@@P .P7
      ?J. :^J&@@@@@@@@@~      Y@@@@@7 ..
      ~Y! ?@@@@@@@@@Y!:      7@&G?: ~
      .7J~ ^&@@@@@B:      :^: ^?7
      .7J~. J@@@@@5^      : ^?7.
      .!7^ .P@@G~      :!7~.
      :!77^ .J@B.      :!7!:.
      :~77!~: .^??^:      .: ^!7!~:
      :^!7!~::~: ^?7JGG&@&5      .: ^!77!
  
```



# Optimization!

---





- [Circuitree](#) [[code](#)]: A Datalog Reasoner in Zero-Knowledge
  - Pros:
    - Close to SPARQL technology (existing N3 reasoning and SPARQL querying engines are built on top of)
  - Cons:
    - Not tailored for proof of derived facts - and likely has large proofs size; instead is defined for full ZKP of arbitrary prolog execution.
- [Lean ZKP](#) [[code](#)]: A zero knowledge theorem prover compatible with lean4:
  - Pros:
    - Specialised for proving that theorems are true in zero knowledge - including SAT and other formal logic problems. This is a problem that can easily be translated into "is the following query true given this set of facts".
    - Has a relatively small proof size according to the paper
  - Cons:
    - Codebase appears to have been unmaintained for last 8 months
    - Unclear how easy it would be to translate this to a production system - would a theorem prover be needed to make every implementation work?
    - Would likely need to collaborate directly with the original author to make this work.
- [ZKSMT](#) [[code](#)]: Similar to Lean ZKP, but specifically targeted towards SAT problems



100-1000x

---



# Standardization!

---



## Standardization

- Bind the proof to the *SPARQL 1.2 algebra* rather than to the Rust Risc-v implementation
- This could become a new proof method in W3C verifiable credentials, but ...





# Abstraction!

---

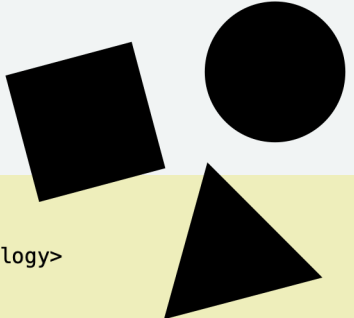


# User abstractions != Standards abstractions

- The and wallet for *work, personal* etc.; and the credential for *education, transport, health* should be **user abstractions** not what we are tied to in the specifications.

# Shapes as a method for shaping credentials

- Data shapes languages such as SHACL enable the ability to query, frame and validate data; and thus generate a credential that conforms to an expected schema.



```
BASE <http://example.com/ns>

IMPORTS <http://example.com/person-ontology>

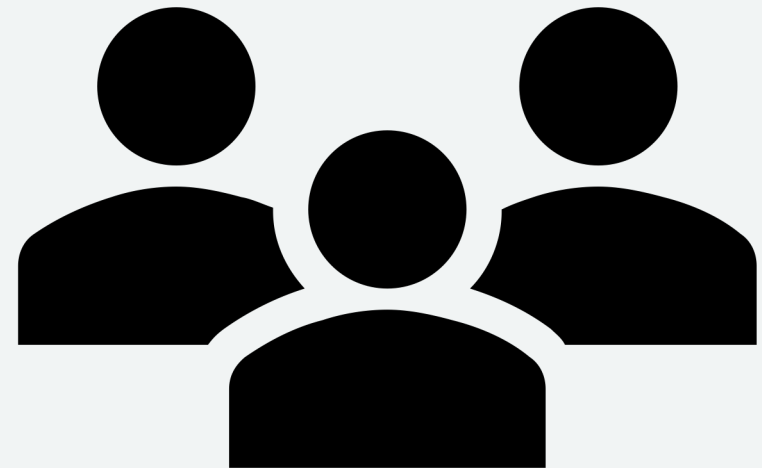
PREFIX ex: <http://example.com/ns#>

shape ex:PersonShape -> ex:Person {
  closed=true ignoredProperties=[rdf:type] .

  ex:ssn      xsd:string [0..1] pattern="^\d{3}-\d{2}-\d{4}$" .
  ex:worksFor IRI ex:Company [0..*] .
  ex:address  BlankNode [0..1] {
    ex:city xsd:string [1..1] .
    ex:postalCode xsd:integer|xsd:string [1..1] maxLength=5 .
  } .
}
```

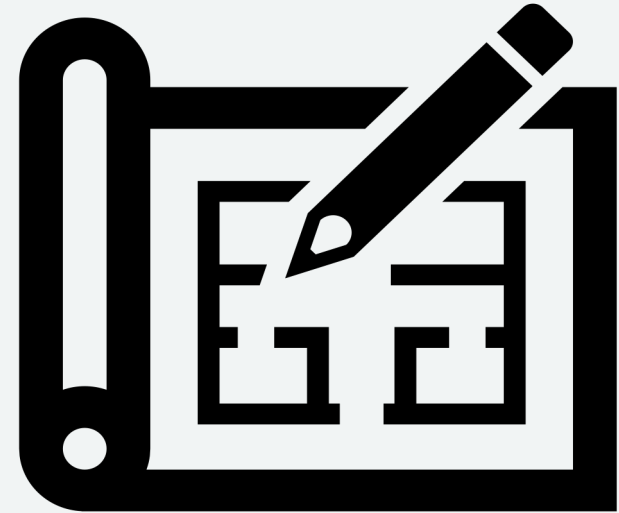
# Call for collaboration

- Performance optimisation (rust)
- ZKP: Custom algo design with specialised circuits
- Modelling (rdf-star)
- Deployment in particular use-cases, including:
  - "the better versions" of examples from the Gamma trust framework
  - Classes of use cases which require the "integrate and derive" pattern
  - Perhaps too academic for some "declarative OIDC"





# Call for use-cases



# Future Work

- Emergent multi-party computation
  - User data stores declare “I permit my salary to be used to publicly declare the average salary of my workplace, but **no one** can know my individual salary”
- Policy aware query
  - See ODRL:  
<https://www.w3.org/community/reports/odrl/CG-FINAL-profile-bp-20240808.html>





# My Recommended Reading List

- <https://www.w3.org/TR/sparql12-query/>
- <https://www.w3.org/community/reports/odrl/CG-FINAL-profile-bp-20240808.html>
- <https://solidproject.org>
- <https://www.w3.org/TR/prov-o/>

## Questions

Email: [jesse@jeswr.org](mailto:jesse@jeswr.org)  
Mastodon: [jeswr@sfba.social](https://sfba.social/@jeswr)

