ORACLE

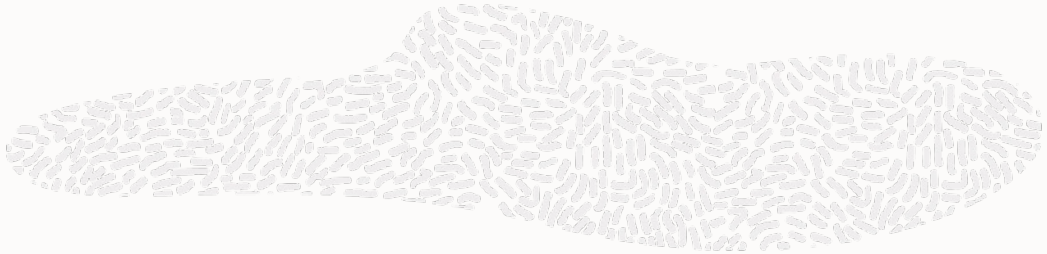
Towards an Abstraction for Verifiable Credentials and Zero Knowledge Proofs

April 2024

Mark Moir (mark.moir@oracle.com)
Architect
Oracle Labs

Joint work with Harold Carr (harold.carr@oracle.com)

About us



- Full-time
  - Harold Carr (https://www.linkedin.com/in/haroldcarr)
    - Blockchain fault tolerance and scalability, Infiniband transport, distributed systems, logic circuit simulation, …
  - Mark Moir (https://www.linkedin.com/in/markmoir)
    - Blockchain fault tolerance and scalability, synchronization primitives, concurrent algorithms, formal verification, …
- Former intern
  - Christoph Braun, Ph.D. student, Karlsruhe Institute of Technology
    - Contributed to use case demo and machinery to enable it

Agenda

1. “Elevator pitch” and research overview
2. Background and example use case
3. Motivation for and overview of our abstraction
4. Challenges and discussion
5. Ongoing work and summary

Agenda

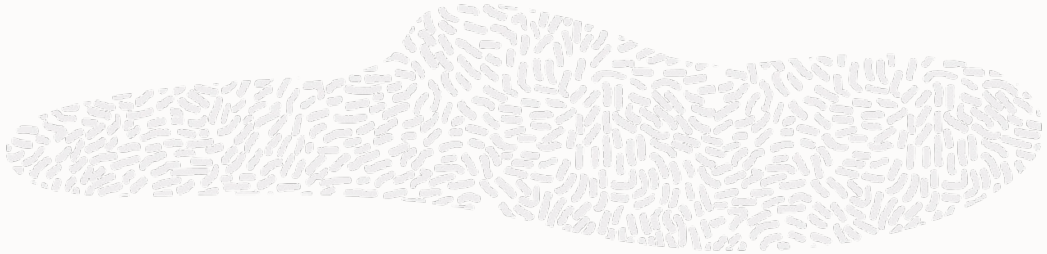
1. “Elevator pitch” and research overview
2. Background and example use case
3. Motivation for and overview of our abstraction
4. Challenges and discussion
5. Ongoing work and summary

“Elevator Pitch”: Using Verifiable Credentials and Zero Knowledge Proofs to balance privacy and accountability

- Collecting too much information creates liability, compliance and privacy issues
- Collecting too little precludes accountability
- Verifiable Credentials and Zero Knowledge Proofs can help to strike an effective balance
- But, many VC projects and standards don’t enable such use of ZKPs and/or are tied too tightly to specific cryptography libraries, limiting choice and progress

Icon attributions on last slide: 1, 2

Overview of our research



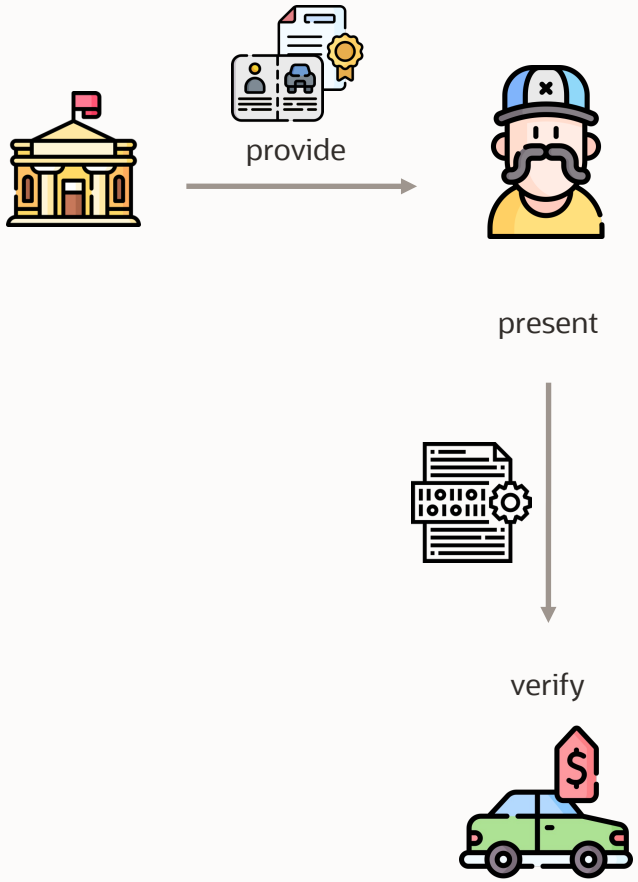
- We are:
  - Learning about these technologies, projects, standards efforts
  - Demonstrating (internally) potential of technologies
  - Developing an experimental abstraction to decouple VCs from underlying cryptography
- So far we have:
  - Defined an initial abstraction, embodied by Swagger/OpenAPI
  - Built a Docker container that serves the API to facilitate experimentation
  - Implemented the abstraction over DockNetwork crypto (https://github.com/docknetwork/crypto)
  - Developed a Car Share Service use case demo involving Driver's License and Monthly Subscription credentials, keeps hirer anonymous, enables identification by Authority if needed
- Currently continuing our work and:
  - Applying lessons learned, revising, generalizing and extending abstraction
  - Preparing to target abstraction to a second cryptography library
  - Sharing our experience and opinions so far, seeking feedback, engagement

Agenda

1. “Elevator pitch” and research overview
2. Background and example use case
3. Motivation for and overview of our abstraction
4. Challenges and discussion
5. Ongoing work and summary

Verifiable Credentials: basic roles

Issuer issues credential that includes attributes (e.g., about Holder)

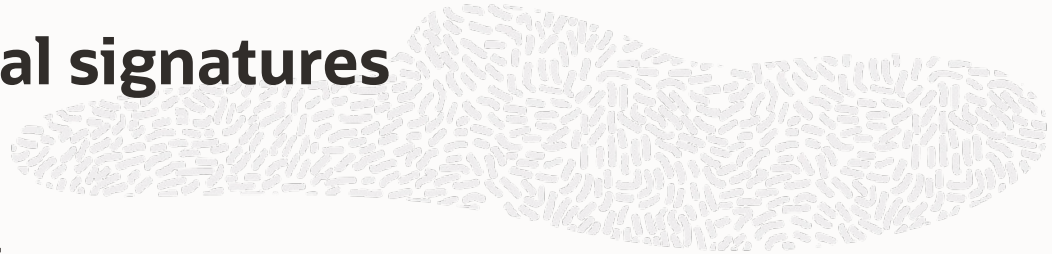


Holder presents "something" to Verifier

Verifier verifies the presented "something"

See last slide for icon attributions

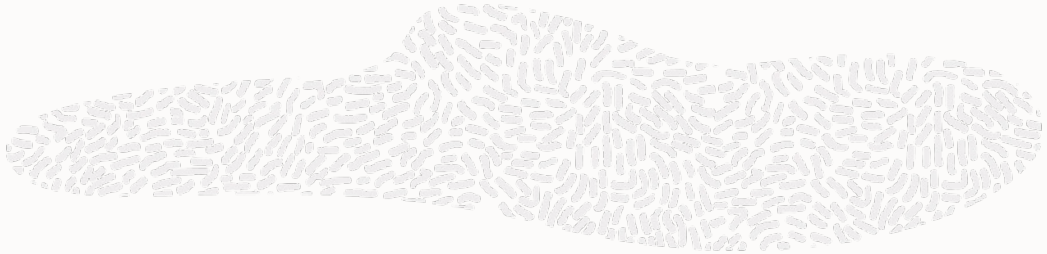
Strawman approach: Use traditional digital signatures



- Issuer signs message that includes attributes
- Holder presents message and signature to Verifier
- Verifier verifies signature

- Privacy implications
  - Holder must reveal entire message to enable verification
  - Holder presents same signature every time, enabling correlation, tracking, etc.
- Regulations, best practices require compliance with Data Minimization principle:
  - Collect only necessary information

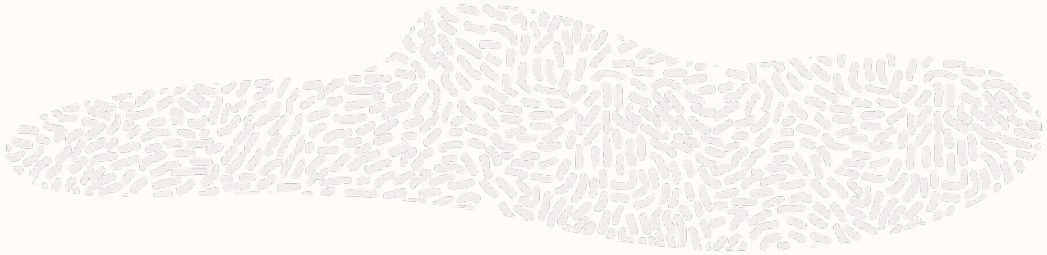
Using Zero Knowledge Proofs



- Issuer signs message that includes attributes
- Holder presents ~~message and signature~~ proof of knowledge of Issuer's signature
- Verifier verifies ~~signature~~ proof

- Zero Knowledge Proofs
  - Prove knowledge of something (e.g., Issuer's signature) *without* disclosing it
  - Different proof each time; prevents unwanted correlation
  - Reveal selected attributes, hide others
  - Prove properties about something without disclosing it, e.g.:
    - Predicates such as DOB > 20 years ago
    - Set (non)membership,
    - Many others

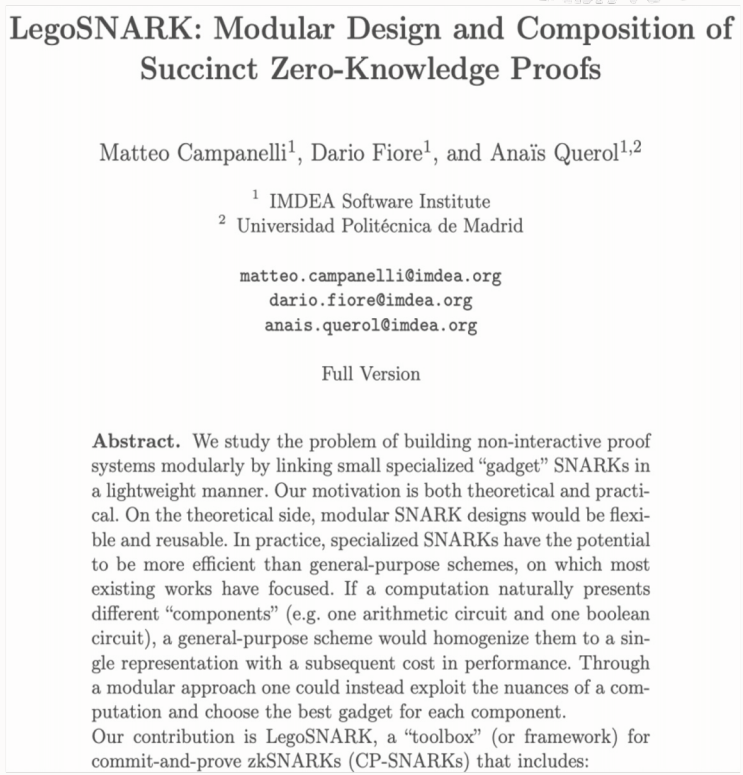
Zero Knowledge Proofs



	Zero-knowledge Proofs:	est. ca.
Privacy	• *Proof of Knowledge of Signature (selective disclosure)*	2004
	• *Range Proofs (range membership)*	2008
	• *Cryptographic Accumulators (set membership)*	2008
Accountability	• *Verifiable Encryption (encrypted disclosure)*	1998

Zero Knowledge Proofs

Commit-and-prove technology enables composing different ZKPs without compromising ZK



LegoSNARK: Modular Design and Composition of Succinct Zero-Knowledge Proofs

Matteo Campanelli[1], Dario Fiore[1], and Anaïs Querol[1,2]

[1] IMDEA Software Institute
[2] Universidad Politécnica de Madrid

matteo.campanelli@imdea.org
dario.fiore@imdea.org
anais.querol@imdea.org

Full Version

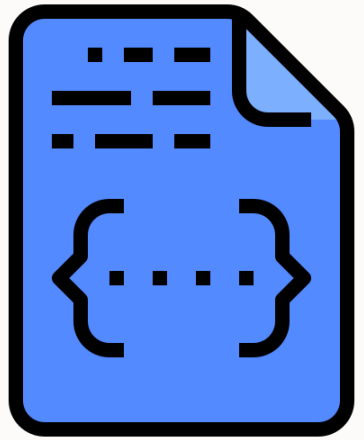
Abstract. We study the problem of building non-interactive proof systems modularly by linking small specialized "gadget" SNARKs in a lightweight manner. Our motivation is both theoretical and practical. On the theoretical side, modular SNARK designs would be flexible and reusable. In practice, specialized SNARKs have the potential to be more efficient than general-purpose schemes, on which most existing works have focused. If a computation naturally presents different "components" (e.g. one arithmetic circuit and one boolean circuit), a general-purpose scheme would homogenize them to a single representation with a subsequent cost in performance. Through a modular approach one could instead exploit the nuances of a computation and choose the best gadget for each component.
Our contribution is LegoSNARK, a "toolbox" (or framework) for commit-and-prove zkSNARKs (CP-SNARKs) that includes:

IACR Cryptol. ePrint Arch. 2019

	Zero-knowledge Proofs:	est. ca.
Privacy	• *Proof of Knowledge of Signature (selective disclosure)*	2004
	• *Range Proofs (range membership)*	2008
	• *Cryptographic Accumulators (set membership)*	2008
Accountability	• *Verifiable Encryption (encrypted disclosure)*	1998
	• ***and composites of those!***	**2016/2019 (in theory!)**

Implementation based on Commit-and-Prove, 2021-2024

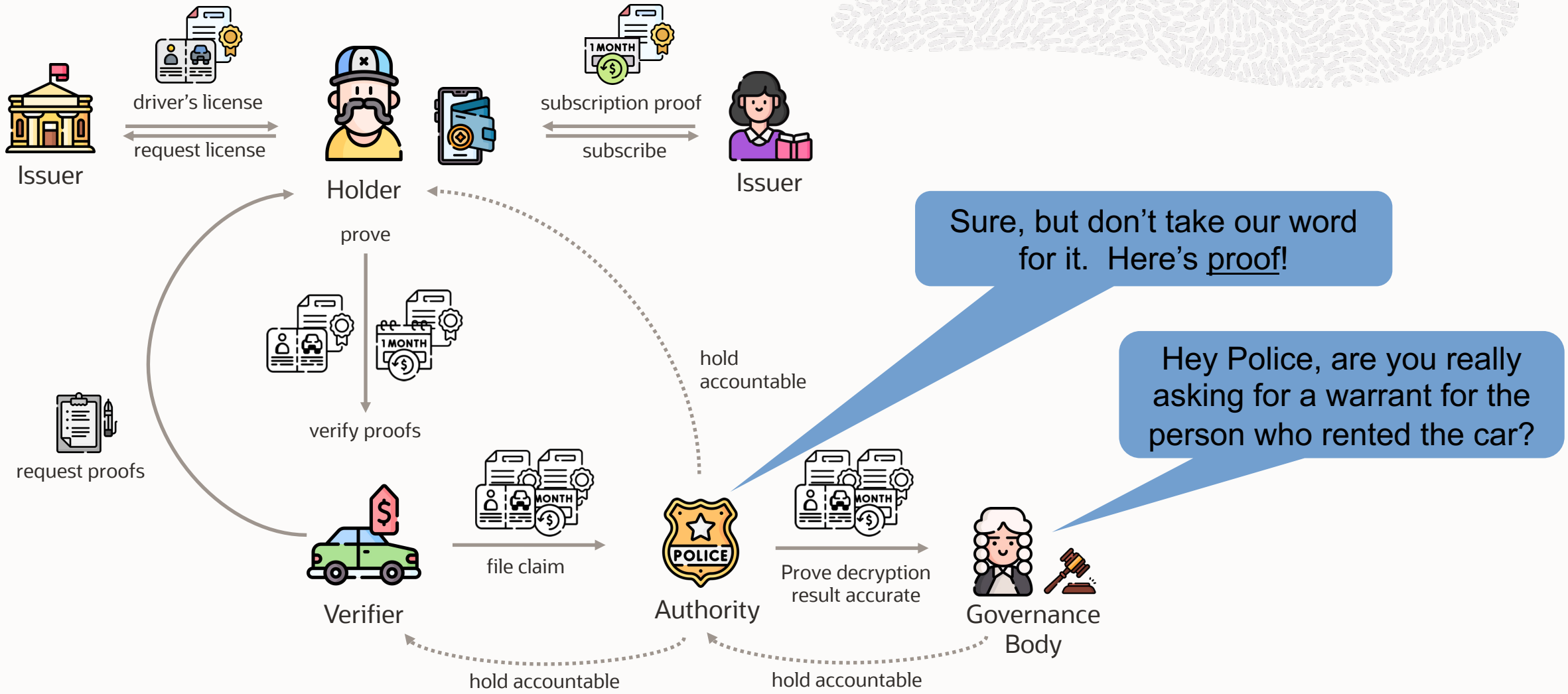
- Open source DockNetwork crypto library uses commit-and-prove to implement “proof system” that enables composing any/all of:
  - Efficient BBS+ signatures, with Selective Disclosure
  - Range proofs (e.g., DOB > 20 years ago)
  - Cryptographic accumulators (supporting privacy-preserving revocation, for example)
  - Other ZKPs (any zk-SNARK that can be expressed in R1CS, e.g., created using Circom)
  - Verifiable encryption
  - More (secret sharing, distributed key generation, …)



DockNetwork crypto (DNC) library

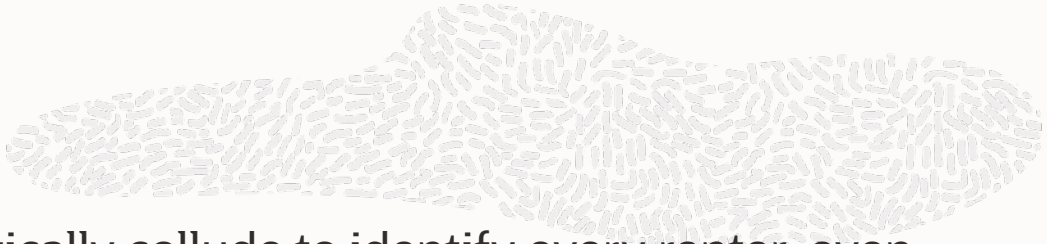
See last slide for icon attributions (3)

Example use case - Accountability in privacy-preserving authentication for services



Icons by Freepik from flaticon.com

Accountability variations

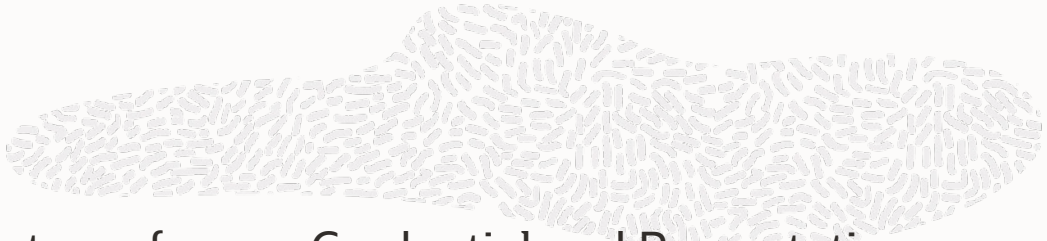


- As described so far, Authority and Verifier could theoretically collude to identify every renter, even well-behaved ones
- Could reverse roles, so Authority (e.g. Police) must request Governance Body (e.g. Court) to identify renter
- DockNetwork crypto library also supports secret sharing (though our work does not yet address it)
  - Could require *both* Authority *and* Governance Body to cooperate with Verifier
  - Could require *k* of *t* parties to identify renter

Agenda

1. “Elevator pitch” and research overview
2. Background and example use case
3. Motivation for and overview of our abstraction
4. Challenges and discussion
5. Ongoing work and summary

Motivation for abstraction

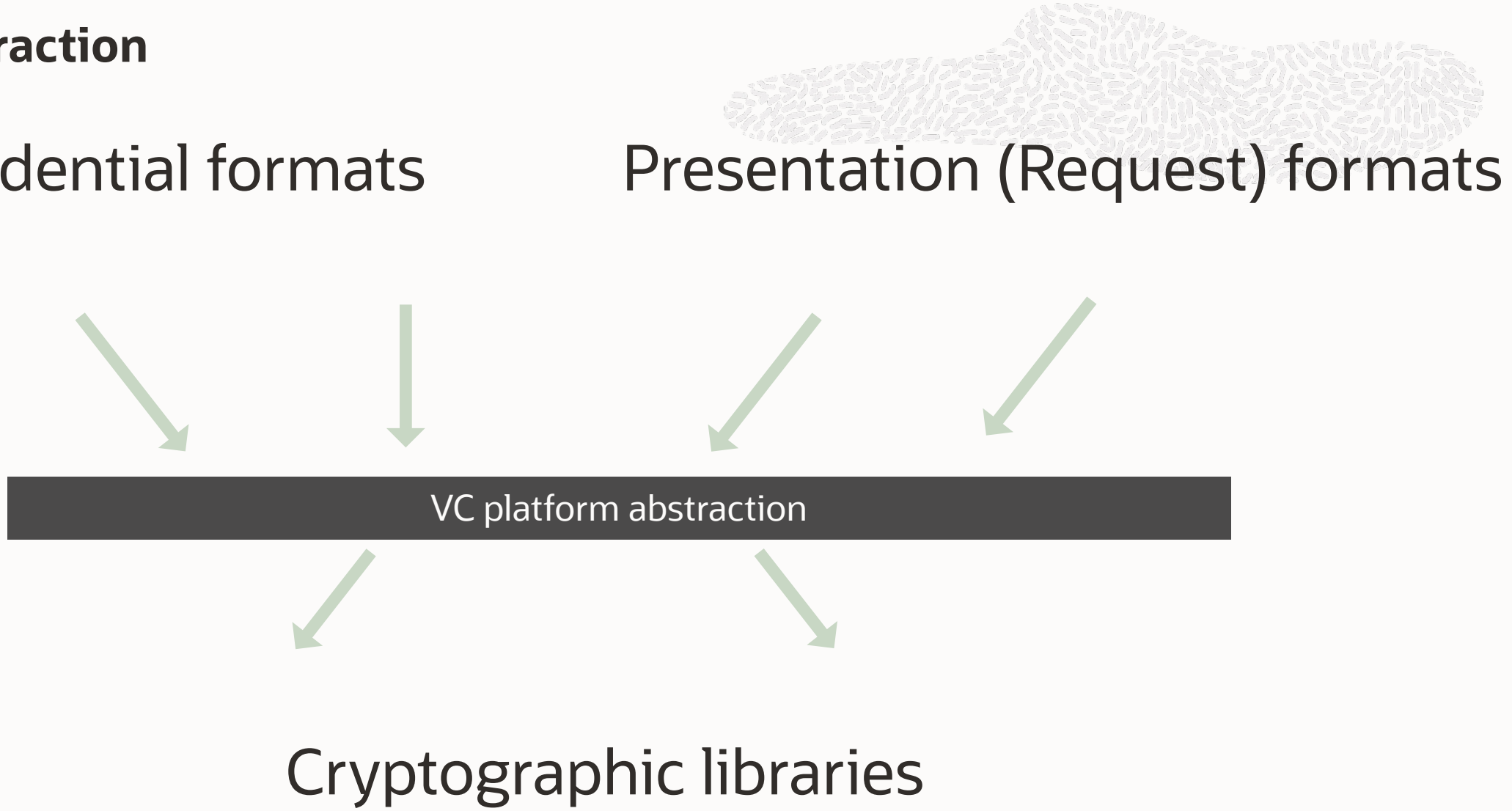


- Facilitate providing strong privacy and accountability features for any Credential and Presentation (Request) formats, using same underlying cryptography library
- Facilitate adoption of new and/or improved cryptography libraries by any Credential and Presentation (Request) formats
- Enable evolution of Credential and/or Presentation (Request) formats and/or cryptography libraries, relatively independently of each other
- Facilitate experimentation with and evaluation of different formats, libraries, etc.
- Reduce risk for projects and products that aim to effectively balance privacy and accountability
- …

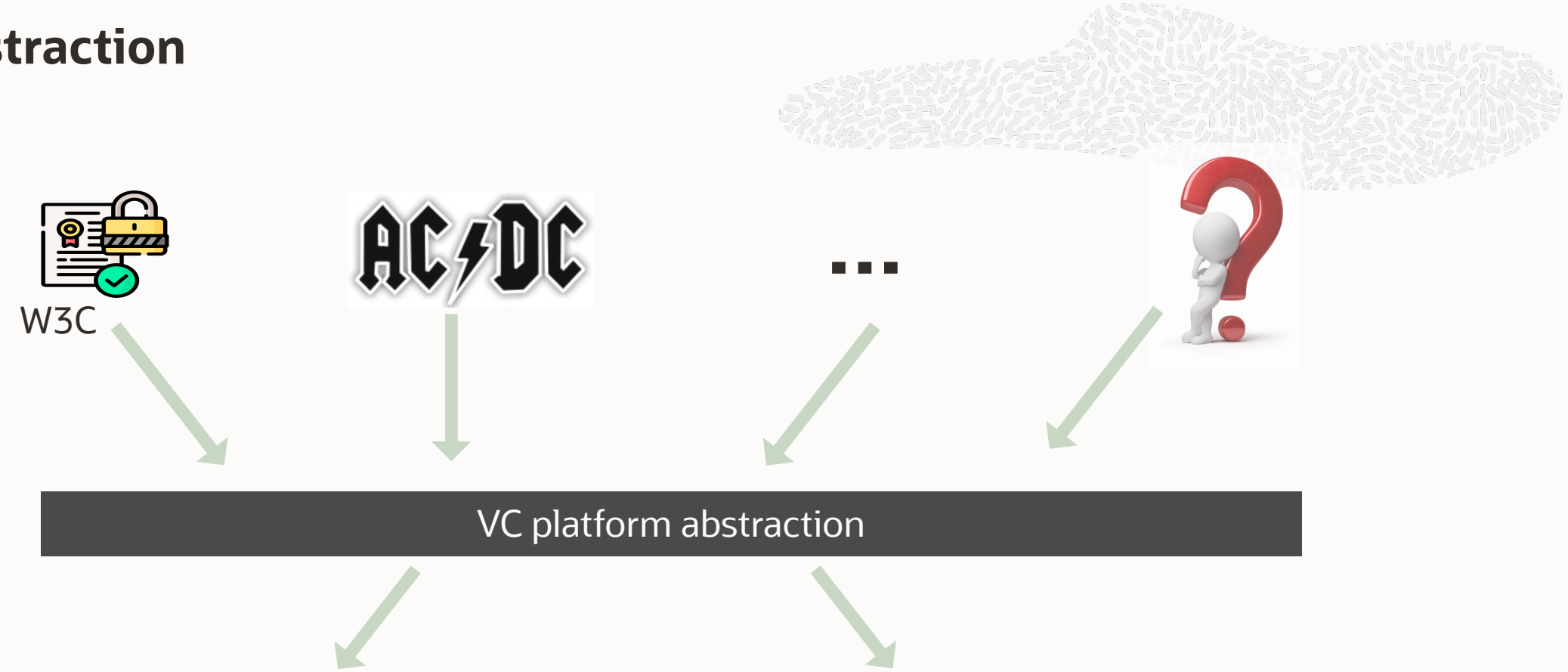
Some potential benefits

- Facilitate supporting privacy and accountability features for a credential format, without requiring low-level knowledge and expertise of underlying cryptography
- Enable people (not just developers with expertise to use specialized cryptography libraries) to more easily express/understand/analyze/audit use case requirements in different ways
- Enable reports/summaries/warnings for use case requirements in various formats, levels of detail
  - Example: User requests to disclose a verifiably encrypted attribute: likely a mistake. Conceivably intentional, so we allow it, but could generate a warning in a summary of requirements
  - Similarly for revealing accumulator members (thus leaking correlatable info)
- Facilitate switching underlying cryptography libraries for various reasons:
  - Performance (comparison), features, stronger security guarantees (e.g., post-quantum), more rigorous proofs, more favourable license, health of project, …
- Could enable formal proofs of privacy properties about use cases, based on assumptions about guarantees made by underlying cryptography (stated formally, perhaps proved formally)
- …

Our abstraction

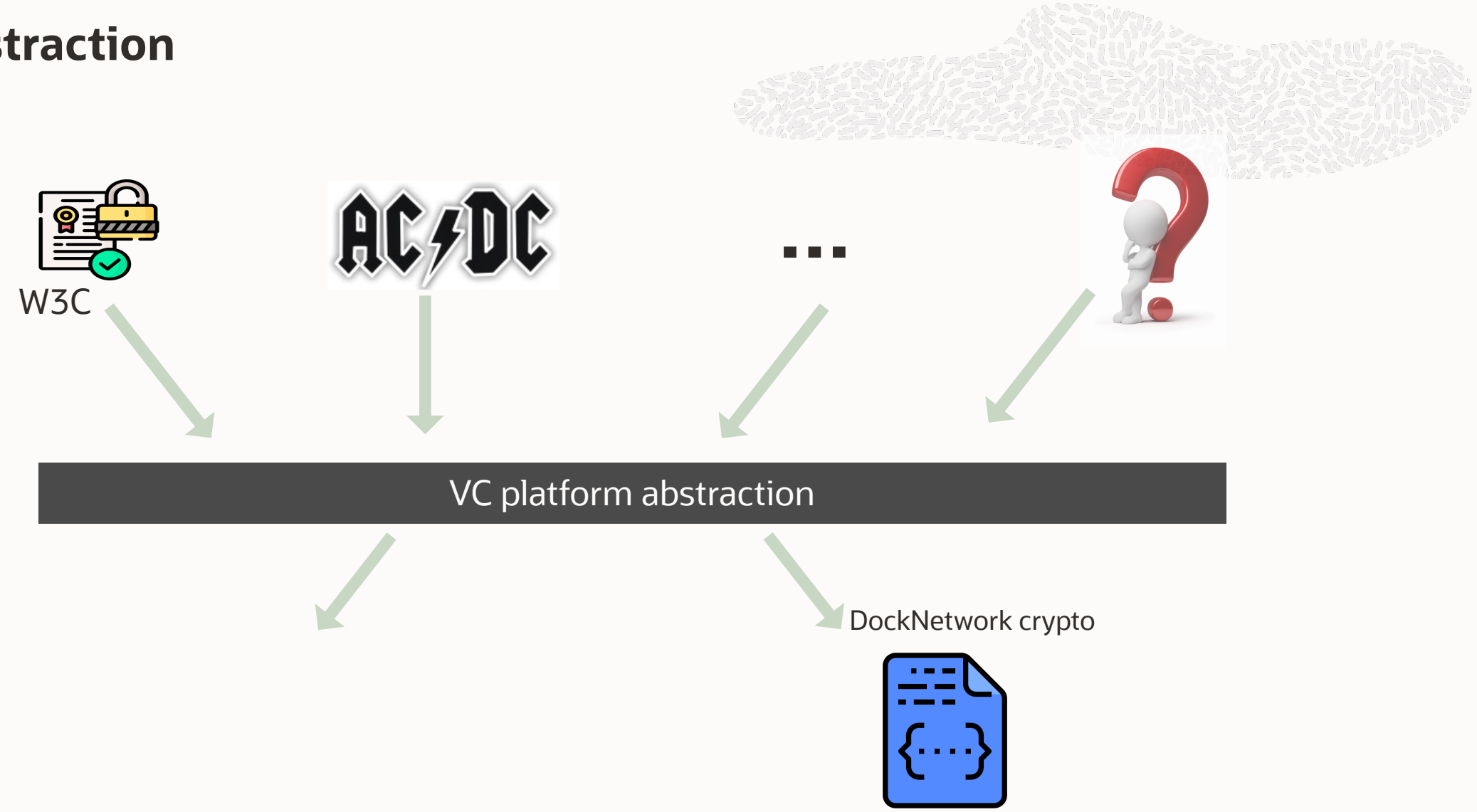


Our abstraction



See last slide for icon attributions (4)

Our abstraction



See last slide for icon attributions (3,4)

Our abstraction

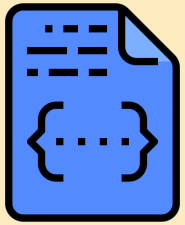
W3C

AC/DC

...

VC platform abstraction

DockNetwork crypto



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

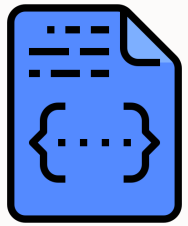
W3C

AC/DC

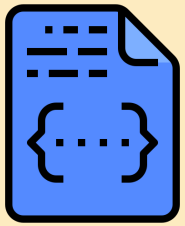
...

VC platform abstraction

Hyperledger Ursa



DockNetwork crypto



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

W3C

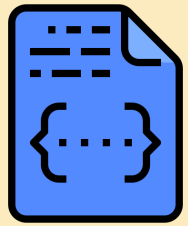
AC/DC

...

VC platform abstraction

Hyperledger Ursa

DockNetwork crypto



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

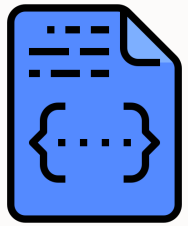
W3C

AC/DC

...

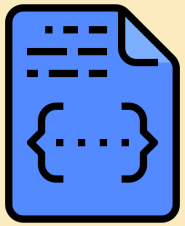
VC platform abstraction

Hyperledger Ursa



Hyperledger AnonCreds

DockNetwork crypto



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

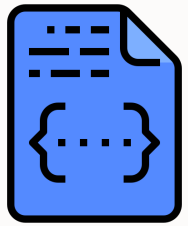
W3C

AC/DC

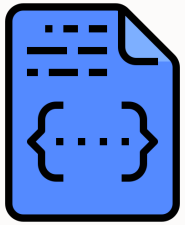
...

VC platform abstraction

Hyperledger Ursa

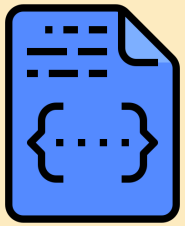


Hyperledger AnonCreds



Hyperledger AnonCreds 2.0

DockNetwork crypto



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

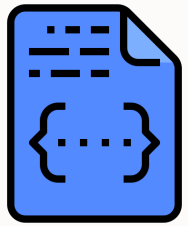
W3C

AC/DC

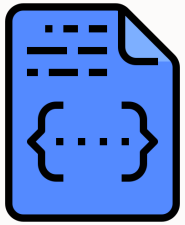
...

VC platform abstraction

Hyperledger Ursa



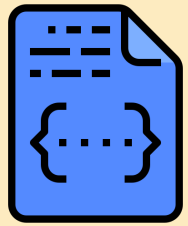
Hyperledger AnonCreds



Hyperledger AnonCreds 2.0

...

DockNetwork crypto



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Our abstraction

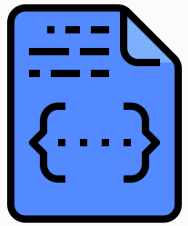
W3C

AC/DC

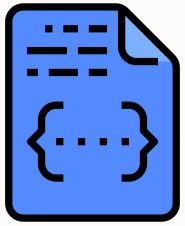
...

VC platform abstraction

Hyperledger Ursa



Hyperledger AnonCreds

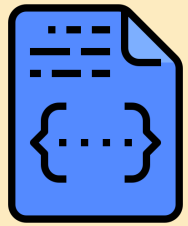


Hyperledger AnonCreds 2.0

...

Post-quantum?

DockNetwork crypto

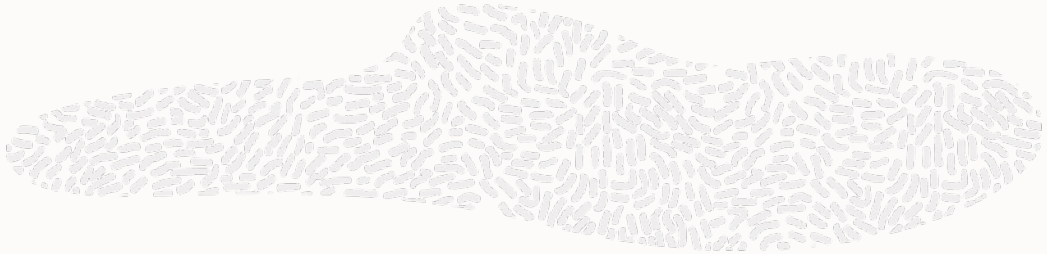


Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Actually, two related abstractions



- “Crypto Interface”
  - Defines minimal requirements that cryptography library must implement
- “Platform API”
  - Provides abstracted access to underlying cryptographic functionality as well as “utility” functionality that can be implemented using it, while remaining independent of Credential and Proof Request formats

Separate minimal requirements from useful higher-level functionality

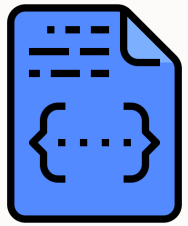
...

W3C

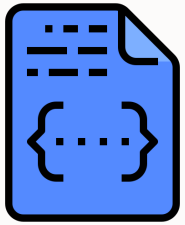
Platform API

Crypto Interface

Hyperledger Ursa



Hyperledger AnonCreds

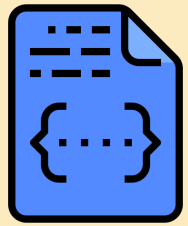


Hyperledger AnonCreds 2.0

...

Post-quantum?

DockNetwork crypto



Our focus so far:

- Simple cred format
- Advanced features via canonical formats
- Enabling and demonstrating use cases

See last slide for icon attributions (3,4)

Example of “utility” functions Platform API can provide using any library that implements Crypto Interface

Crypto Interface

```
CreateSignerData
  :: Natural          -- RNG seed
  -> [ ClaimType ]    -- "schema"
  -> SignerData

CreateAccumulator
  :: Natural
  -> AccumulatorData
```

Platform API

```
CreateAccumulators
  :: Natural
  -> [ ClaimType ]
  -> Map CredAttrIndex AccumulatorData

SetupIssuer
  :: Natural
  -> [ ClaimType ]
  -> (SignerData
     ,Map CredAttrIndex AccumulatorData)
```

Abstracting types via “OpaqueMaterial”

```
-- SignerData has internal structure, exposed by the abstraction

data SignerData = SignerData
  { signerPublicData :: SignerPublicData
  , signerSecretData :: SignerSecretData
  }

-- Components need only be passed back to interface (e.g., to sign or verify
-- a signature); hides library-specific implementation detail, ensuring same
-- interface can be implemented by multiple cryptography libraries

data SignerPublicData = SignerPublicData OpaqueMaterial
data SignerSecretData = SignerSecretData OpaqueMaterial
```

For concreteness, a simple, artificial credential format

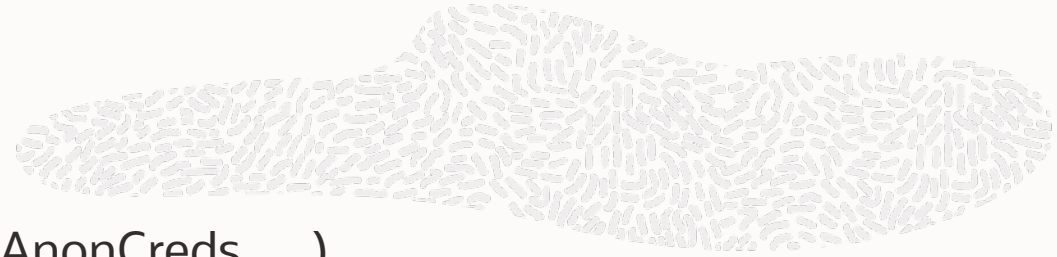
- We think of a “Credential Format Designer” role. CFD decides on format, rules, policy, etc. for what can be in a credential. Represents role of W3C, AnonCreds, etc.
- CFD defines how to map their credential format to the abstraction
- To enable experimentation and demonstration, we played this role by defining a simple Credential format (similar to JWT, but requires metadata that is signed as part of credential)

Example driver's license credential in our simple format

```
{ "metaDataForSimpleFormat" : {
      "purpose" : "DriverLicense",
      "version" : "1.0"
  },
  "sdAttrsForSimpleFormat": {
      "ssn":                    "123-12-1234",
      "eyeColor":               "Brown",
      "daysBornAfterJan_1_1900": 37852,
      "height":                 180,
      "idForRev":               "abcdef0123456789abcdef0123456789"
  }
}
```

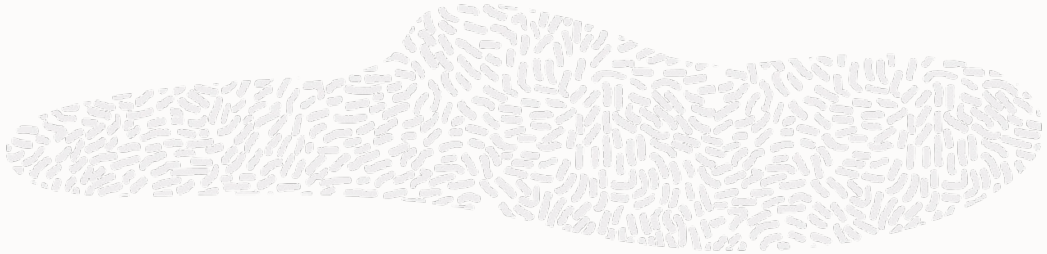
"Credential Format Designer" requires a metadata attribute represented as a JSON object, and a flat list of named values (String or Int).

The abstraction in more detail (1/2)



- Abstraction is agnostic to credential format (e.g., W3C, AnonCreds, …)
- Various “opaque types” that user receives from interface and sends back (perhaps indirectly), with examples of included information when implemented over DockNetwork crypto library:
  - **`SignerPublicData`**
    - Public key, signature parameters
  - **`SignerSecretData`**
    - Secret key
  - **`DataForVerifier`**
    - Proof, values of revealed attributes
  - **`AuthorityPublicData`**
    - Chunk size, commitment generators, encryption generators, encryption key, Groth16ProvingKey, …
  - **`AuthoritySecretData`**
    - Groth16SecretKey
  - **`AuthorityDecryptionKey`**
    - Groth16DecryptionKey
  - …

The abstraction in more detail (2/2)



- Data formats defined by abstraction
  - **Claim Types** (`CTText`, `CTNat`, `CDAccumulatorMember`, or `CTEncryptableText`)
    - Describe attributes (in same order as Values)
    - `CDEncryptableText` enables special encoding required for decryption
  - **Data values** (`DVText` or `DVNat`)
    - Represent values in credential in some defined order
  - **ProofRequest** (a.k.a Presentation Request, etc.)
    - Defines what disclosures and properties Holder and Verifier agree on.
  - **SharedParams**
    - `ProofRequest` refers to parameter values symbolically, `SharedParams` provides values
    - Makes `ProofRequest` more readable, reusable with different parameters
  - **DecryptRequest** (for Verifiable Encryption)
    - Identifies credential and attribute to be decrypted, contains `AuthoritySecretData` and `AuthorityDecryptionKey`
  - **DecryptResponse**
    - Decrypted value, proof that it's correct

Shared Params for example driver license and subscription use case

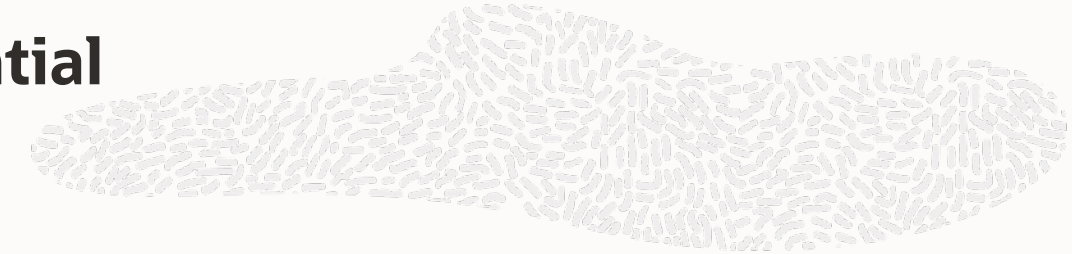
```
{ "dlIssuer"          : "eyJzcHNkU2lnUGFyYW1zIjoie1wiZzFcIjp...zUsNTBdIn0="
, "authorityPubData": "more opaque stuff"
, "maxBDdays"         : 999999999999
, "minBDdays"         : 37696
, …
}
```

Example of “opqaue” data: different implementations may use different underlying cryptography, types, etc.

Values for example driver license credential


```
{"values":
  [ {"contents": "CredentialMetadata (fromList [(\"purpose\",DVText
\"DriverLicense\"),(\"version\",DVText \"1.0\")])",  "tag": "DVText"}
  , {"contents": 37852,                                "tag": "DVInt"}
  , {"contents": "Brown",                              "tag": "DVText"}
  , {"contents": 180,                                  "tag": "DVInt"}
  , {"contents": "abcdef0123456789abcdef0123456789", "tag": "DVText"}
  , {"contents": "123-12-1234",                        "tag": "DVText"}
  ]
}
```

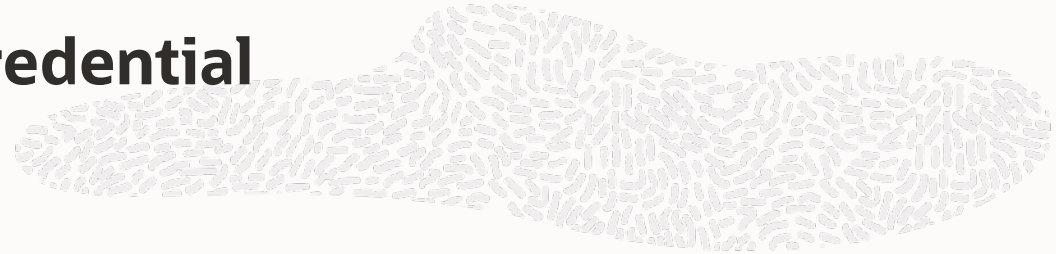
Values for example driver license credential



```
{"values":
  [ {"contents": "CredentialMetadata (fromList [(\"purpose\",DVText
\"DriverLicense\"),(\"version\",DVText \"1.0\")])",  "tag": "DVText"}
  , {"contents": 37852,                                "tag": "DVInt"}
  , {"contents": "Brown",                              "tag": "DVText"}
  , {"contents": 180,                                  "tag": "DVInt"}
  , {"contents": "abcdef0123456789abcdef0123456789", "tag": "DVText"}
  , {"contents": "123-12-1234",                        "tag": "DVText"}
  ]
}
```

Note: The simple credential format we defined for experimentation requires metadata, encoded into a `DVText` as the first value, i.e., the attribute with index 0 in the list. The structure, contents, and encoding of this value are determined by the "Credential Format Definer", independent of the abstraction.

Claim Types for example driver license credential



```
{”claimTypes":
  [ "CTText"
  , "CTNat"
  , "CTText"
  , "CTNat"
  , "CTAccumulatorMember"
  , "CTEncryptableText"
  ]
}
```

Claim Types for example driver license credential

```
{”claimTypes":
  [ "CTText"
  , "CTNat"
  , "CTText"
  , "CTNat"
  , "CTAccumulatorMember"
  , "CTEncryptableText"
  ]
}
```

Note:

- `CTEncryptableText` indicates to Issuer to sign this attribute (Social Security Number in our example) for verifiable encryption.
- `CTAccumulatorMember` indicates that the value is represents an element to be included in an accumulator

Proof Request for driver license and subscription use case (1/2)

```
{"driverLicense”:
  { ”signerPublicData”: "dlIssuer"
  , "disclosed"       : [0]
  , "inAccum"         : [[4, "dlCurrentAccum"]]
  , "notInAccum"      : []
  , "inRange"         : [[1, ["minBDdays", "maxBDdays", "provingKey"]]]
  , "encryptedFor”.   : [[5, "commonAuthorityPK"]]
  , "equalTo"         : [[5, ["subscriptionCred", 2]]]
  }
, …
```

Proof Request for driver license and subscription use case (1/2)

```
{"driverLicense”:
  { "signerPublicData": "dlIssuer"
  , "disclosed"       : [0]
  , "inAccum"         : [[4, "dlCurrentAccum"]]
  , "notInAccum"      : []
  , "inRange"         : [[1, ["minBDdays", "maxBDdays", "provingKey"]]]
  , "encryptedFor”    : [[5, "commonAuthorityPK"]]
  , "equalTo"         : [[5, ["subscriptionCred", 2]]]
  }
, …
```

Note:

- These are indices into the list of Values; metadata is 0, Social Security Number is 5.
- Internal debate about whether to allow/require readable labels. Not strictly required, but small burden to provide and useful for understanding and debugging

Proof Request for example driver license and subscription use case (2/2)

```
…
,"subscriptionCred":
  { "signerPublicData” : "subIssuer"
  , "disclosed"        : [0, 4]
  , "inAccum"          : [[1, "subCurrentAccum"]]
  , "notInAccum"       : []
  , "inRange"          : [[3, ["minValiddays", "maxValiddays", "provingKey"]]]
  , "encryptedFor"     : [[2, "commonAuthorityPK"]]
  , "equalTo"          : []
  }
}
```

DecryptRequests for example driver license and subscription use case

```
[ { "credLabel": "subscriptionCred"
  , "attrIndex": 2
  , "sk"       : "more opaque stuff"
  , "dk"       : "more opaque stuff"
  }
]
```

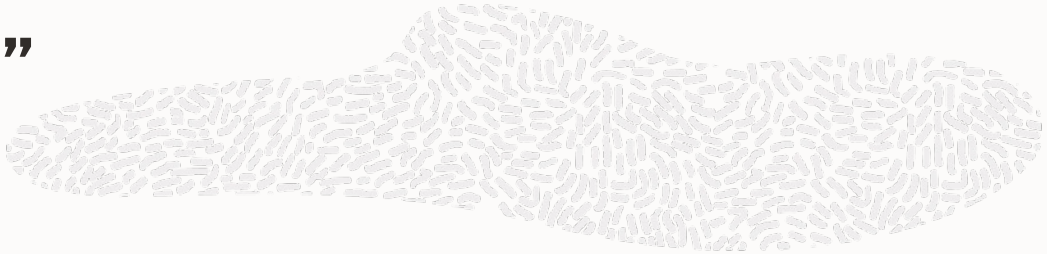
Agenda

1. “Elevator pitch” and research overview
2. Background and example use case
3. Motivation for and overview of our abstraction
4. Challenges and discussion
5. Ongoing work and summary

Abstraction challenges

- Different implementations may support different subsets of features
  - Warnings for unsupported features enable policy decision whether to accept
- Different implementations may make different assumptions and guarantees
  - Quantum-safe signature schemes vs. non
  - Extend abstraction to enable structured “documentation”?
- …

More abstract types via “OpaqueMaterial”



```
CreateProof
  :: Map CredentialLabel CredentialReqs
  -> Map SharedParamKey SharedParamValue
  -> Map CredentialLabel (Signature, CredAuxiliaryData)
  -> WarningsAndResult DataForVerifier

data DataForVerifier = DataForVerifier
  { revealedIdxsAndVals :: Map CredentialLabel [ CredAttrIndexAndDataValue ]
  , proof               :: Proof          -- OpaqueData
  }

VerifyProof
  :: Map CredentialLabel CredentialReqs
  -> Map SharedParamKey SharedParamValue
  -> Proof
  -> Map CredentialLabel (Map CredAttrIndex DecryptRequest)
  -> WarningsAndResult DecryptResponses   -- Exception if verification fails
```

Abstraction challenges

- Different implementations may support different subsets of features
  - Warnings for unsupported features enable policy decision whether to accept
- Different implementations may make different assumptions and guarantees
  - Quantum-safe signature schemes vs. non
  - Extend abstraction to enable structured “documentation”?
- Example challenge with OpaqueData approach
  - DockNetwork crypto’s createProof returns a single item, we serialise it to OpaqueData, Prover sends to Verifier, who passes it to verifyProof
  - In contrast, AnonCreds 2.0 Presentations include separate proofs for each Statement (proof of knowledge of signature, range proof, etc.)
    - Thus internal structure is visible “above the abstraction”
    - Can’t implement CryptoInterface without changing Presentation format?

Partial list of things we think should live above the abstraction

- VC format (W3C VCDM, AnonCreds, ACDC,…)
- Attribute names (if any), but recall “internal debate”
- Negotiation of presentation requirements
- Communication protocols
- VC contents
  - Example: questions such as whether credentials require metadata could/should be be separate from underlying cryptography
  - Similarly for link secrets
  - Similarly for accumulators (e.g., zero, one, or multiple revocation managers?)
- Rules, policies, roles
  - Example: AnonCreds v2 assumes Authority = Issuer
    - We don’t think this should be assumed (Authority could be Police, some other government entity, some legal entity, multiple entities, etc., determined by use case)
    - Regardless, such decisions should be separate from underlying cryptography
- …

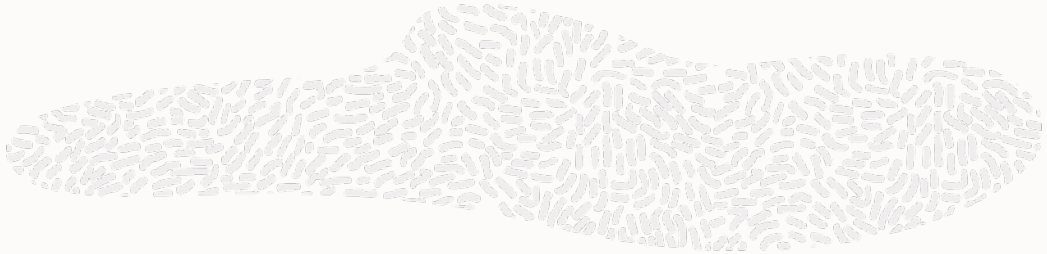
Agenda

1. “Elevator pitch” and research overview
2. Background and example use case
3. Motivation for and overview of our abstraction
4. Challenges and discussion
5. Ongoing work and summary

Ongoing work

- We continue to refine our abstraction and target it to DockNetwork crypto
- We are preparing to target our abstraction to the cryptography library underlying AnonCreds 2.0, testing the abstraction's generality and potentially facilitating direct use of underlying cryptogrsphy by other Credential formats
- For accumulators, our abstraction so far supports only positive accumulators with single managers. AnonCreds 2.0 aims to support ALLOSAUR, which enables scalable multi-party accumulator managers. How does the abstraction look for that?
- …

Summary



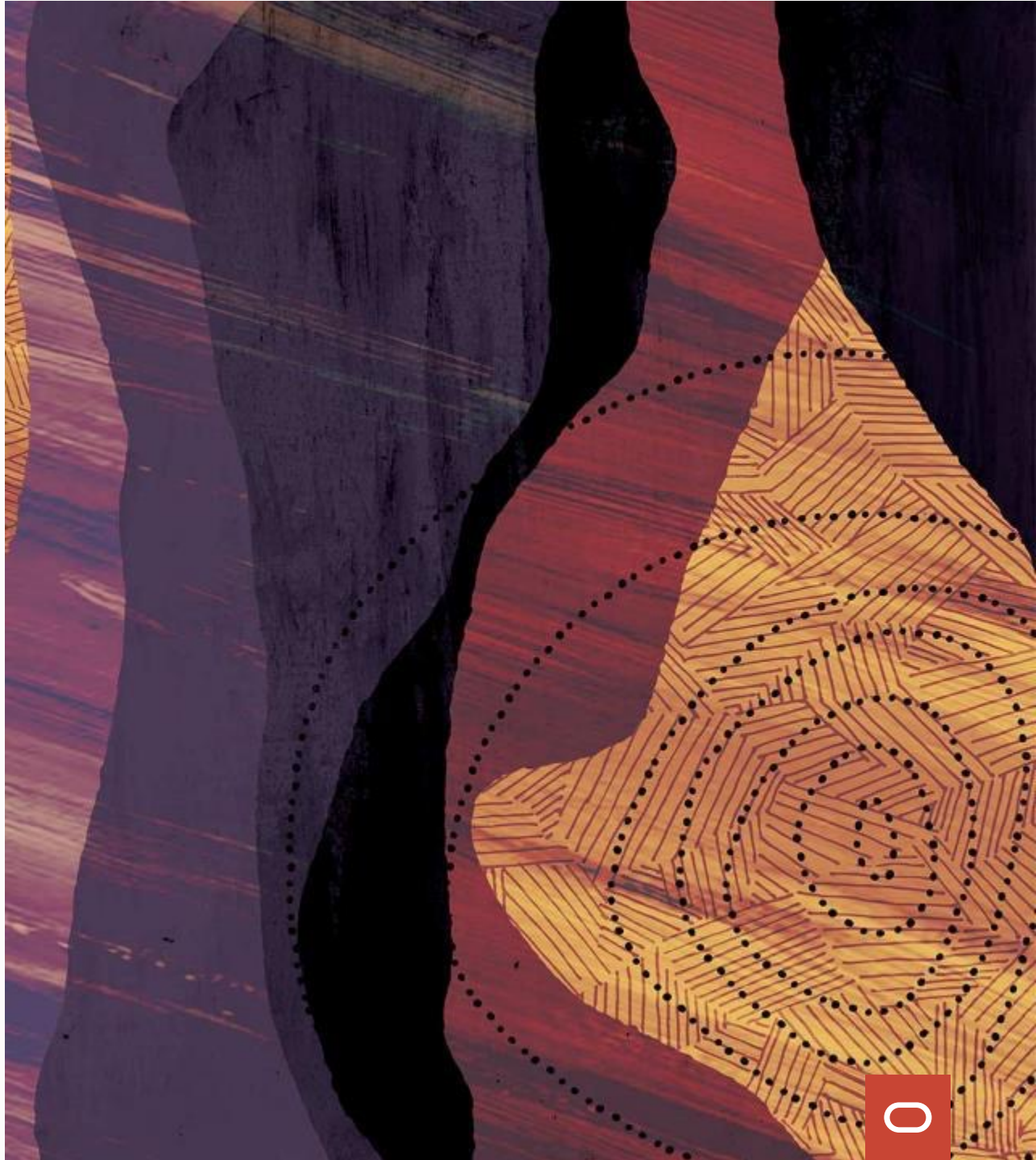
- We are:
  - Learning about technologies, projects, and standards efforts around Verifiable Credentials and Zero Knowledge Proofs
  - Demonstrating (internally) potential of these technologies
  - Developing an abstraction to decouple VCs from underlying cryptography
  - Preparing for open source contribution to enable our abstraction to benefit the ecosystem
- This has led us to some opinions and ideas that we think can be beneficial beyond our organisation
- Therefore we are sharing our experience and opinions so far, seeking feedback, engagement

Thank you

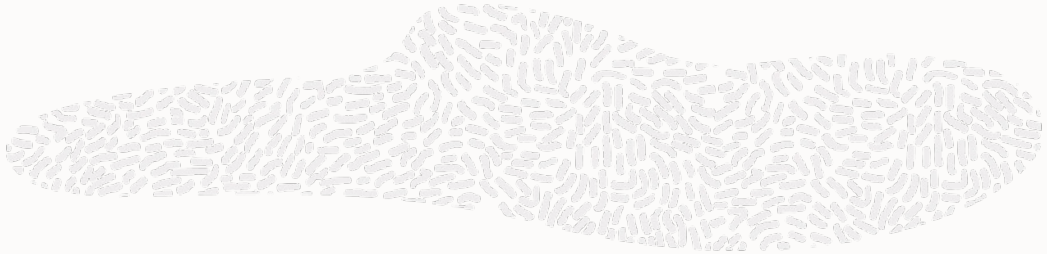
Mark Moir, Architect, Oracle Labs

mark.moir@oracle.com

www.linkedin.com/in/markmoir



Icon attributions



All icons by Freepik from flaticon.com, except:

1. Private message icons created by juicy_fish – Flaticon
2. Arrest icons created by shmai – Flaticon
3. Icon by ultimatearm
4. By Denmokin - Own work, Public Domain
5. Professional icons created by Uniconlabs - Flaticon