Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers

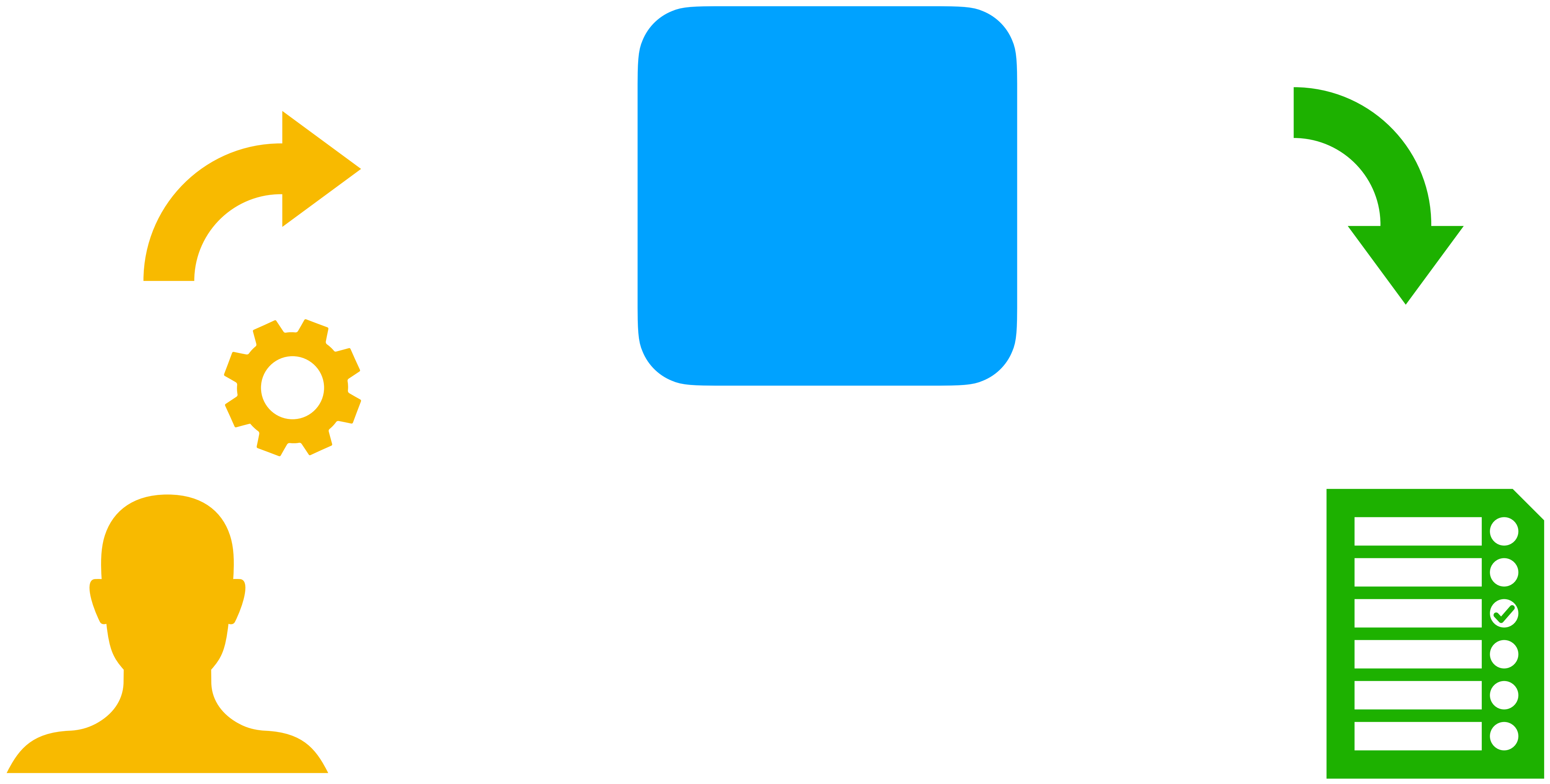
Authors
Alberto Sonnino*
Mustafa Al-Bassam*
Shehar Bano*
Sarah Meiklejohn*
George Danezis*

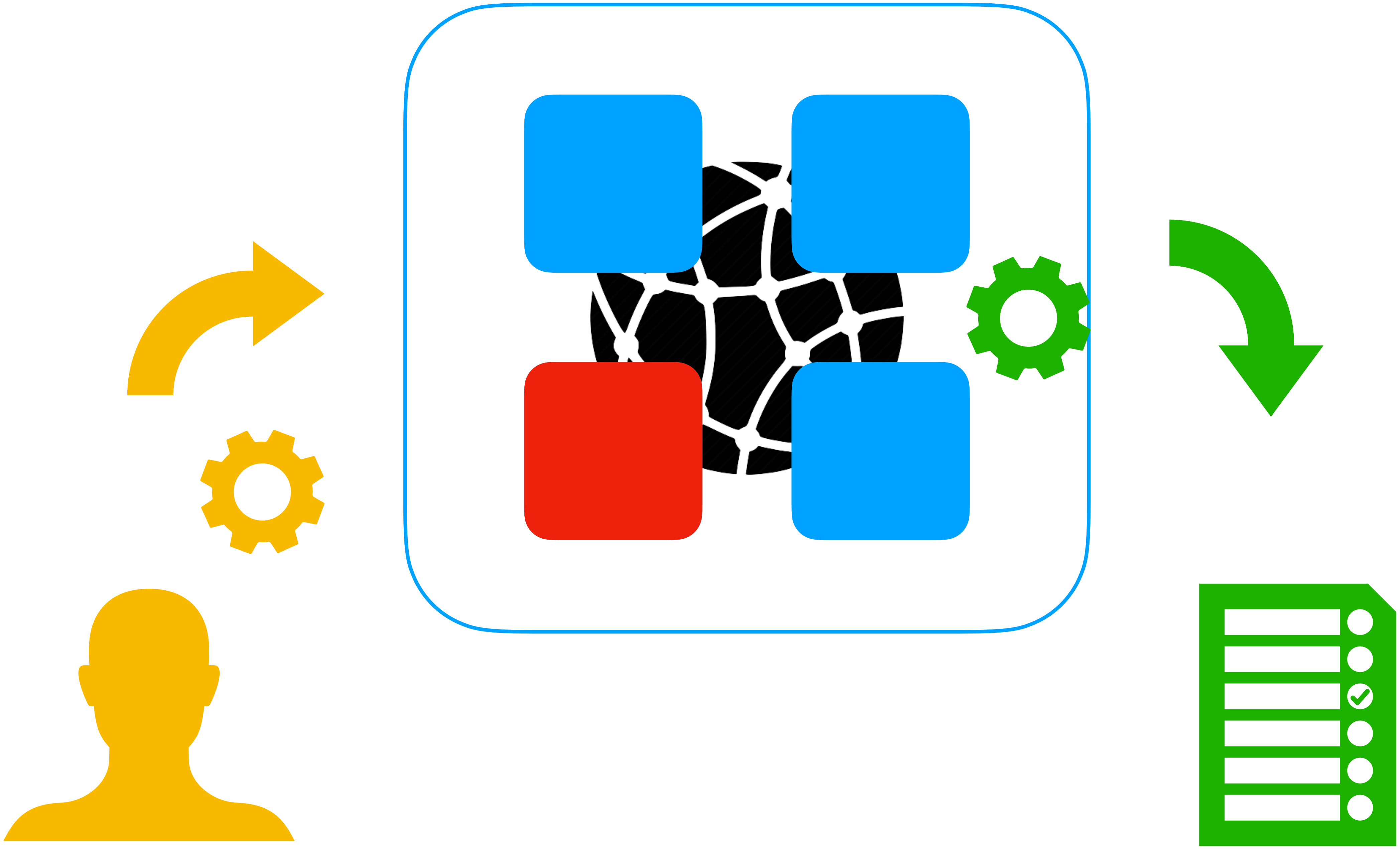
* University College London

Privacy-preserving credentials

...without a single issuer

Blockchains



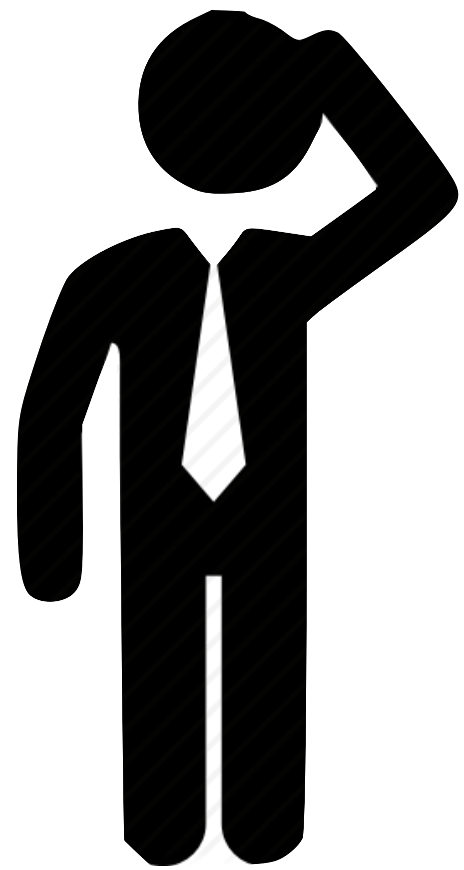


UCL

Byzantine

Hard to build

Expensive



No failure

Conventional

Cheap

UCL


betway
TO BOOK

The Authors

Alberto Sonnino

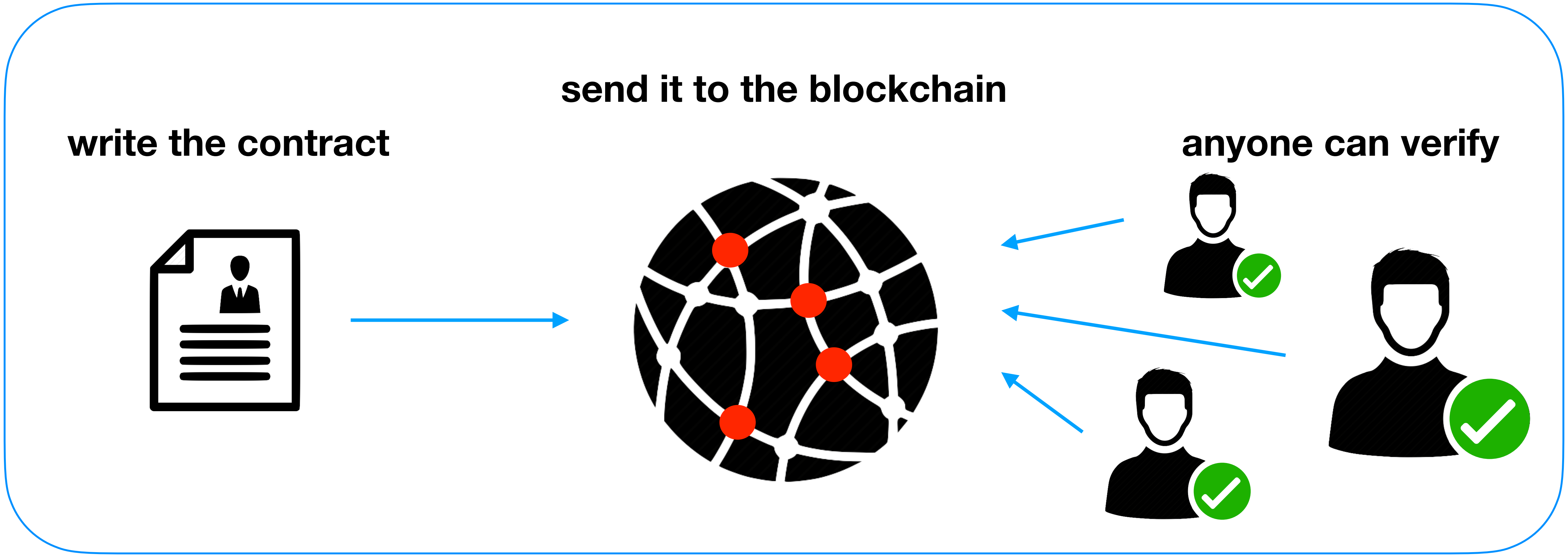
Mustafa Al-Bassam

Bano Shehar

Sarah Meiklejohn

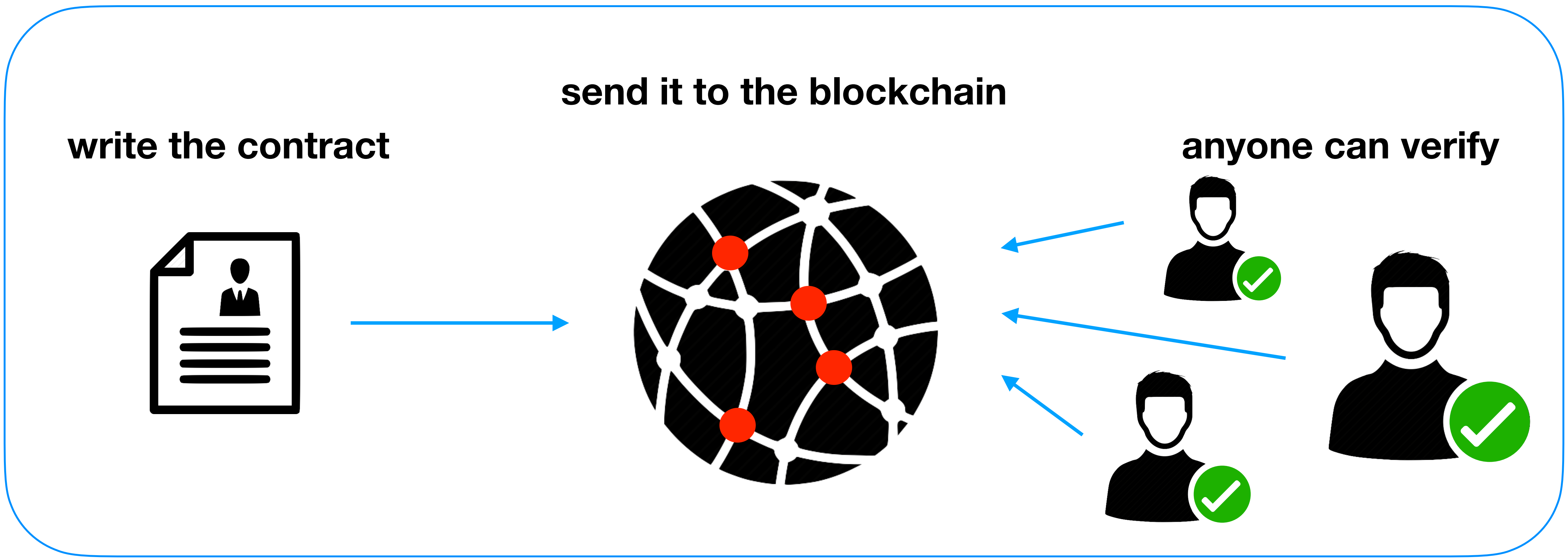
George Danezis

Challenges in blockchains



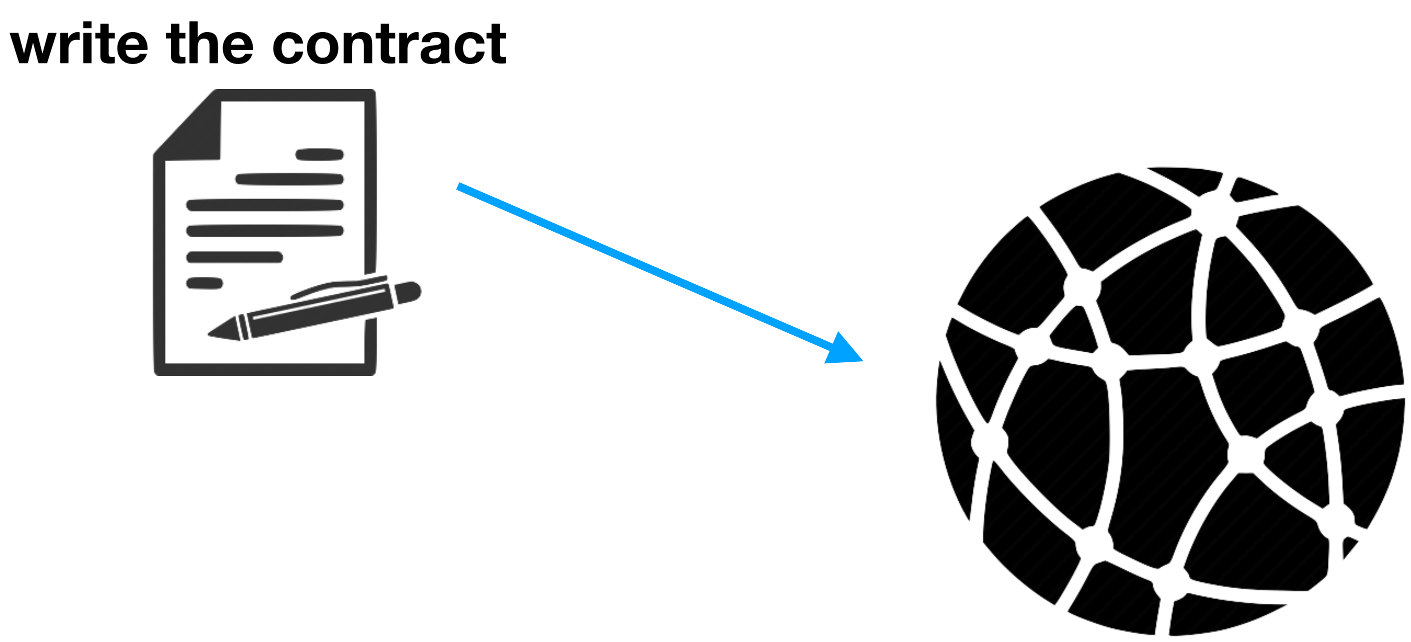
Challenges in blockchains

Can we issue credentials in this setting?



What are we trying to do?

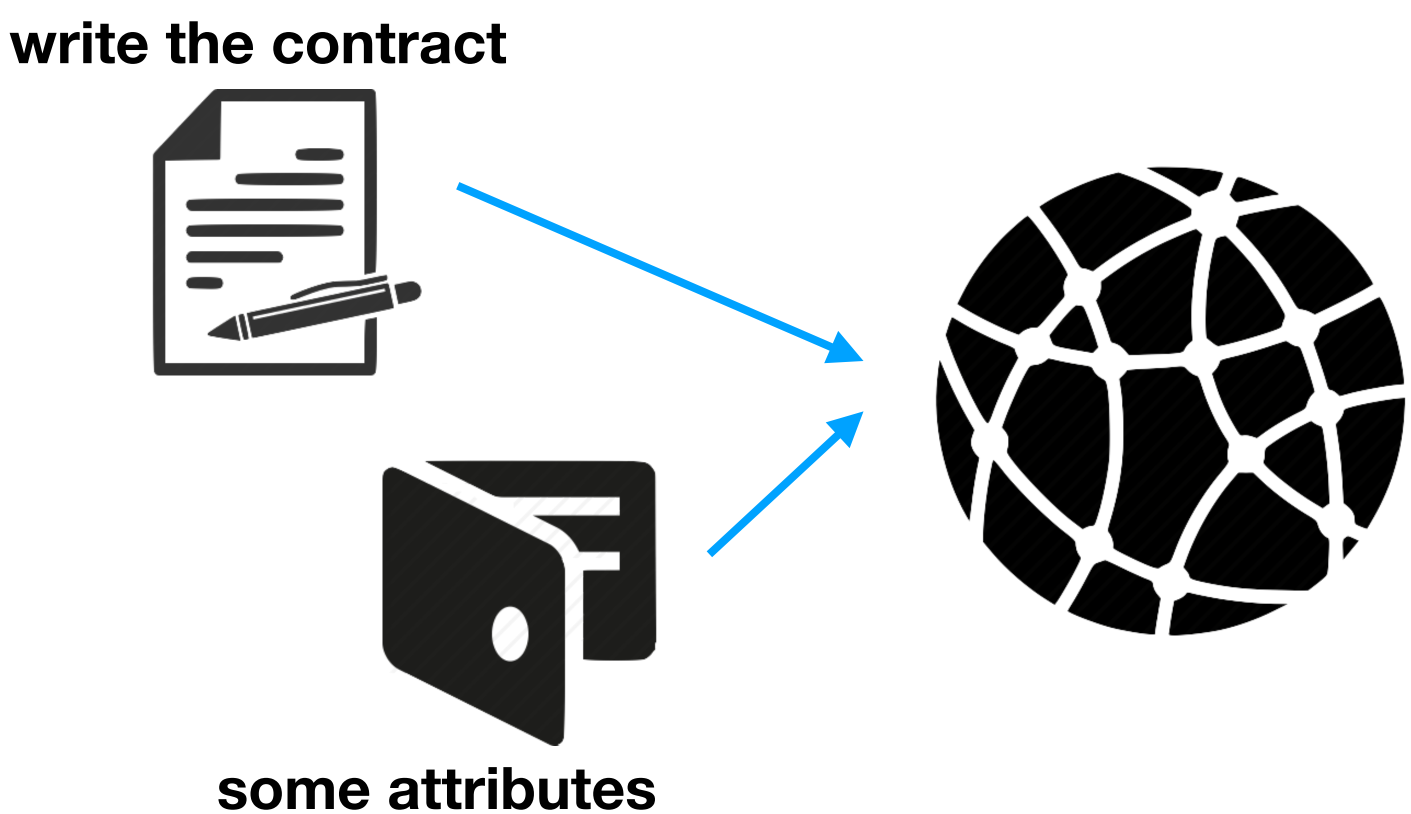
- Issuing credentials through smart contracts



... while preserving privacy

What are we trying to do?

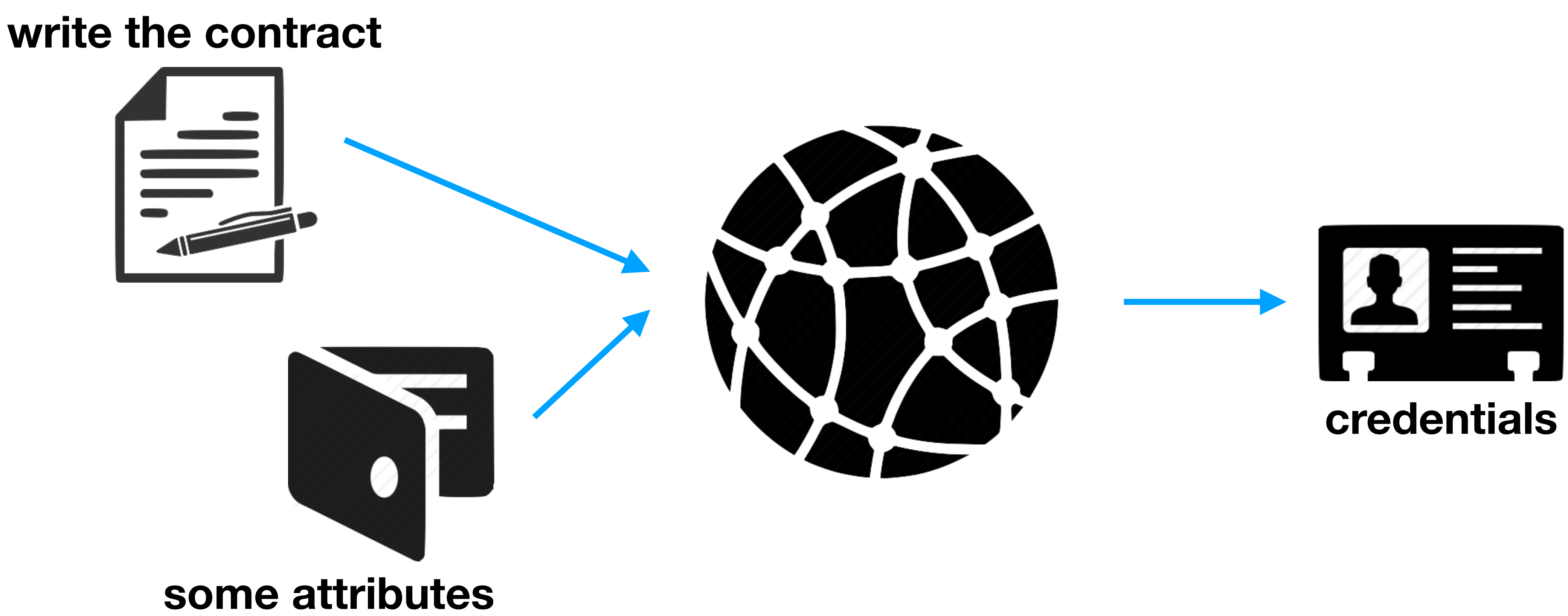
- Issuing credentials through smart contracts



... while preserving privacy

What are we trying to do?

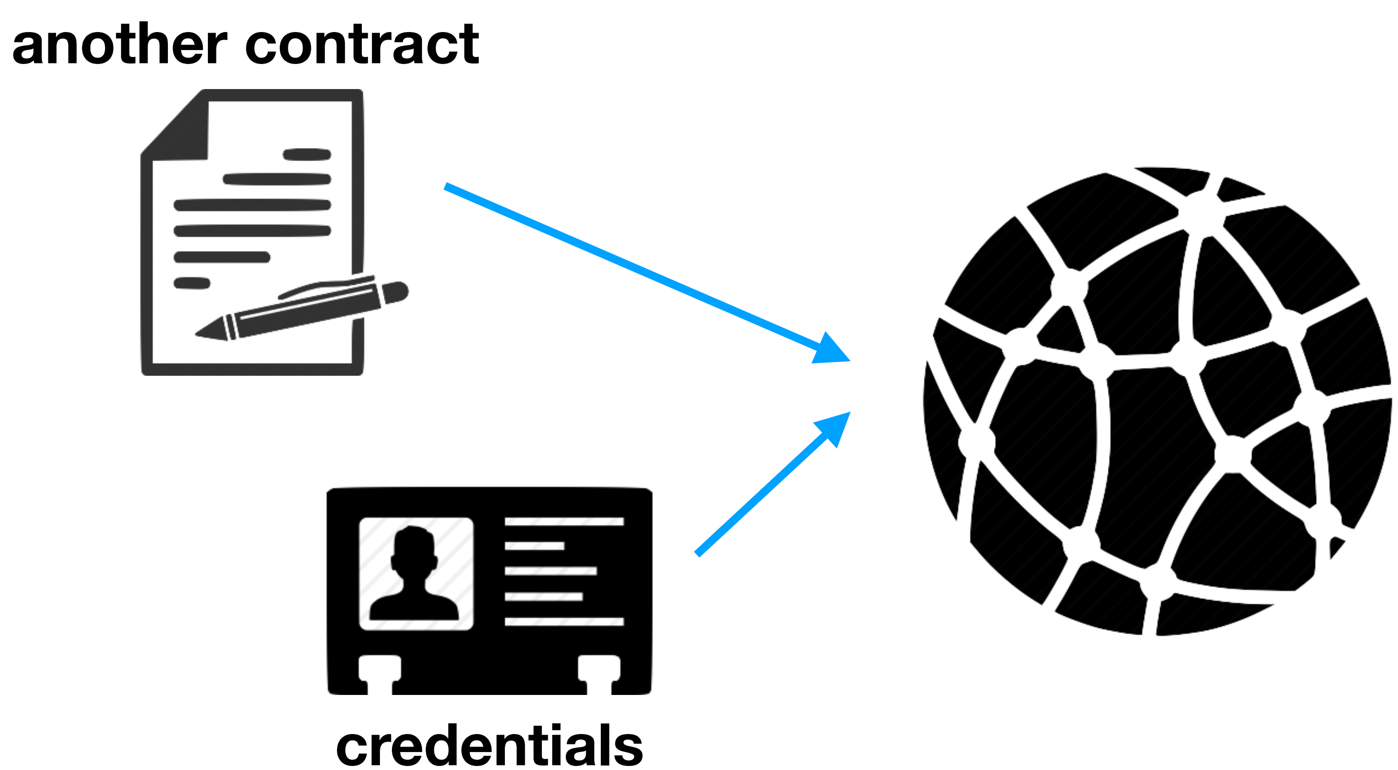
- Issuing credentials through smart contracts



... while preserving privacy

What are we trying to do?

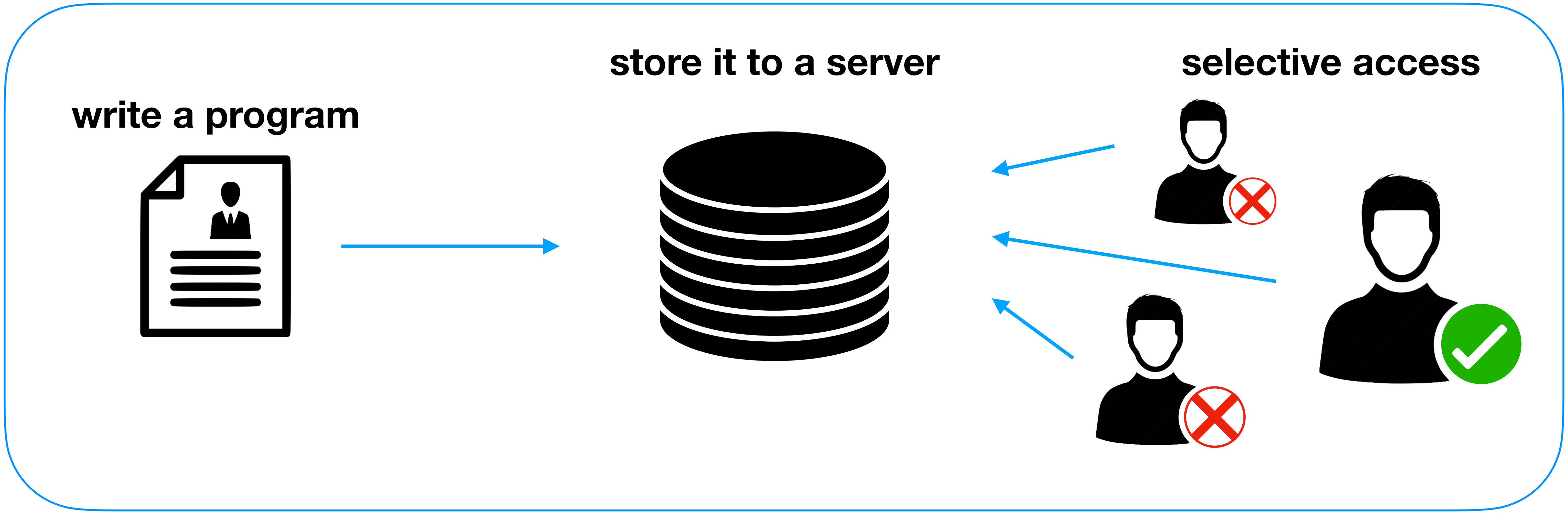
- Issuing credentials through smart contracts



... while preserving privacy

What are we trying to do?

- The more traditional setting



… but without any central authority

Distributed settings

What are we trying to do?

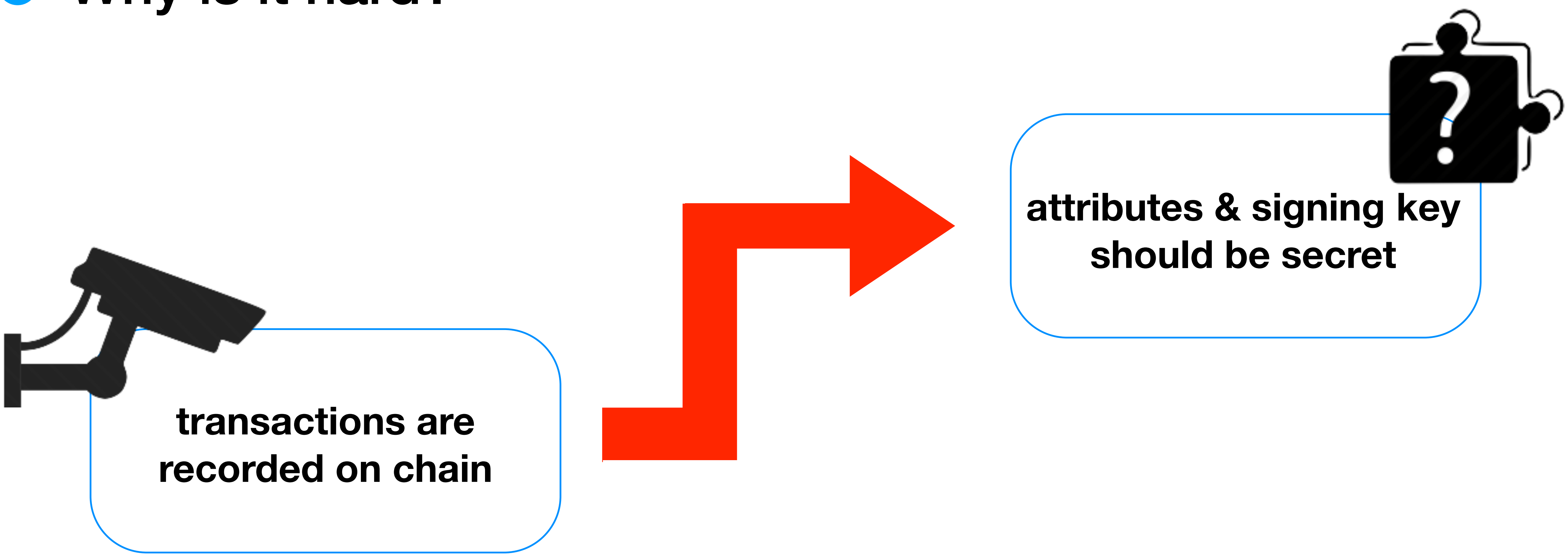
- Why is it hard?

transactions are recorded on chain

In a decentralised setting

What are we trying to do?

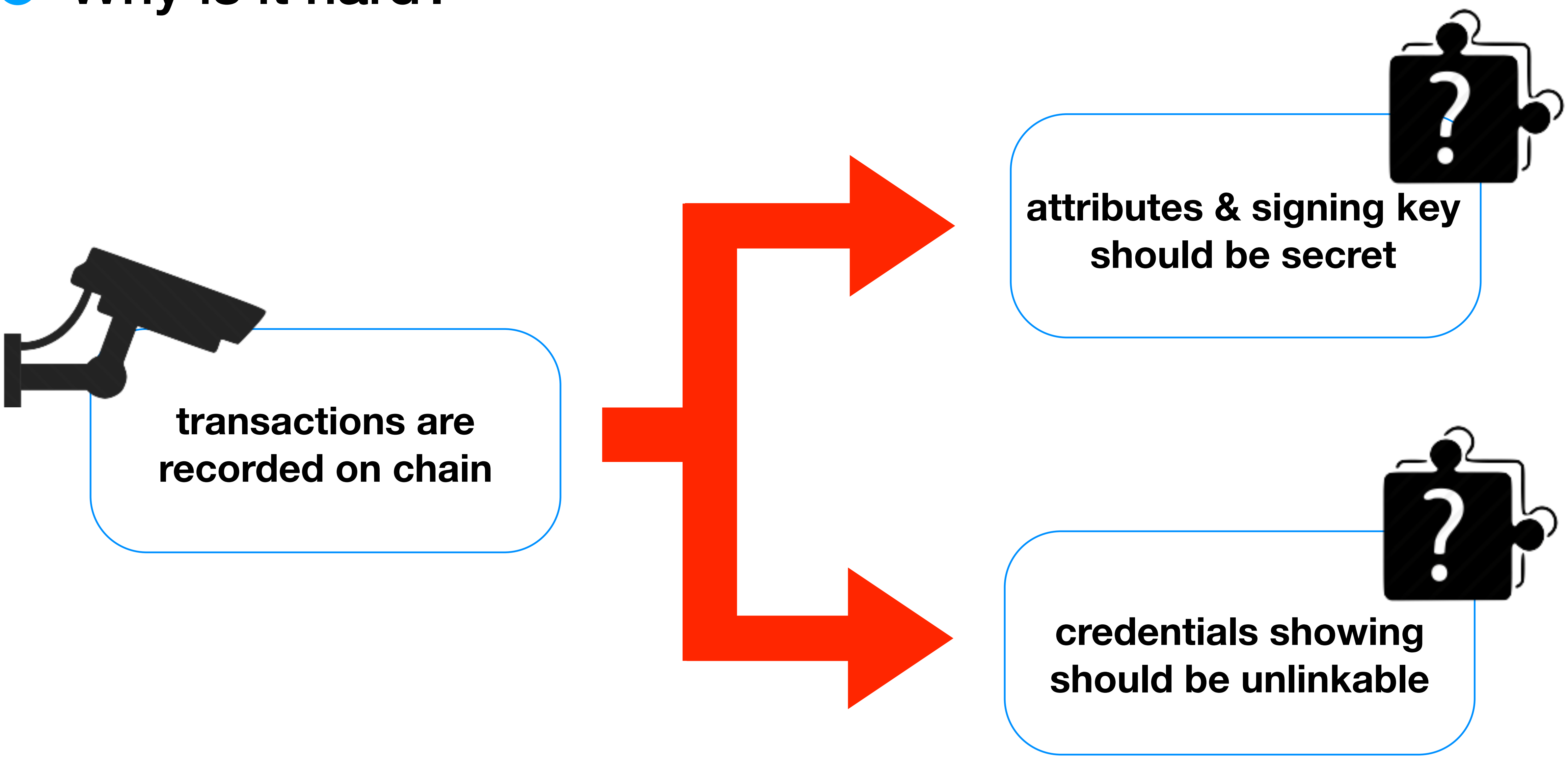
- Why is it hard?



In a decentralised setting

What are we trying to do?

- Why is it hard?



In a decentralized setting

Introduction

- Which properties do we need?

Introduction

- Which properties do we need?

Blindness

Introduction

- Which properties do we need?

Blindness

Unlinkability

Introduction

- Which properties do we need?

Blindness

Unlinkability

Threshold Authority

Introduction

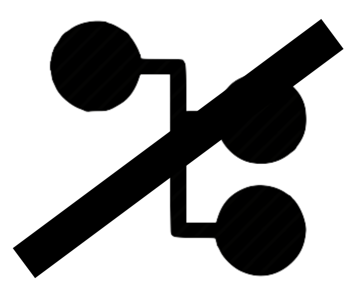
- Which properties do we need?

Blindness

Unlinkability

Threshold Authority

Authorities Non-Interactivity



Introduction

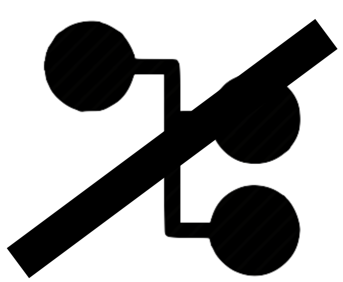
- Which properties do we need?

Blindness

Unlinkability

Threshold Authority

Authorities Non-Interactivity



Efficiency

So we built Coconut

Introduction

- Related works

Scheme	Blindness	Unlinkable	Aggregable	Threshold	Signature Size
[39] Waters Signature	✗	✗	○	✗	2 Elements
[26] LOSSW Signature	✗	✗	◒	✗	2 Elements
[8] BGLS Signature	✗	✗	●	✓	1 Element
[15] CL Signature	✓	✓	○	✗	$O(q)$ Elements
[31] Pointcheval *et al.*	✓	✓	◒	✗	2 Elements
Coconut	✓	✓	●	✓	2 Elements

○ not aggregable
◒ sequentially aggregable
● user-side aggregable
q number of attributes

Introduction

- What is Coconut?

Introduction

- What is Coconut?

Contribution I

Coconut credentials scheme

Introduction

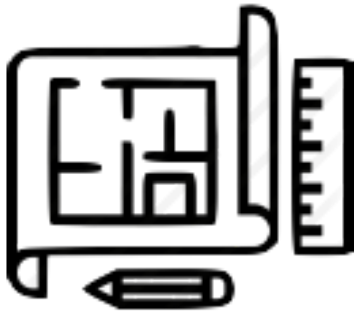
- What is Coconut?

Contribution I

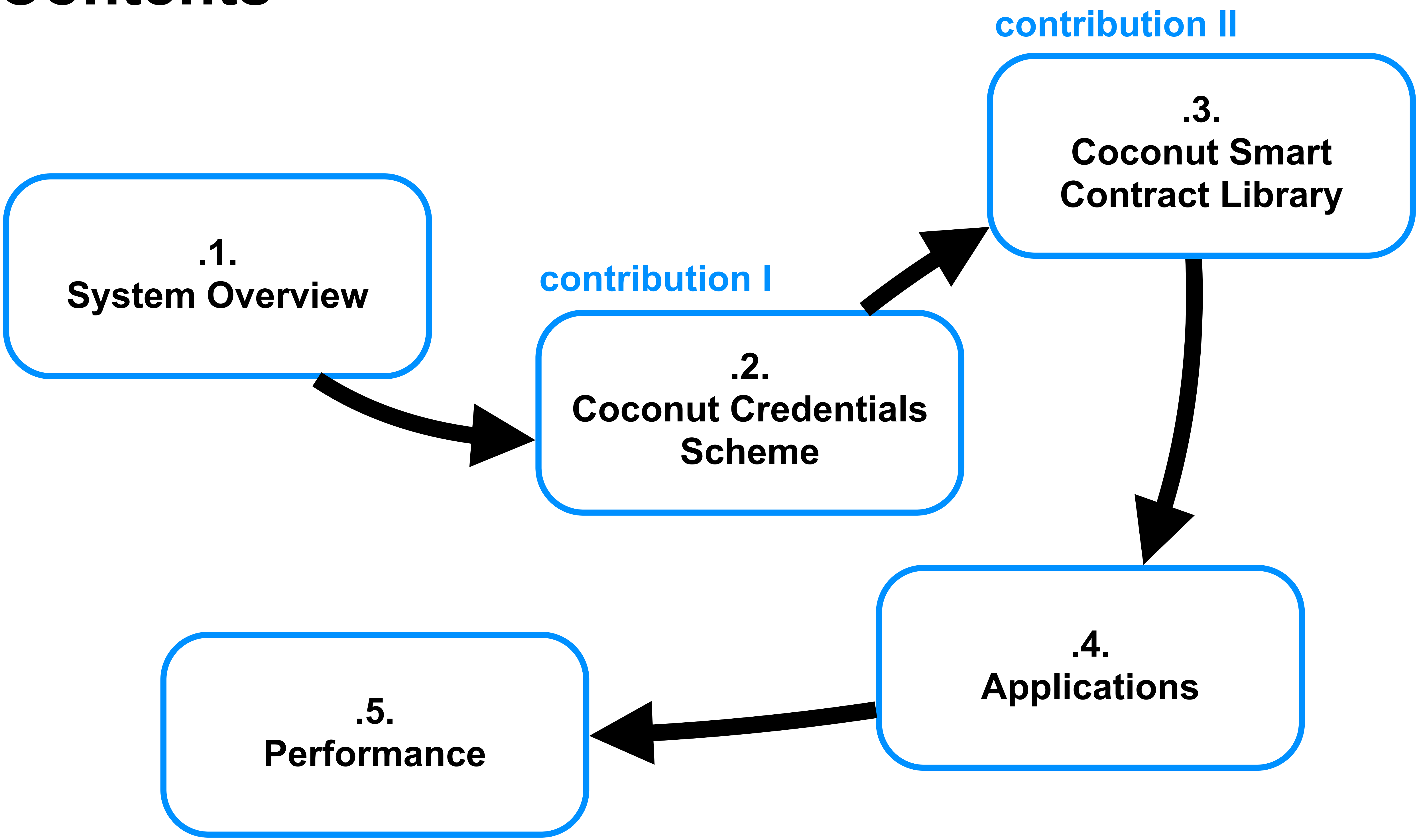
Coconut credentials scheme

Contribution II

Coconut smart contract library & example of applications

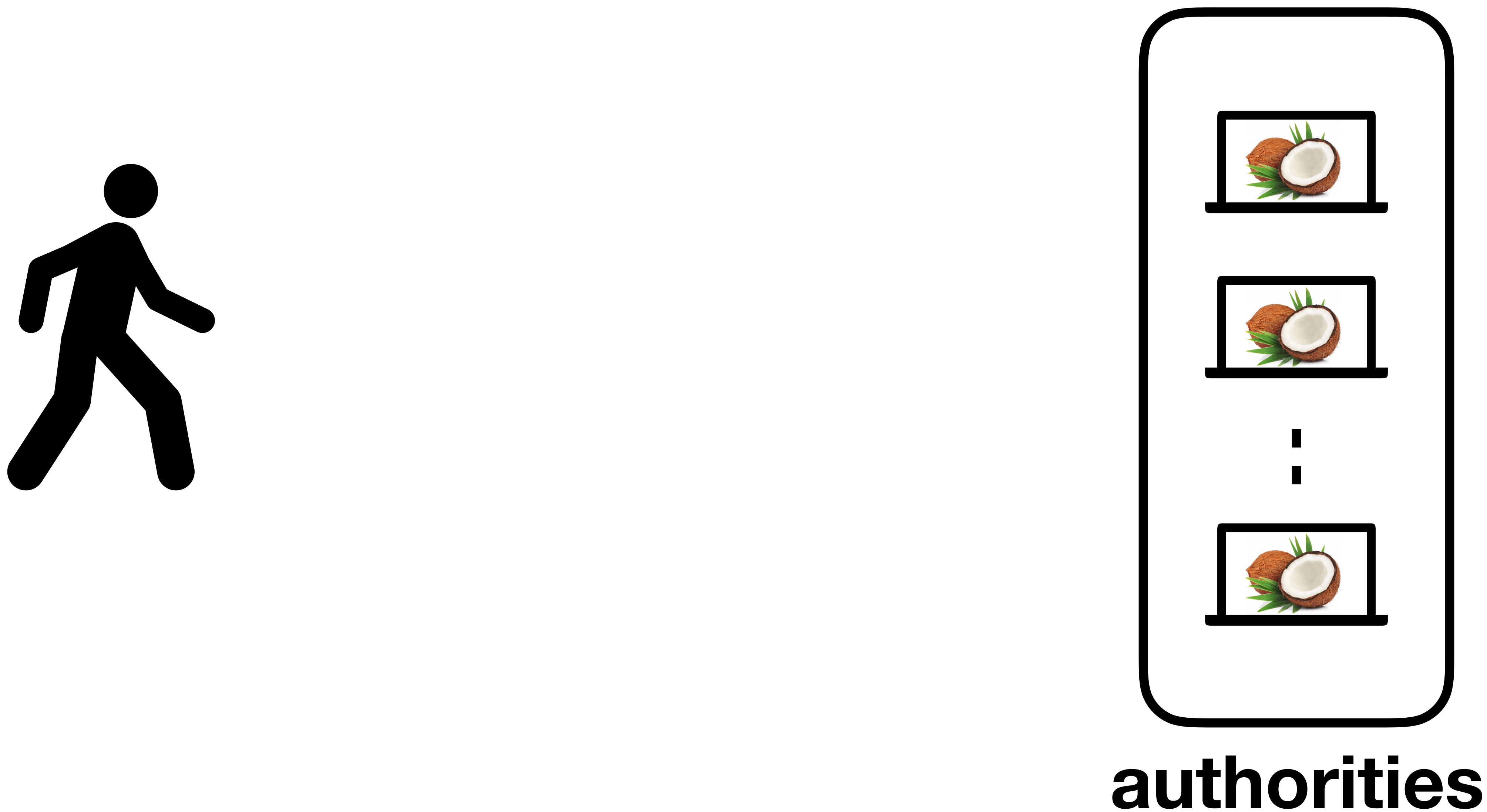


Contents


contribution II
.3.
Coconut Smart Contract Library
.1.
System Overview
contribution I
.2.
Coconut Credentials Scheme
.4.
Applications
.5.
Performance

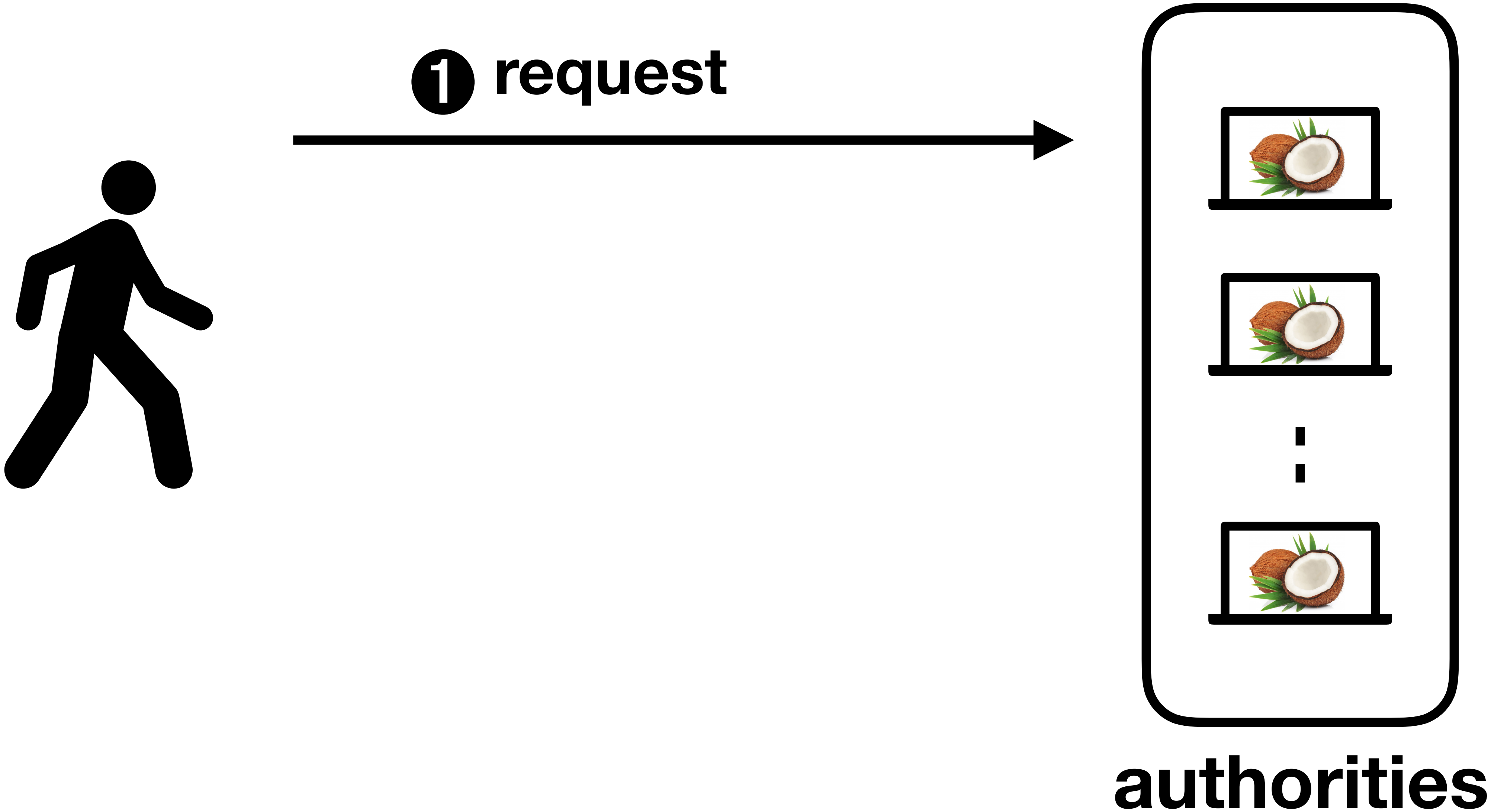
System Overview

- How does Coconut work?



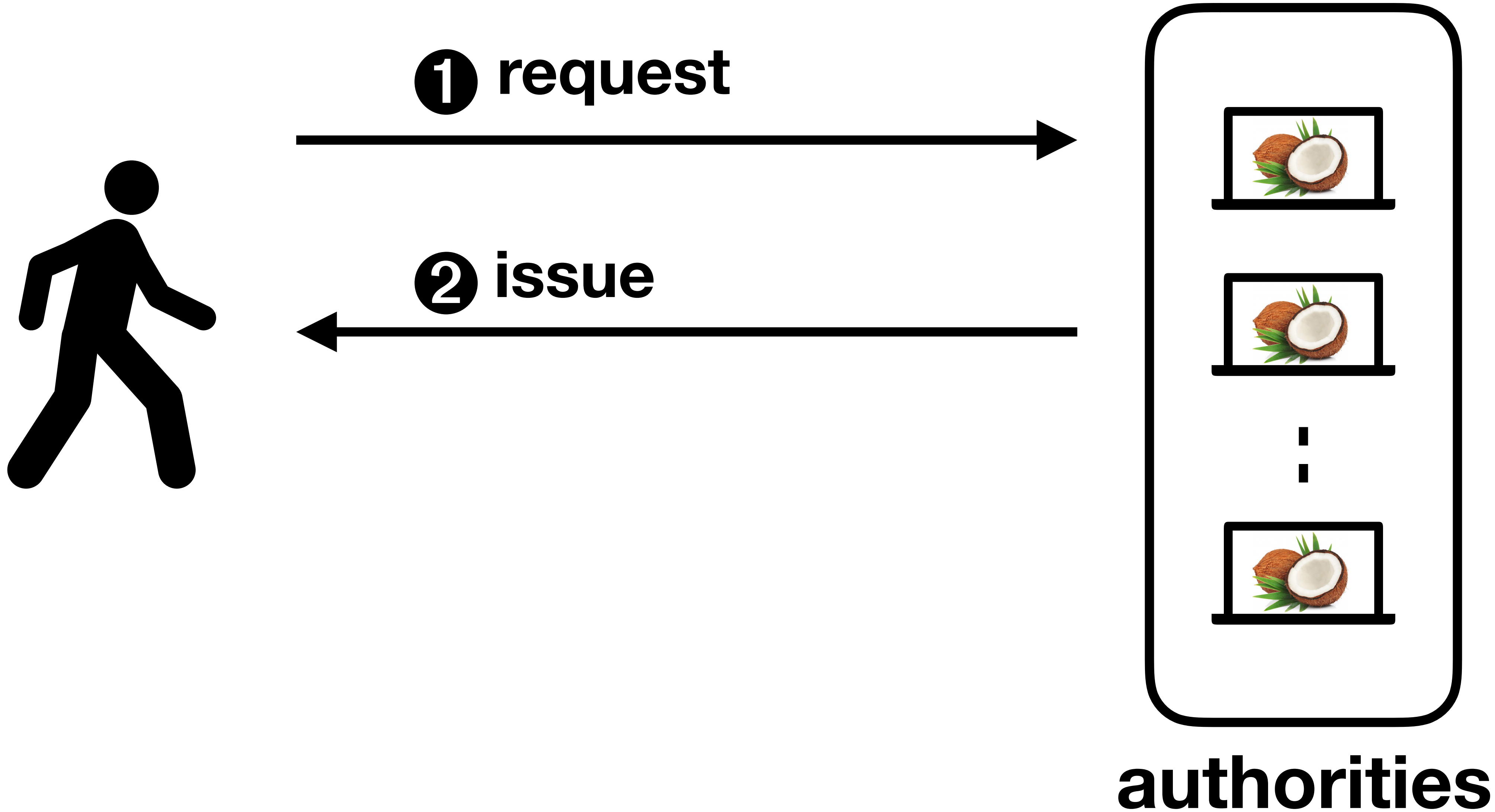
System Overview

- How does Coconut work?



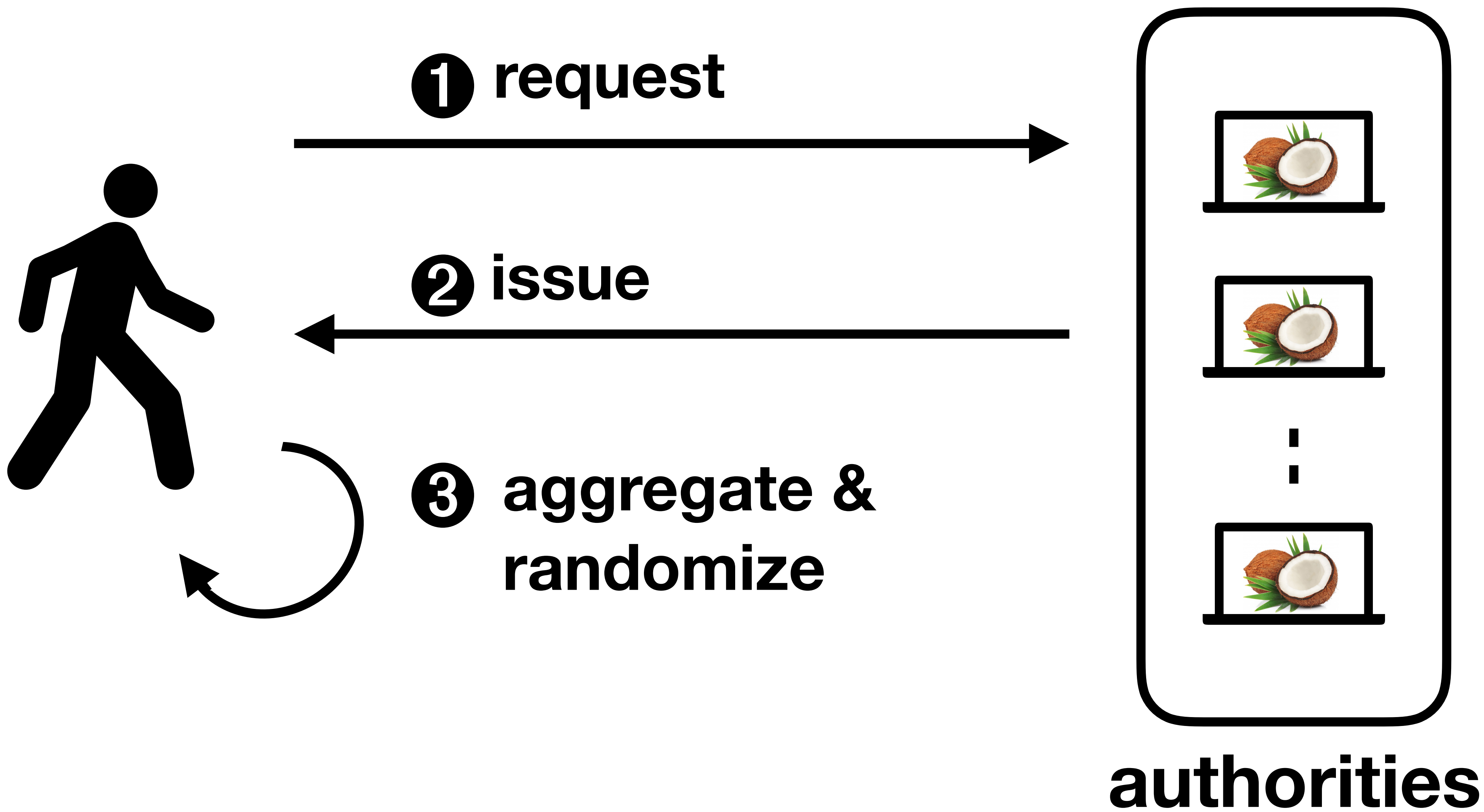
System Overview

- How does Coconut work?



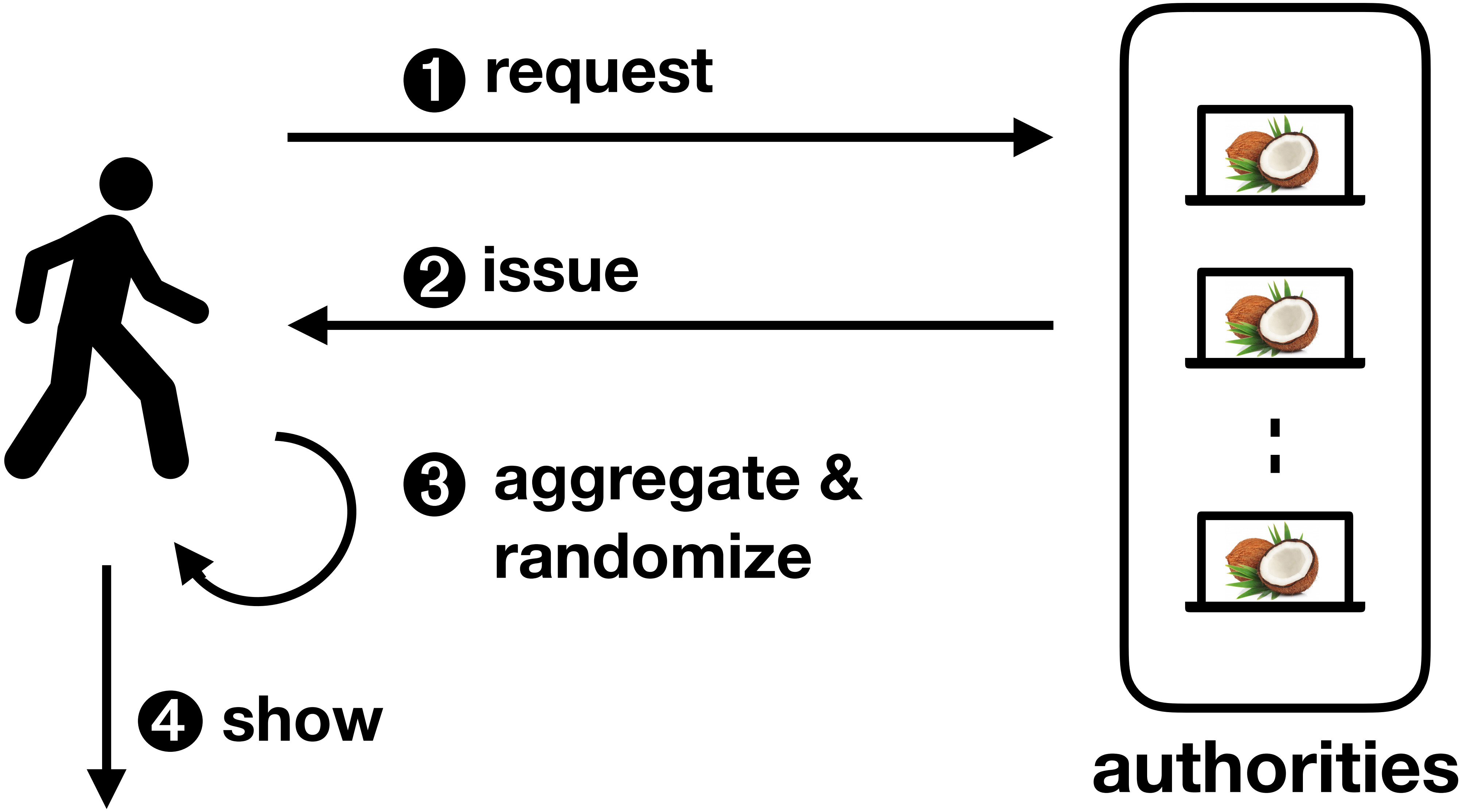
System Overview

- How does Coconut work?



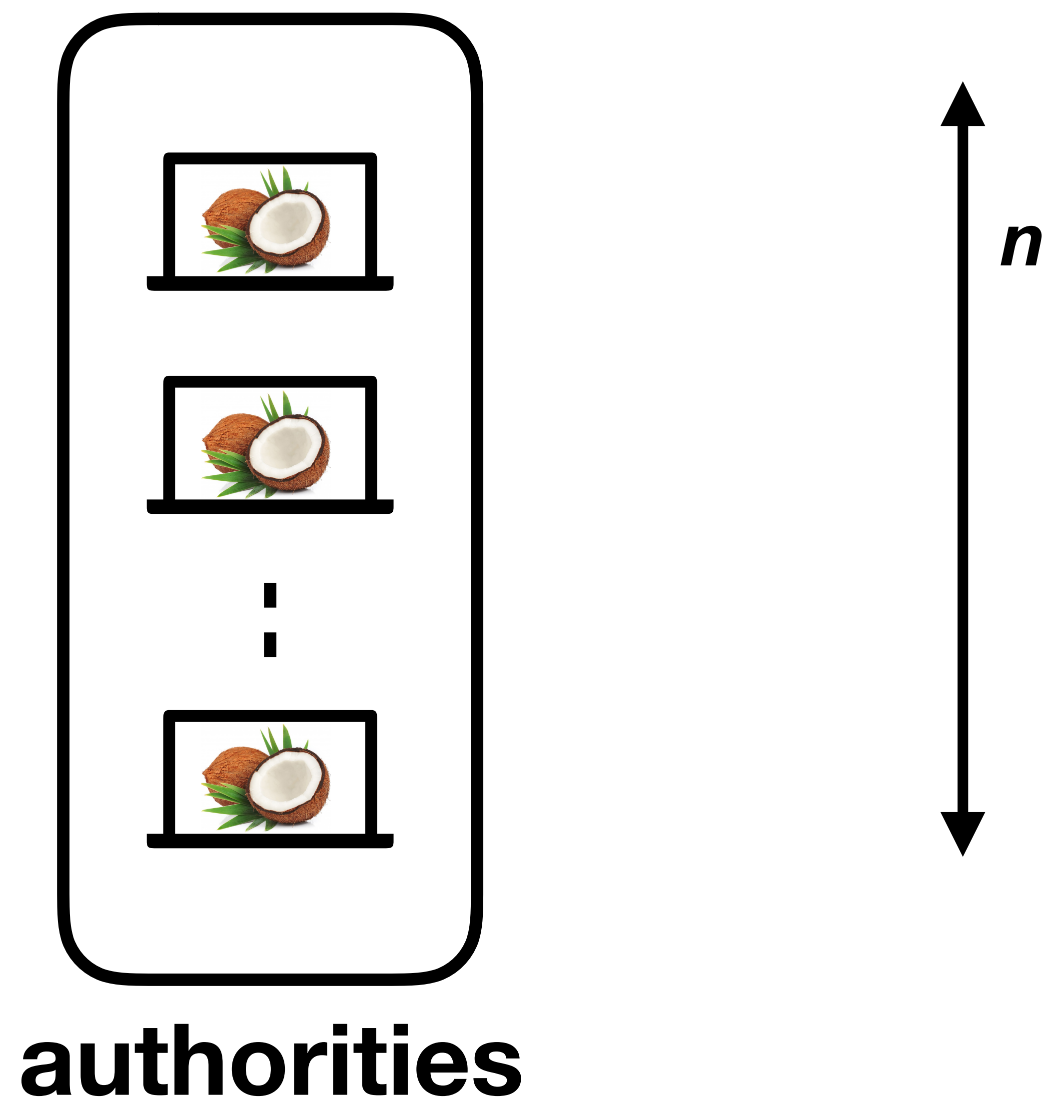
System Overview

- How does Coconut work?



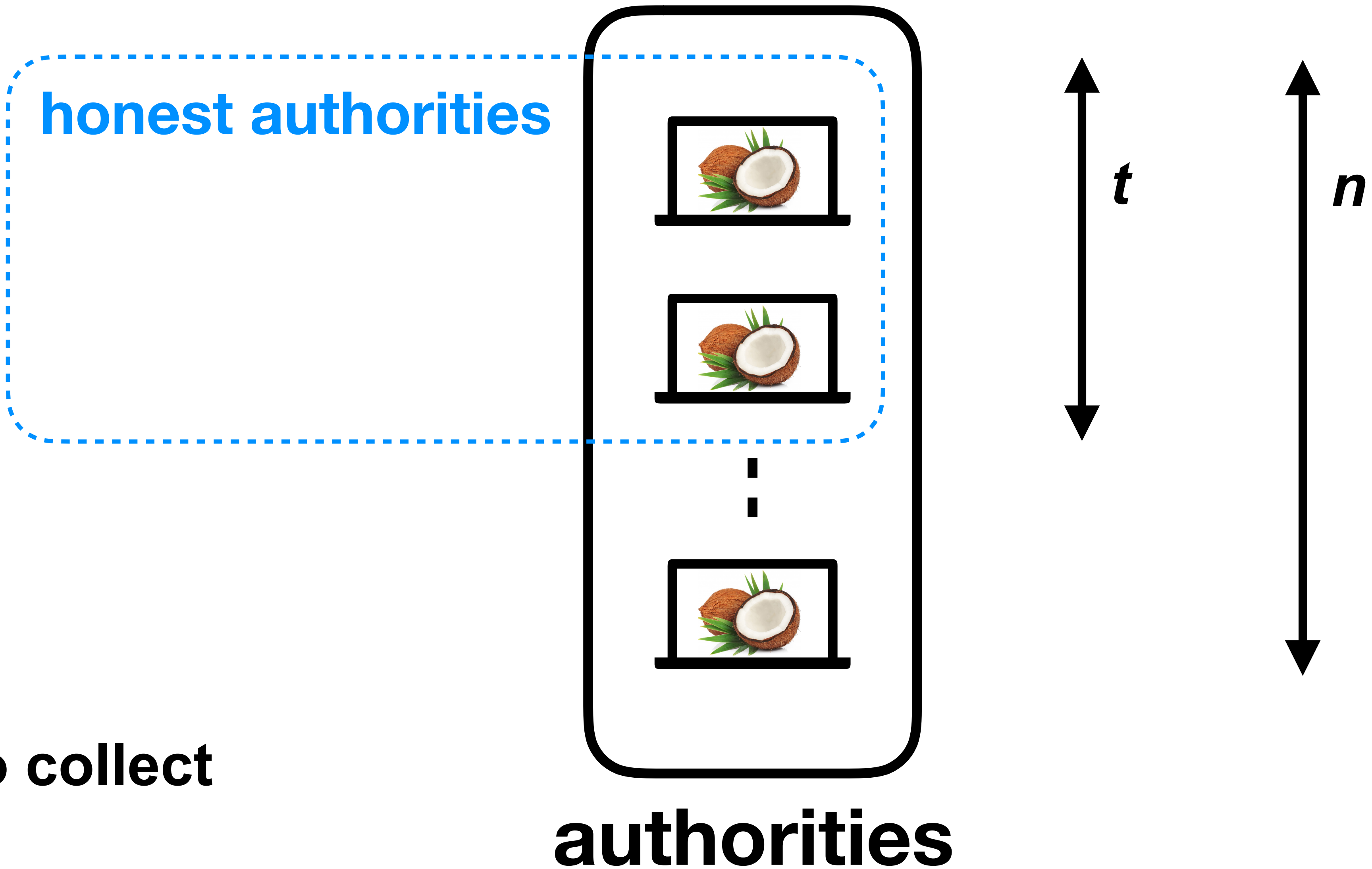
System Overview

- Threshold authorities



System Overview

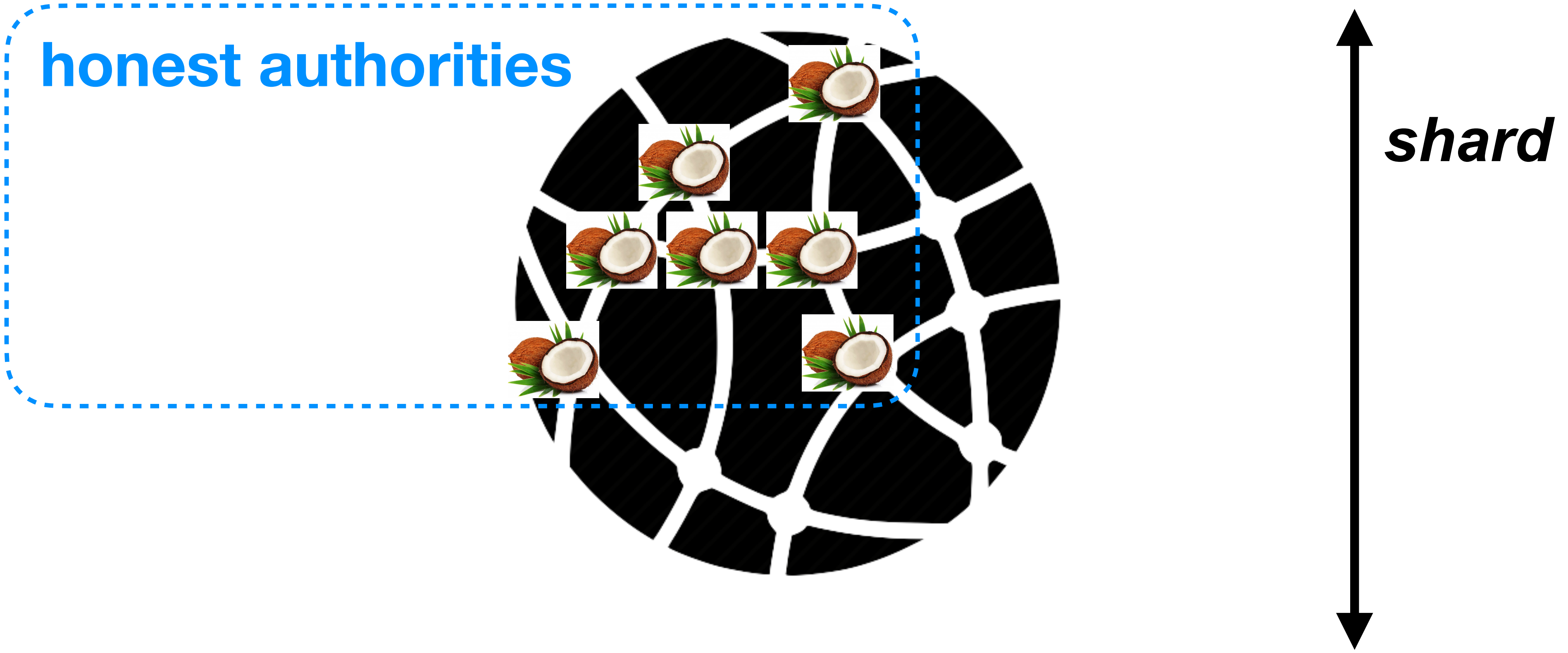
- Threshold authorities



Users need to collect only *t* shares

System Overview

- Threshold authorities



Users need to collect only (*2f+1*) shares

Coconut Credentials Scheme

- Cryptographic primitives

initialisation	❖ Setup ❖ KeyGen ❖ AggKey
blind issuance	❖ IssueCred
verify & break linkability	❖ ProveCred ❖ VerifyCred

Coconut Credentials Scheme

- From where do coconuts come from?

Coconut Credentials Scheme

- From where do coconuts come from?

Coconut

BLS Signatures + **PS Signatures**

- What do they look like?

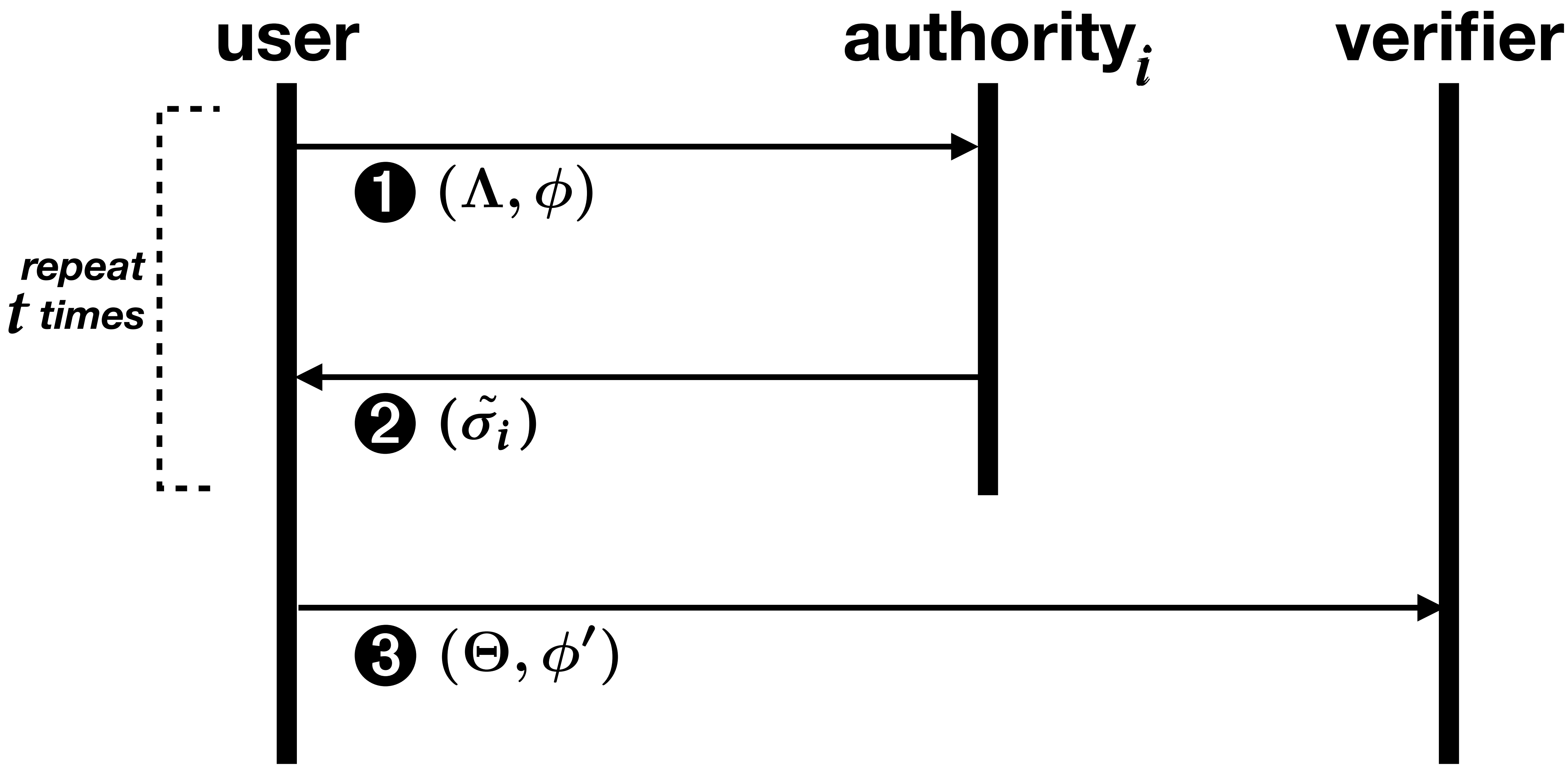
take an attribute: m

compute: $h \leftarrow H(c_m)$

signature: $\sigma \leftarrow (h, h^{x+my})$ & secret key: (x, y)

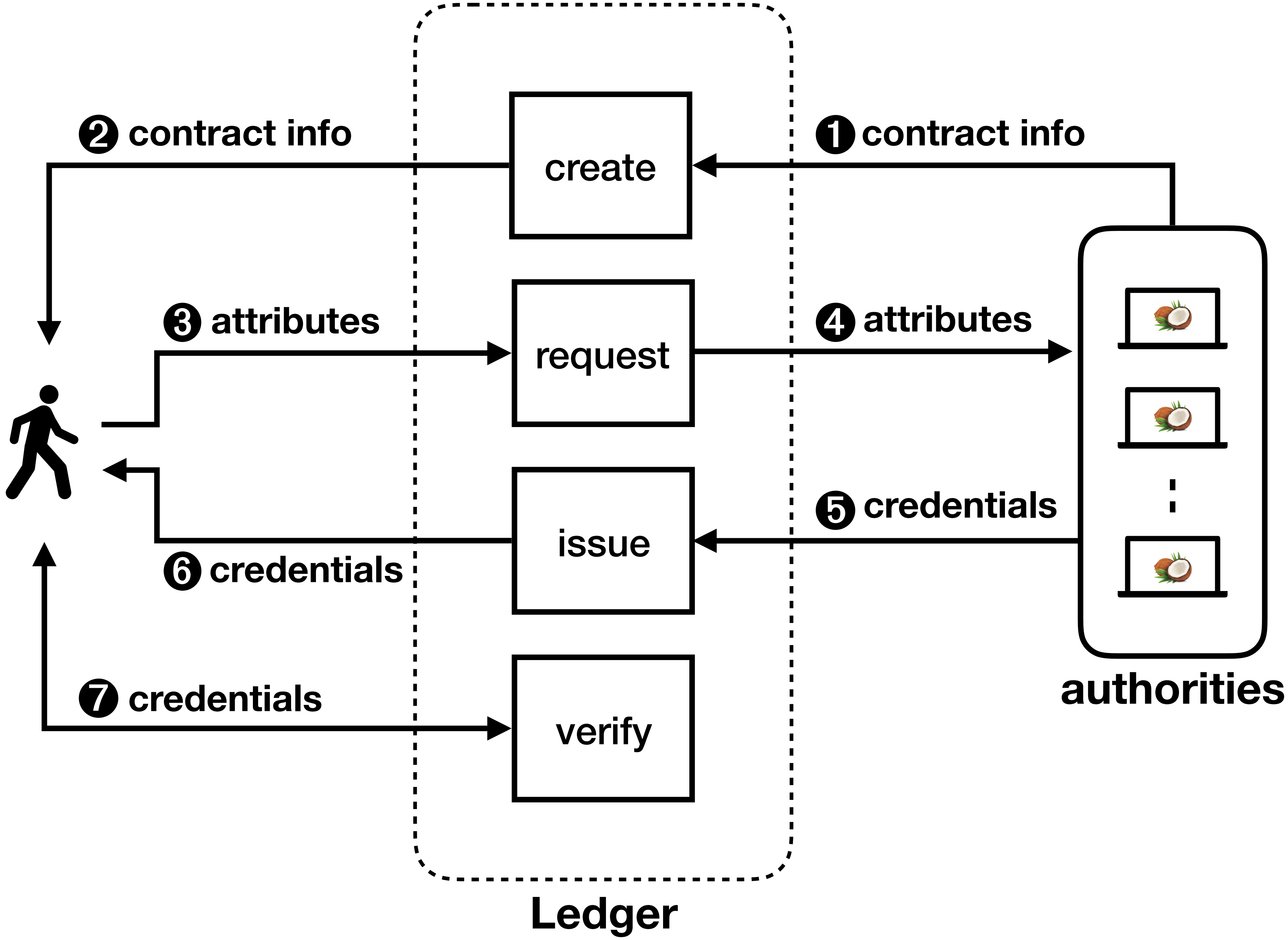
Coconut Credentials Scheme

- Communication protocol



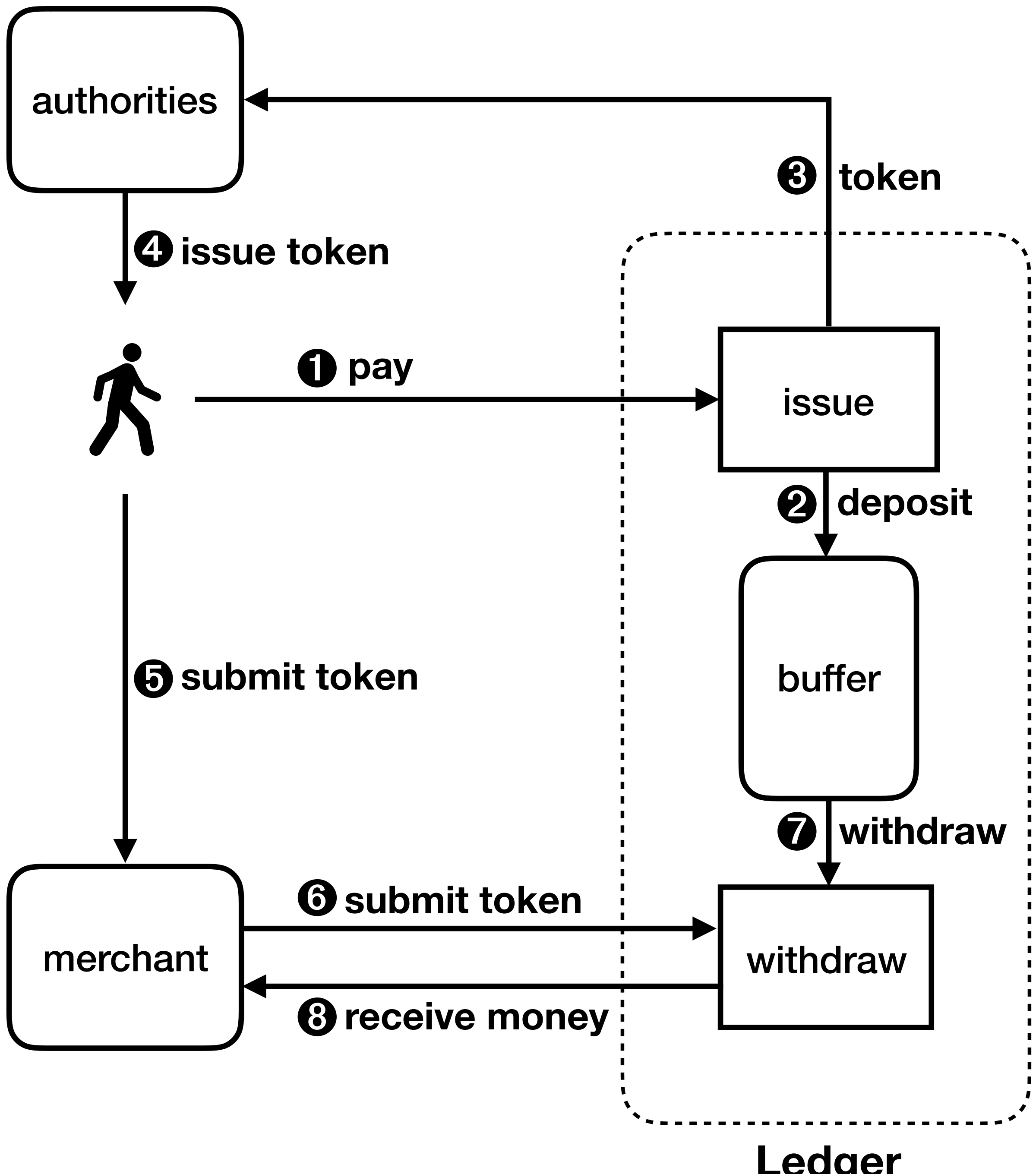
Coconut Smart Contract Library

- General purpose library



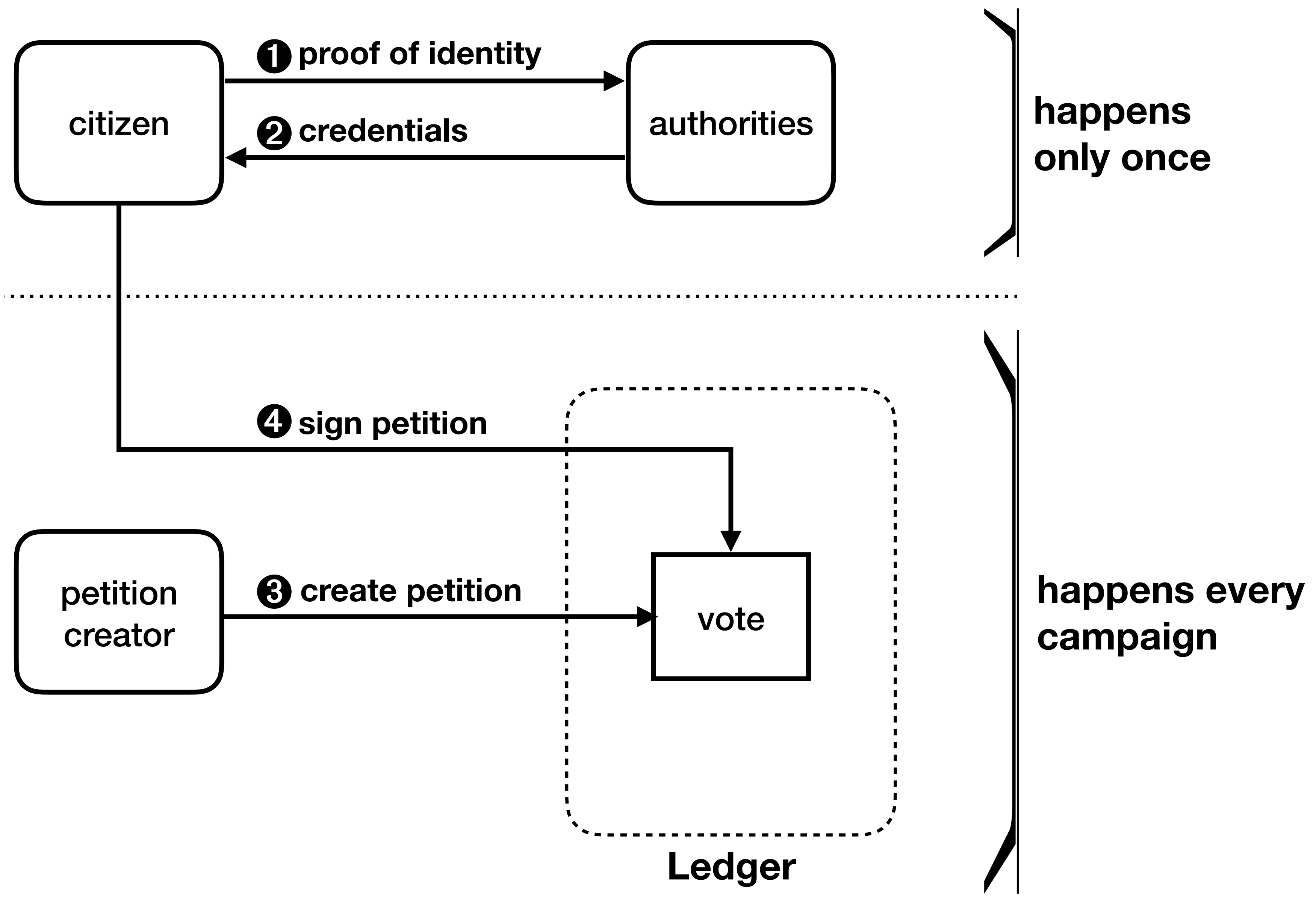
Applications

- Coin tumbler



Applications

- Privacy-preserving petitions



Performance

- What is out there?

Performance

- What is out there?

The Coconut cryptographic library

Python & Timing benchmark

Performance

- What is out there?

The Coconut cryptographic library

Python & Timing benchmark

Smart contract library

&

UCL

Performance

- What is out there?

The Coconut cryptographic library

Python & Timing benchmark

Smart contract library

&

Applications

Coin tumbler
E-Petition
(CRD proxy distribution)

Performance

- What is out there?

The Coconut cryptographic library

Python & Timing benchmark

Smart contract library

&

Applications

Coin tumbler
E-Petition
(CRD proxy distribution)

Everything is released as open source software

`https://github.com/asonnino/coconut`

Performance

- How fast is Coconut?

	Operation	μ [ms]	$\sqrt{\sigma^2}$ [ms]
	PrepareBlindSign	2.633	$\pm$ 0.003
sign	BlindSign	3.356	$\pm$ 0.002
	Unblind	0.445	$\pm$ 0.002
	AggCred	0.454	$\pm$ 0.000
	ProveCred	1.544	$\pm$ 0.001
verify	VerifyCred	10.497	$\pm$ 0.002

signing is fast, verifying takes 10ms

Performance

- What is the size of the credentials?

2 Group Elements

No matter how many attributes…

No matter how many authorities…

Performance

- How does Coconut scale?

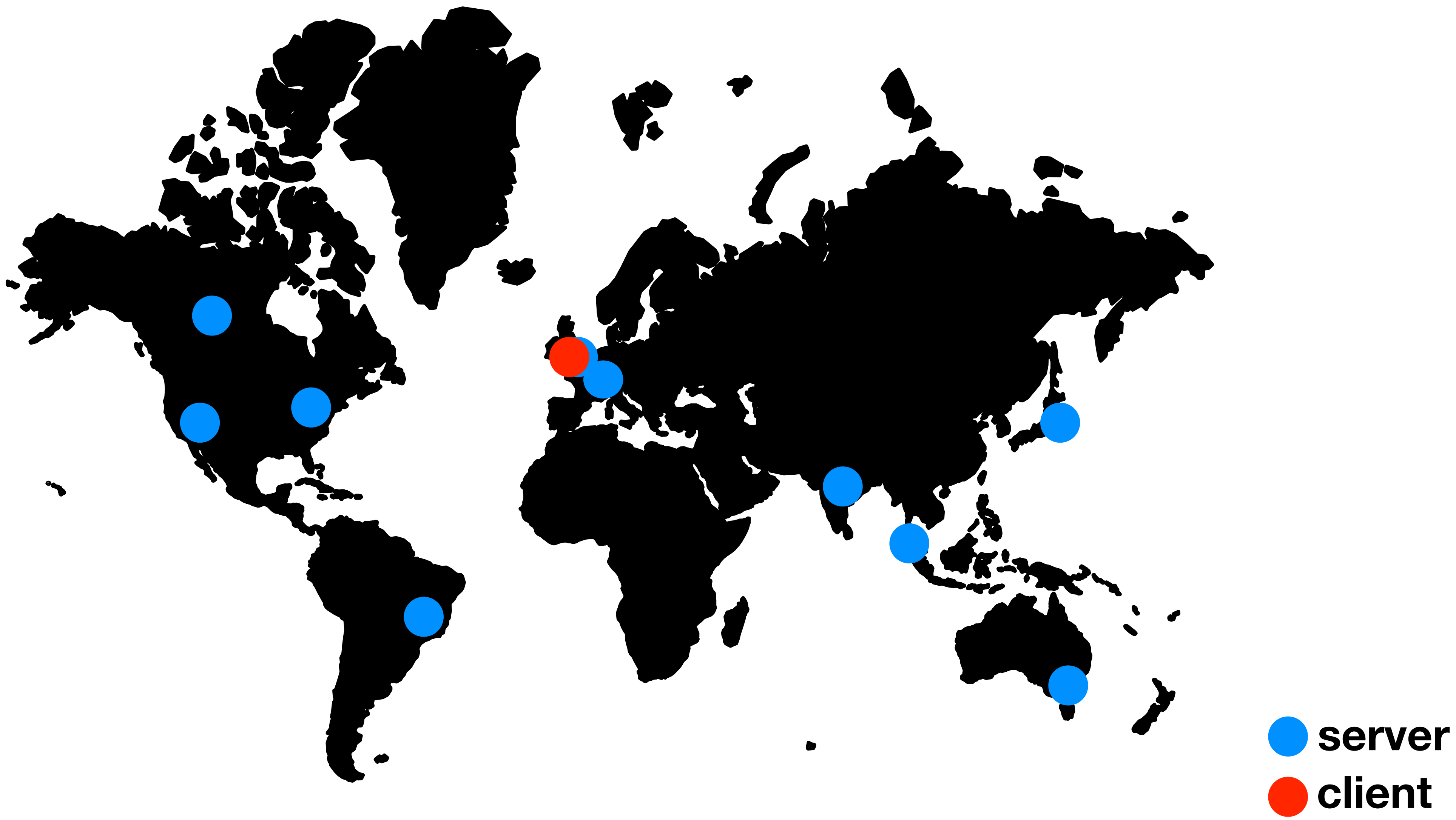
Number of authorities: n, Signature size: 132 bytes		
Transaction	**complexity**	**size [B]**
Signature on public attribute:		
❶ request credential	$O(n)$	32
❷ issue credential	$O(n)$	132
❸ verify credential	$O(1)$	162
Signature on private attribute:		
issue ❶ request credential	$O(n)$	516
❷ issue credential	$O(n)$	132
verify ❸ verify credential	$O(1)$	355

Signing scales linearly, verifying is constant time

Performance

- Did you evaluate it in the real world?

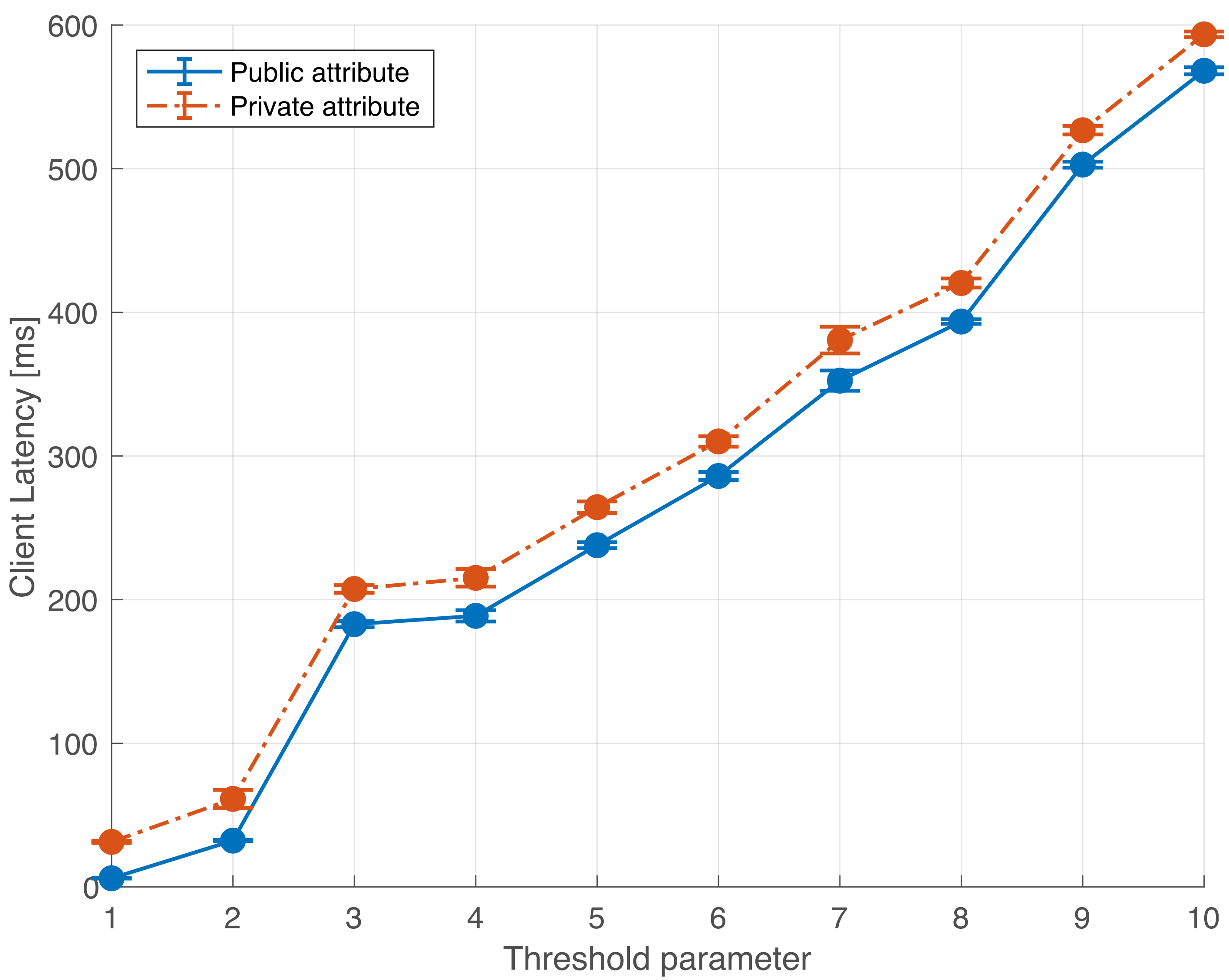
aws



pick 10 locations across the world

Performance

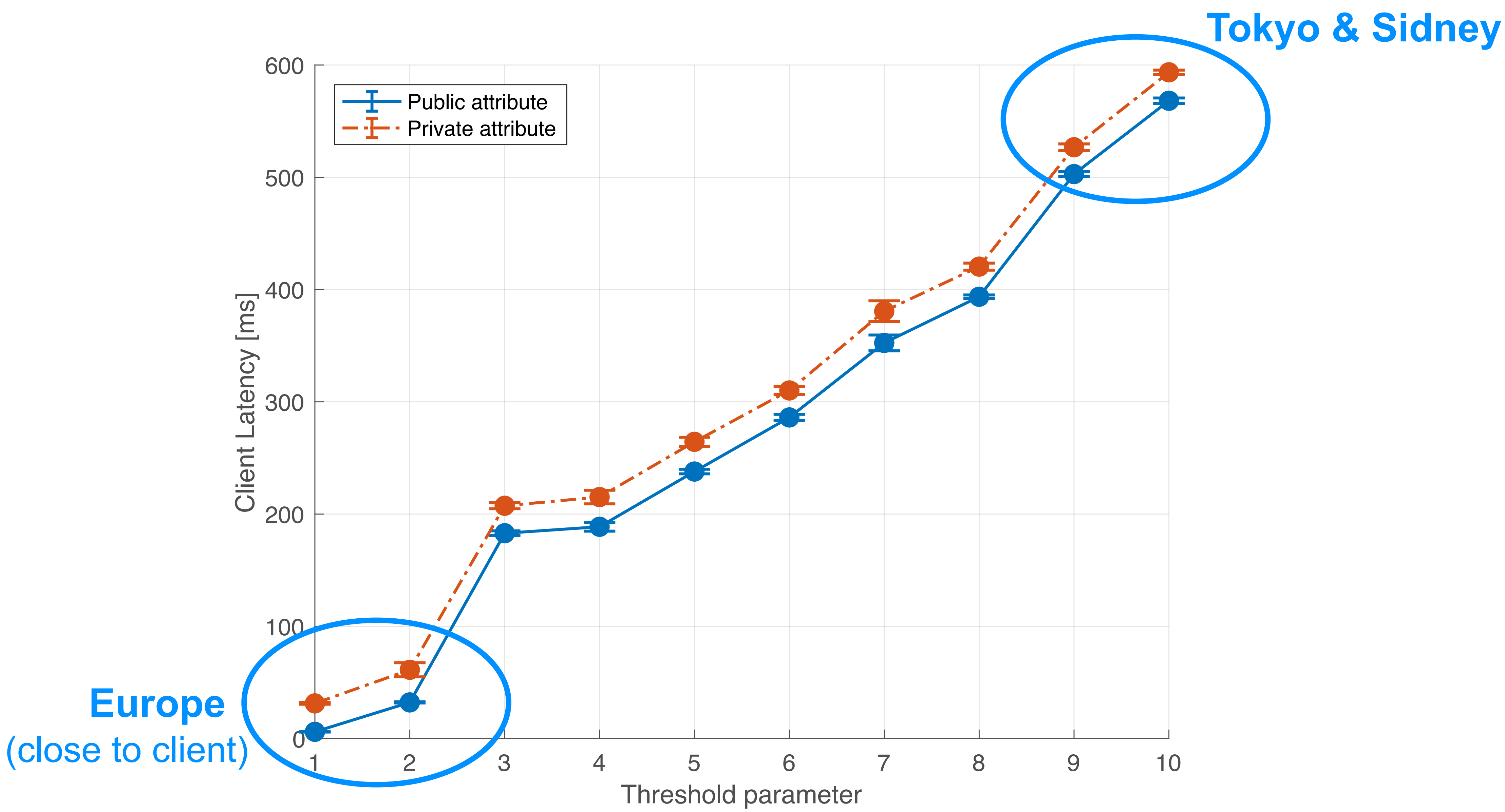
- Did you evaluate it in the real world?



client latency VS **number of authorities**

Performance

- Did you evaluate it in the real world?



client latency VS **number of authorities**

What else is in the paper?

Full cryptographic scheme

Smart contract library evaluation

Coin tumbler, CRD proxy applications

Applications evaluation and benchmarking

arXiv:submit/2158644 [cs.CR] 20 Feb 2018

Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers

Alberto Sonnino, *University College London*
Mustafa Al-Bassam, *University College London*
Shehar Bano, *University College London*
George Danezis, *University College London*, *The Alan Turing Institute*

Abstract

We present Coconut, a novel selective disclosure credential scheme supporting distributed threshold issuance, public and private attributes, re-randomization, and multiple unlinkable selective attribute revelations. Coconut can be used by modern blockchains to ensure confidentiality, authenticity and availability even when a subset of credential issuing authorities are malicious or offline. We implement and evaluate a generic Coconut smart contract library for Chainspace and Ethereum; and present three applications related to anonymous payments, electronic petitions, and distribution of proxies for censorship resistance. Coconut uses short and computationally efficient credentials, and our evaluation shows that most Coconut cryptographic primitives take just a few milliseconds on average, with verification taking the longest time (10 milliseconds).

1 Introduction

Selective disclosure credentials [15, 17] allow the issuance of a credential to a user, and the subsequent unlinkable revelation (or 'showing') of some of the attributes it encodes to a verifier for the purposes of authentication, authorization or to implement electronic cash. However, established schemes have shortcomings. Some entrust a single issuer with the credential signature key, allowing a malicious issuer to forge any credential or electronic coin. Other schemes do not provide the necessary re-randomization or blind issuing properties necessary to implement modern selective disclosure credentials. No existing scheme provides all of threshold distributed issuance, private attributes, re-randomization, and unlinkable multi-show selective disclosure.

The lack of full-featured selective disclosure credentials impacts platforms that support 'smart contracts', such as Ethereum [40], Hyperledger [14] and Chainspace [3]. They all share the limitation that verifiable smart contracts may only perform operations recorded on a public blockchain. Moreover, the security models of these systems generally assume that integrity should hold in the presence of a threshold number of dishonest or faulty nodes (Byzantine fault tolerance); it is desirable for similar assumptions to hold for multiple credential issuers (threshold aggregability).

Issuing credentials through smart contracts would be very desirable: a smart contract could conditionally issue user credentials depending on the state of the blockchain, or attest some claim about a user operating through the contract—such as their identity, attributes, or even the balance of their wallet. This is not possible, with current selective credential schemes that would either entrust a single party as an issuer, or would not provide appropriate re-randomization, blind issuance and selective disclosure capabilities (as in the case of threshold signatures [5]). For example, the Hyperledger system supports CL credentials [15] through a trusted third party issuer, illustrating their usefulness, but also their fragility against the issuer becoming malicious.

Coconut addresses this challenge, and allows a subset of decentralized mutually distrustful authorities to jointly issue credentials, on public or private attributes. Those credentials cannot be forged by users, or any small subset of potentially corrupt authorities. Credentials can be re-randomized before selected attributes being shown to a verifier, protecting privacy even in the case all authorities and verifiers collude. The Coconut scheme is based on a threshold issuance signature scheme, that allows partial claims to be aggregated into a single credential. Mapped to the context of permissioned and semi-permissioned blockchains, Coconut allows collections of authorities in charge of maintaining a blockchain, or a side chain [5] based on a federated peg, to jointly issue selective disclosure credentials.

Coconut uses short and computationally efficient credentials, and efficient revelation of selected attributes and verification protocols. Each partial credentials and the

Limitations & Future Works

- Would you like to contribute?

Limitation I

Adding and removing authorities is complicated.
Can we do better than re-running the key generation algorithm?

Limitations & Future Works

- Would you like to contribute?

Limitation I

Adding and removing authorities is complicated.
Can we do better than re-running the key generation algorithm?

Limitation II

Current key generation algorithms are complex to implement.
Can we design a key generation algorithm for blockchains?

Limitations & Future Works

- What is the next milestone?

A general framework allowing nodes to execute any kind of threshold cryptography?

Conclusion

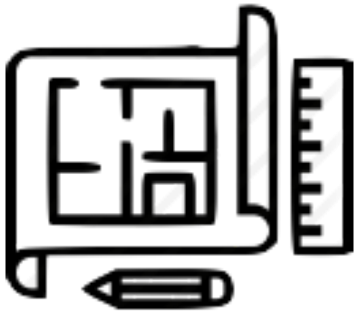
- What did we talk about?

Contribution I

Coconut credentials scheme

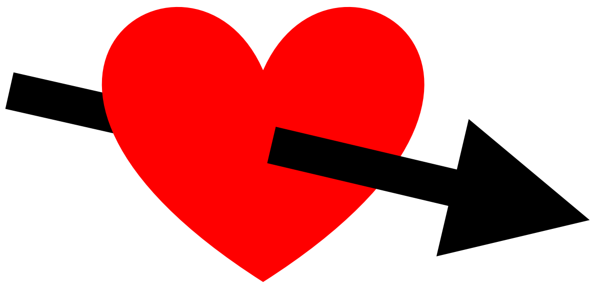
Contribution II

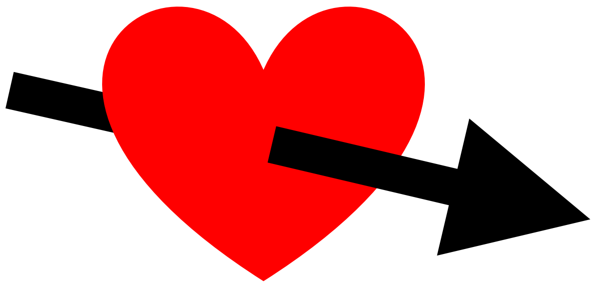
Coconut smart contract library & example of applications



Conclusion

- Main take-aways

Threshold issuance  **Sweet for blockchains**

Randomizable  **Multi-use & unlinkability**

UCL

Thank you for your attention
Questions?

Alberto Sonnino
alberto.sonnino@ucl.ac.uk
https://sonnino.com

https://github.com/asonnino/coconut

EPSRC Engineering and Physical Sciences Research Council

decode

THE ALAN TURING INSTITUTE

This work is supported in part by EPSRC Grant EP/M013286/1, the EU H2020 DECODE project (grant agreement number 732546), and The Alan Turing Institute.

The ugly

How coconuts are made

- Issue credentials

take an attribute: m

compute: $c_m = g_1^m h_1^o$ and $h = H(c_m)$

credential: $\sigma_i = (h, h^{x_i + y_i \cdot m})$ and secret key (x_i, y_i)

- Aggregate credentials

Lagrange polynomial: $l_i = \left(\prod_{i=1, j \neq i}^{t} (0 - j) \right) \left(\prod_{i=1, j \neq i}^{t} (i - j) \right)^{-1} \bmod p$

compute: $\prod_{i=1}^{t} (h^{x_i + y_i \cdot m})^{l_i} = \prod_{i=1}^{t} h^{(x_i l_i)} \prod_{i=1}^{t} h^{(y_i l_i) \cdot m} = h^{x + y \cdot m}$

How coconuts are made

- Prove credentials

public key: $(g_2, \alpha, \beta) = (g_2, g_2^{x_i}, g_2^{y_i})$

pick at random: r' and compute $\sigma' = (h^{r'}, h^{(x_i + y_i \cdot m)r'})$

pick at random: r and compute $\kappa = \alpha \beta^m g_2^r$ and $\nu = (h^{r'})^r$

- Verify credentials

parse: $\sigma' = (h', s')$

verify: $e(h', \kappa) = e(s'\nu, g_2)$

$$e(h^{r'}, g_2^{x+y\cdot m+r}) = e((h^{(x_i+y_i\cdot m)r'})(h^{r'})^r, g_2)$$