



UNIVERSITY OF
OXFORD

Verifiable Credentials with expressive zero knowledge query

JESSE WRIGHT | FEB 2025





DPhil @ Oxford

Building neurosymbolic Web agents and privacy preserving query





Solid Project Lead @ 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.



Issuer



Passport department

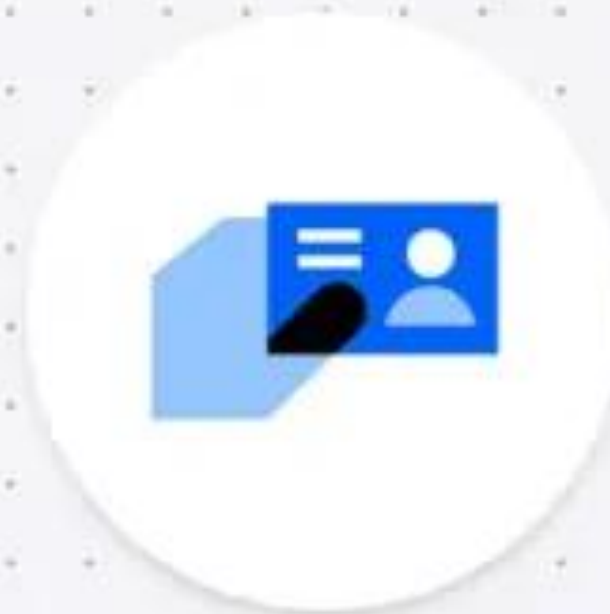


Training program



University

Holder



Credential holder

Verifier



Employer



Online shop



Bank

Request proof

Present proof



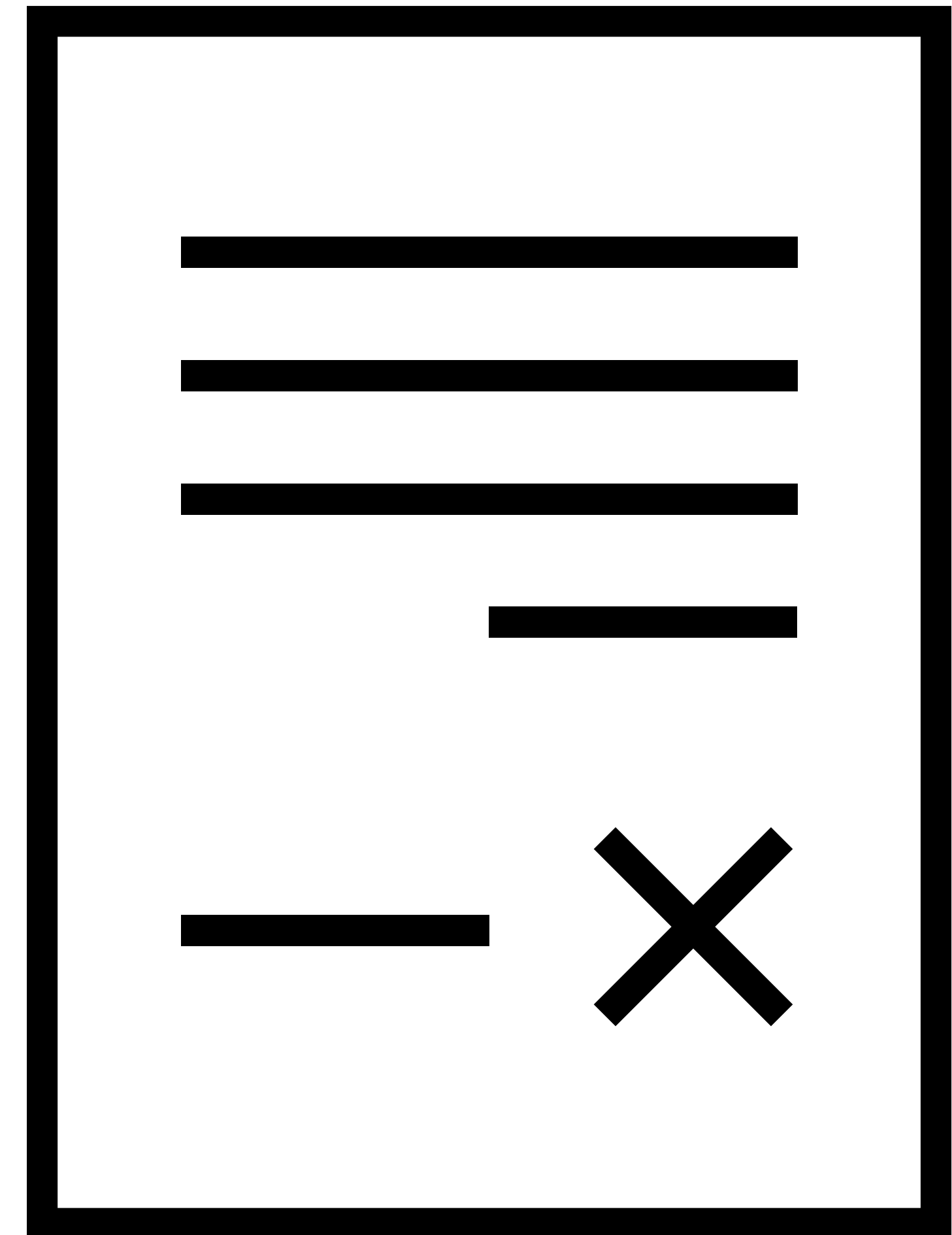
EXAMPLE 15: A verifiable credential using an embedded proof

```
{
  "@context": [
    "https://www.w3.org/ns/credentials/v2",
    "https://www.w3.org/ns/credentials/examples/v2"
  ],
  "id": "http://example.gov/credentials/3732",
  "type": ["VerifiableCredential", "ExampleDegreeCredential"],
  "issuer": "did:example:6fb1f712ebe12c27cc26eebfe11",
  "validFrom": "2010-01-01T19:23:24Z",
  "credentialSubject": {
    "id": "https://subject.example/subject/3921",
    "degree": {
      "type": "ExampleBachelorDegree",
      "name": "Bachelor of Science and Arts"
    }
  },
  "proof": {
    "type": "DataIntegrityProof",
    "cryptosuite": "eddsa-rdfc-2022",
    "created": "2021-11-13T18:19:39Z",
    "verificationMethod": "https://university.example/issuers/14#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "z58DAdFfa9SkqZMVPxAQp...jQCrFPP2oumHKtz"
  }
}
```

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

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

Interface Options

SPARQL PROVE

```
SELECT PROVE ?s WHERE {
  ?s :dob ?x .
  :hasCitizenship [ a :CommonwealthCountry ] .

  FILTER(?x > "03-01-2006")
}
```

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

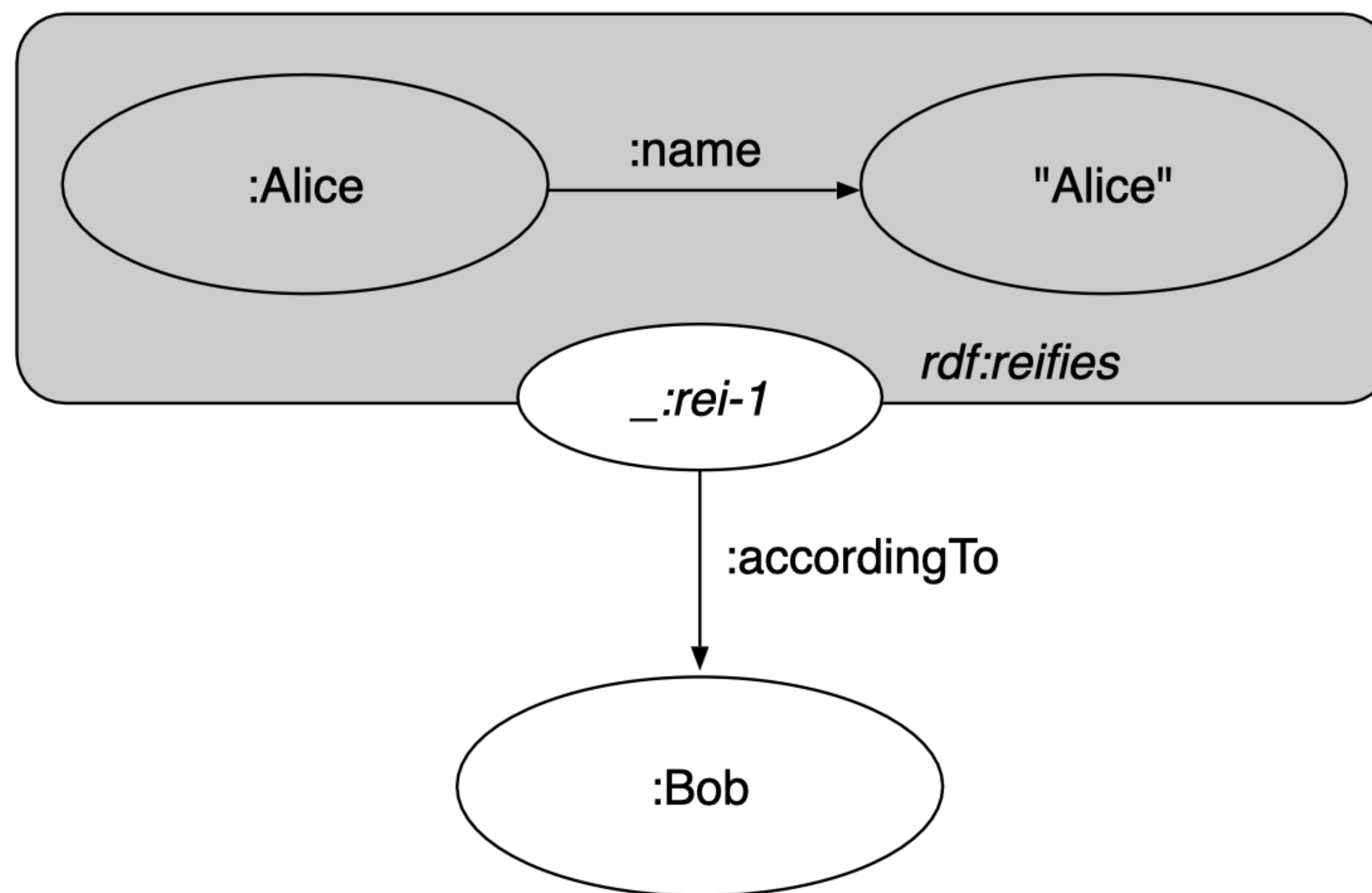
  FILTER(?x > "03-01-2006")
}
```

```
{
  "head": {
    "vars": [
      "o",
      "s",
      "!"
    ]
  },
  "results": {
    "bindings": [
      {
        "o": {
          "type": "uri",
          "value": "http://www.w3.org/2002/07/owl#Class"
        },
        "s": {
          "type": "uri",
          "value": "http://example.com/Person"
        },
        "!proof": {
          "type": "ProofObject",
          "value": {
            "derivationProof": {
              "type": "RiscZero-verification",
              "programId": "24;oitn[q934j[t0qmgmergbnqn23r8pweraggwoeFINa",
              "proof": "IU0iubIOUBiubIUOB...jQCrFPP2oumHKtz"
            }
          }
        }
      }
    ]
  },
  "!sources": [{
    "type": "DataIntegrityProof",
    "cryptosuite": "eddsa-rdfc-2022",
    "created": "2021-11-13T18:19:39Z",
    "verificationMethod": "https://university.example/issuers/14#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "IU0iubIOUBiubIUOB...jQCrFPP2oumHKtz"
  }, {
    "type": "DataIntegrityProof",
    "cryptosuite": "eddsa-rdfc-2022",
    "created": "2021-11-13T18:19:39Z",
    "verificationMethod": "https://bank.example/issuers/14#key-1",
    "proofPurpose": "assertionMethod",
    "proofValue": "IU0iubIOUBiubIUOB...jQCrFPP2oumHKtz"
  }
]
```



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.

RISC
ZERO

```

      . . . : ^ ^ ~ ~ ~ ~ ~ ~ ~ ~ ^ ^ : . .
      : ^ ! 777 ! ~ ~ ~ ~ ~ ! ^ ^ : : ^ ^ ~ ~ ! ! 7 ! ~ ^ :
      : ! 77 ! ~ ~ : : ^ ? 7 J G G B & @ & 5      : : ^ ! 77 ! .
      : 7 P G J ^ ~ : Y P J 5 J 7 J B @ @ @ @ @ # :      : Y B P 7 :
      . ! P & @ @ 5 J P ? 5 G G P 5 .   7 @ @ @ @ @ B ^      : ? J ~ ^ : J @ @ & P !
      . ! Y @ @ @ @ @ @ # 5 P G 5 J B @ Y   . P @ @ @ & G G J ! ~ ~ .   . B @ & B @ 5 ! & @ @ @ @ B 7 .
      . 7 J J P @ @ @ @ @ @ # @ P ! ! Y B G P   7 @ @ @ 5 ^   . ! ? 7 .   ? @ @ G : # & & @ @ @ @ @ @ @ # 7
      ~ Y ~ 7 @ @ @ @ @ @ G ? . : ! G P ? P ~   . Y B 5      ! ! 7 P 5 @ 5 . 7 @ @ @ @ @ @ @ @ @ @ @
      ? J . 7 @ @ @ @ @ @ & ~   G @ @ Y G ^   .   ! G B P ^   7 & G J # @ @ @ @ @ @ @ @ @ @ @
      . Y 7   Y @ @ @ @ @ @ @ @ @ @ Y ~ G @ @ @ @ @ G !   ~ 7 J & G P @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
      5 ! ^ B @ @ @ @ @ @ @ @ @ @ & @ @ @ & B B G # ^   ~ G @ @ @ @ @ & @ @ @ @ G J Y & & 7 #
      Y 7   G @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ & J ! ! 5 Y :   ! B P # @ P ? ! ? Y P G @ # Y ? ? 5 @ B ~
      7 Y   ^ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ B ! ^ 7 : .   7 B & # 5 ^ ^ 7 ! ~ ! : P ! Y # # @ @ B
      . P .   ? @ @ @ @ @ @ @ @ @ @ @ @ P ! :   . ? # B # @ @ @ @ P 7 : ! ! : : J @ @ @
      7 J   Y @ @ @ @ @ @ @ @ @ @ @ B Y .   . 5 @ @ @ @ @ @ @ @ @ @ @ & @ @ @ & P B @ @
      P ~   7 5 @ @ @ 5 J 5 J # P   . Y & @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ & Y G @
      . G .   ^ @ @ P   ^ ?   7 @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ 7 5
      . G .   Y @ # ~ ? P ^ ^ ~ J ? ^ ! : .   J @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ & 7
      . G .   ^ J G @ G ! .   . : ~ J 7 ^   7 # @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
      5 ~   . ^ Y @ ?   ^ ~ ^ .   : 5 & @ @ @ @ & B @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
      7 J   ~ Y Y 7 ? @ @ @ & B ! . .   ! ! ~ : ~ .   ~ 7 B @ @ @ @ @ @ @ @ @ @ @
      . G .   .   7 @ @ @ @ @ @ @ # B ?   G @ @ @ @ @ @ @ @ @ @ @ # !
      7 Y   . B @ @ @ @ @ @ @ @ @ @ @ 5 Y 7 ^ ^ .   : B @ @ @ @ @ @ @ @ @ ?
      Y 7   G @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ # J   G @ @ @ @ @ @ @ @ @ !
      5 !   . P @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ G ^   . P @ @ @ @ @ @ @ B 7 ~ G
      . Y 7   ~ # @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ ~   5 @ @ @ @ @ @ P   . P 7
      ? J .   : ^ J & @ @ @ @ @ @ @ @ @ @ @ & ~   Y @ @ @ @ & 7   . .
      ~ Y !   ? @ @ @ @ @ @ @ @ @ @ Y ! :   7 @ & G ? :   ~
      . 7 J ~   ^ & @ @ @ @ @ @ B :   : ^ :   ^ ? 7
      . 7 J ~ .   J @ @ @ @ & 5 ^   : ^ ? 7 .
      . ! ? 7 ^   . P @ @ G ~   : ! 7 ~ .
      : ! ? 7 ^ .   J @ B .   . ^ ! 7 ! :
      : ~ 77 ! ~ : .   ^ ? ? ^ :   : : ^ ! 7 ! ~ :
      : ^ ! ! 7 ! ! ! ~ ~ ^ : : : : : : ^ ^ ~ ~ ! ! ! ! ~ ^ :

```






```
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 ! ~ ~ ~ ~ ~ ! ^ ^ : : ^ ^ ~ ~ ! ! 7 ! ~ ^ :
      : ! 77 ! ~ ~ : : ^ ? 7 J G G B & @ & 5      : : ^ ! 77 ! .
      : 7 P G J ^ ~ : Y P J 5 J 7 J B @ @ @ @ @ # :      : Y B P 7 :
      . ! P & @ @ 5 J P ? 5 G G P 5 .      7 @ @ @ @ @ B ^      : ? J ~ ^ : J @ @ & P !
      . ! Y @ @ @ @ @ @ # 5 P G 5 J B @ Y . P @ @ @ & G G J ! ~ ~ .      . B @ & B @ 5 ! & @ @ @ @ @ B 7 .
      . 7 J J P @ @ @ @ @ @ # @ P ! ! Y B G P      7 @ @ @ 5 ^      . ! ? 7 .      ? @ @ G : # & & @ @ @ @ @ @ @ @ # 7
      ~ Y ~ 7 @ @ @ @ @ @ G ? . : ! G P ? P ~      . Y B 5      !!      7 P 5 @ 5 . 7 @ @ @ @ @ @ @ @ @ @ @ @
      ? J . 7 @ @ @ @ @ @ & ~      G @ @ Y G ^      .      ! G B P ^      7 & G J # @ @ @ @ @ @ @ @ @ @ @ @
      . Y 7      Y @ @ @ @ @ @ @ @ @ @ Y ~ G @ @ @ @ @ G !      ~ 7 J & G P @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
      5 ! ^ B @ @ @ @ @ @ @ @ @ @ & @ @ @ & B B G # ^      ~ G @ @ @ @ @ & @ @ @ @ G J Y & & 7 #
      Y 7      G @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ & J ! ! 5 Y :      ! B P # @ P ? ! ? Y P G @ # Y ? ? 5 @ B ~
      7 Y      ^ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ B ! ^ 7 : .      7 B & # 5 ^ ^ 7 ! ~ ! : P ! Y # # @ @ B
      . P .      ? @ @ @ @ @ @ @ @ @ @ @ @ P ! :      . ? # B # @ @ @ @ P 7 : ! ! : : J @ @ @
      7 J      Y @ @ @ @ @ @ @ @ @ @ @ B Y .      . 5 @ @ @ @ @ @ @ @ @ @ @ & @ @ @ & P B @ @
      P ~      7 5 @ @ @ 5 J 5 J # P      . Y & @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ & Y G @
      . G .      ^ @ @ P      ^ ?      7 @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ 7 5
      . G .      Y @ # ~ ? P ^ ^ ~ J ? ^ ! : .      J @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ 7
      . G .      ^ J G @ G ! .      . : ~ J 7 ^      7 # @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
      5 ~      . ^ Y @ ?      ^ ~ ^ .      : 5 & @ @ @ @ & B @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @
      7 J      ~ Y Y 7 ? @ @ @ & B ! . .      ! ~ : ~ .      ~ 7 B @ @ @ @ @ @ @ @ @ @ @ @
      . G .      .      7 @ @ @ @ @ @ @ # B ?      G @ @ @ @ @ @ @ @ @ @ # !
      7 Y      . B @ @ @ @ @ @ @ @ @ @ @ 5 Y 7 ^ ^ .      : B @ @ @ @ @ @ @ @ @ @ ?
      Y 7      G @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ # J      G @ @ @ @ @ @ @ @ @ @ !
      5 !      . P @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ G ^      . P @ @ @ @ @ @ @ B 7 ~ G
      . Y 7      ~ # @ @ @ @ @ @ @ @ @ @ @ @ @ @ @ ~      5 @ @ @ @ @ @ P      . P 7
      ? J .      : ^ J & @ @ @ @ @ @ @ @ @ @ @ & ~      Y @ @ @ @ & 7      . .
      ~ Y !      ? @ @ @ @ @ @ @ @ @ @ Y ! :      7 @ & G ? :      ~
      . 7 J ~      ^ & @ @ @ @ @ @ B :      : ^ :      ^ ? 7
      . 7 J ~ .      J @ @ @ @ & 5 ^      . ^ ? 7 .
      . ! ? 7 ^      . P @ @ G ~      : ! 7 ~ .
      : ! ? 7 ^ .      J @ B .      . ^ ! 7 ! :
      : ~ 77 ! ~ : .      ^ ? ? ^ :      . : ^ ! 7 ! ~ :
      : ^ ! ! 7 ! ! ! ~ ~ ^ : : : : : : : ^ ^ ~ ~ ! ! ! ! ~ ^ :

```

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!

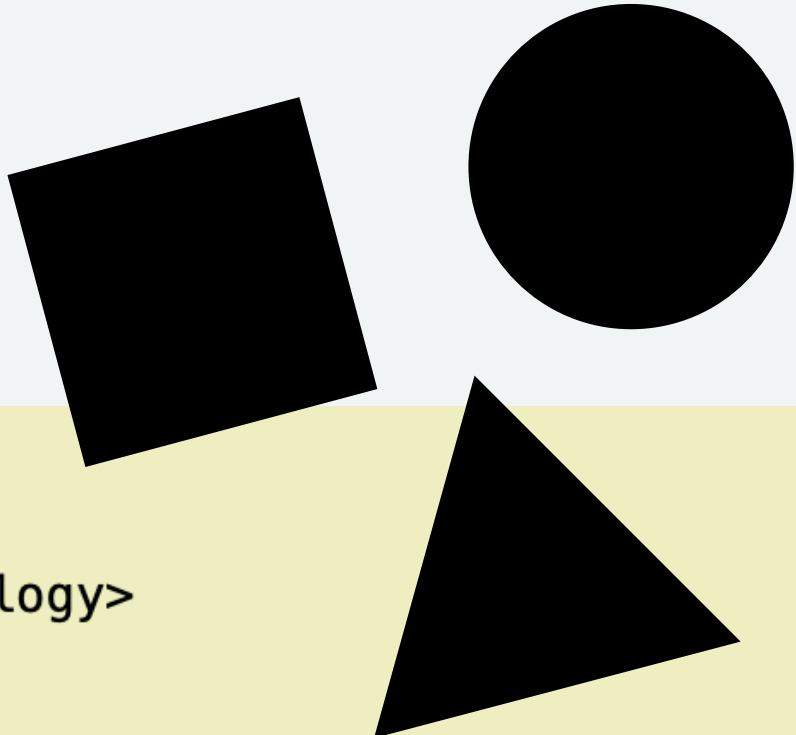


User abstractions != Standards abstractions

- 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>

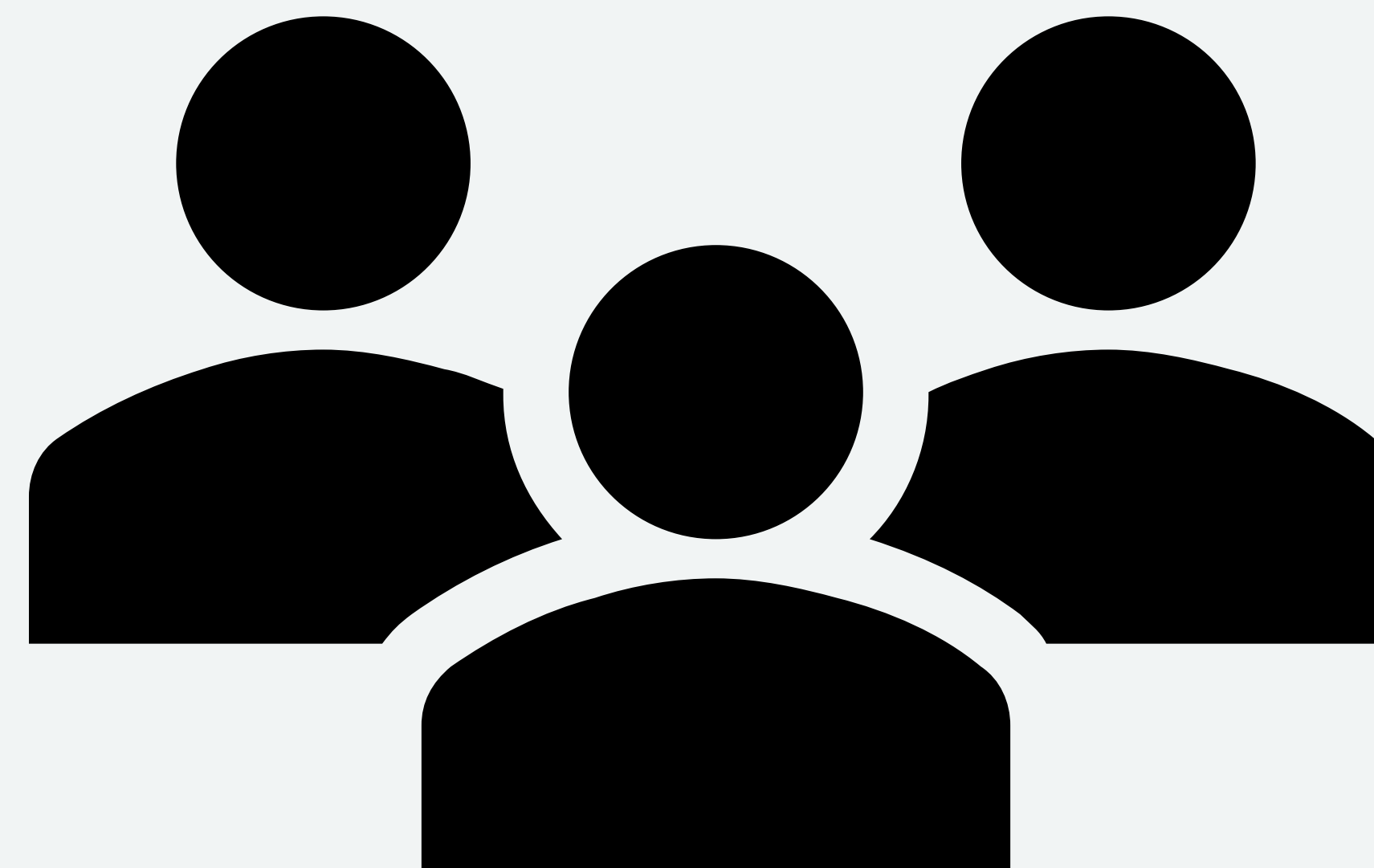
PREFIX ex: <http://example.com/ns#>

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

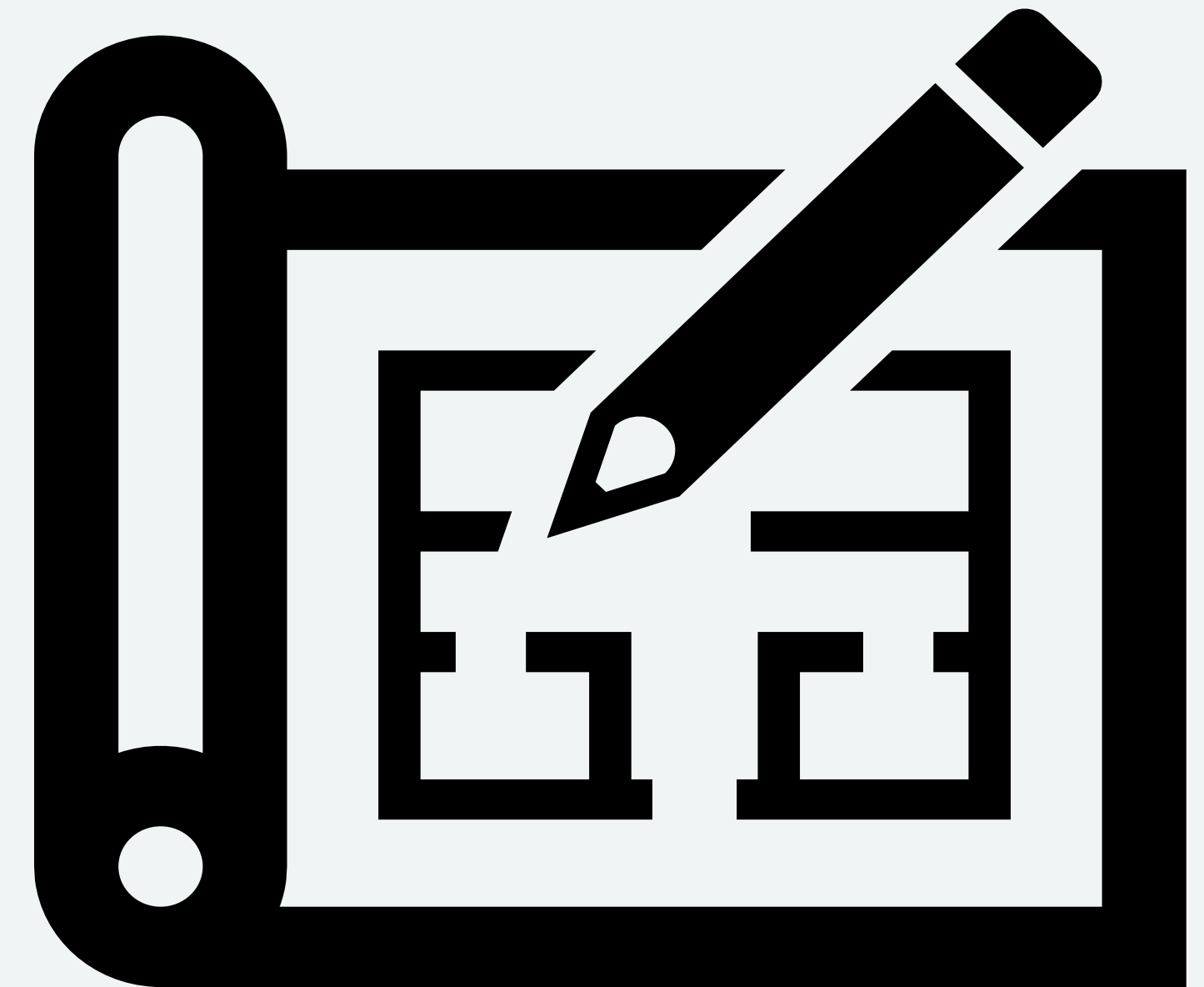
  ex:ssn      xsd:string [0..1] pattern="^\d{3}-\d{2}-\d{4}$" .
  ex:worksFor IRI ex:Company [0..*] .
  ex:address  BlankNode [0..1] {
    ex:city xsd:string [1..1] .
    ex:postalCode xsd:integer|xsd:string [1..1] maxLength=5 .
  } .
}
```

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"

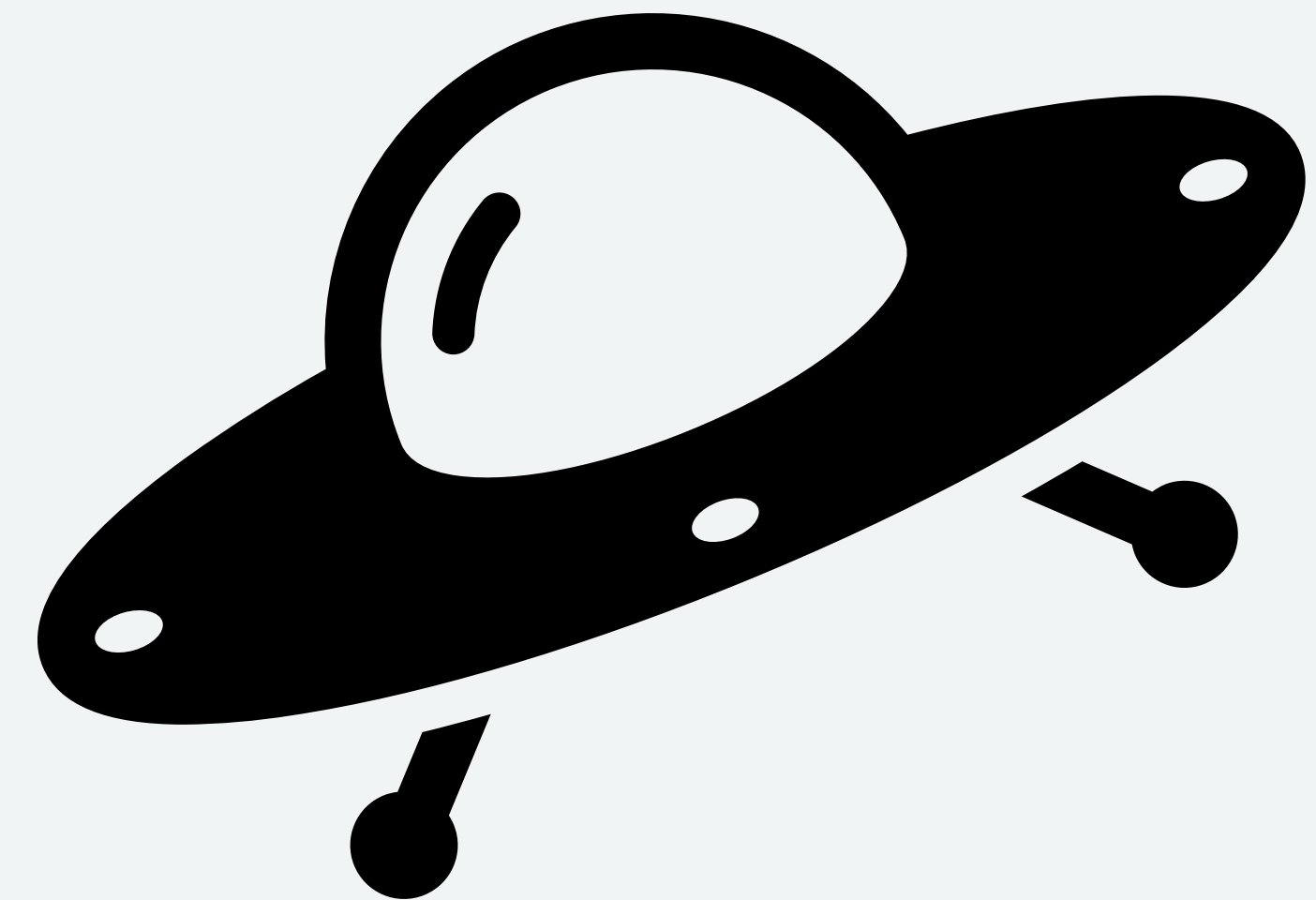


Call for use-cases



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/>
- <https://blog.jeswr.org/2025/02/14/data-wallets>



Questions

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

