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

November 2018
The Authors

Alberto Sonnino

Mustafa Al-Bassam

Bano Shehar

Sarah Meiklejohn

George Danezis
Alberto Sonnino

- Ph.D researcher UCL
- Co-founder of Chainspace

Background:

**University College London (UCL)**
Master Degree — Information Security
United Kingdom

**Karlsruhe Institute of Technology (KIT)**
Master Degree — Information Technology Engineering
Germany

**Catholic University of Louvain (UCLouvain)**
Master Degree — Information Technology Engineering
Belgium
Challenges in blockchains

- Poor privacy
- Governance
- Scalability
- Security
Challenges in blockchains

- Poor privacy
- Scalability
- Governance
- Security

write the contract

send it to the blockchain

anyone can verify
Challenges in blockchains

Can we issue credentials in this setting?

- write the contract
- send it to the blockchain
- anyone can verify
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

write the contract

some attributes
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
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 decentralised setting

transactions are recorded on chain
What are we trying to do?

- Why is it hard?

In a decentralised setting

- transactions are recorded on chain
- attributes & signing key should be secret
- credentials showing should be unlinkable
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

<table>
<thead>
<tr>
<th>Scheme</th>
<th>Blindness</th>
<th>Unlinkable</th>
<th>Aggregable</th>
<th>Threshold</th>
<th>Signature Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>[39] Waters Signature</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>2 Elements</td>
</tr>
<tr>
<td>[26] LOSSW Signature</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
<td>2 Elements</td>
</tr>
<tr>
<td>[8] BGLS Signature</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
<td>1 Element</td>
</tr>
<tr>
<td>[15] CL Signature</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>$O(q)$ Elements</td>
</tr>
<tr>
<td>[31] Pointcheval et al.</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
<td>2 Elements</td>
</tr>
<tr>
<td>Coconut</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>2 Elements</td>
</tr>
</tbody>
</table>

○ 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

.1. System Overview

.2. Coconut Credentials Scheme

.3. Coconut Smart Contract Library

.4. Applications

.5. Performance
System Overview

How does Coconut work?
System Overview

- How does Coconut work?

1 request

authorities
System Overview

How does Coconut work?

1. request
2. issue

authorities
System Overview

- How does Coconut work?

1. **request**
   - User sends a request to the authorities.

2. **issue**
   - Authorities issue credentials to the user.

3. **aggregate & randomize**
   - The authorities aggregate the credentials and randomize the process.

**Authorities**
System Overview

How does Coconut work?

1. request
2. issue
3. aggregate & randomize
4. show

authorities

Alberto: @Bano, test system on AWS (4)
Alberto: discuss crypto related works
Alberto: compare results (speed and size) with alternatives

<table>
<thead>
<tr>
<th>Operation</th>
<th>Speed (ms)</th>
<th>Size (B)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature on clear message:</td>
<td>0.545</td>
<td>132</td>
</tr>
<tr>
<td>Signature on hidden message:</td>
<td>6.714</td>
<td>132</td>
</tr>
</tbody>
</table>

The highest transaction size appears when the user asks signature: 132 bytes. The proof size of signing authorities: 2.392 bytes. Also, the proof blindness allows a system where citizens have to go on their behave. Also, it allows the users to vote anonymous.

Malet provides a set of ORKS.
System Overview

- Threshold authorities

![Threshold Authorities Diagram](image-url)
System Overview

- Threshold authorities

Users need to collect only $t$ shares

honest authorities

authorities
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

BLS Signatures

PS Signatures
Coconut Credentials Scheme

- From where do coconuts come from?

Coconut

BLS Signatures

PS Signatures

- What do they look like?

<table>
<thead>
<tr>
<th>Take an attribute: $m$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Compute: $h \leftarrow H(c_m)$</td>
</tr>
<tr>
<td>Signature: $\sigma \leftarrow (h, h^{x+my})$ &amp; Secret key: $(x, y)$</td>
</tr>
</tbody>
</table>
Coconut Credentials Scheme

- Communication protocol

1. $(\Lambda, \phi)$
2. $(\tilde{\sigma}_i)$
3. $(\Theta, \phi')$
Coconut Smart Contract Library

- General purpose library

![Diagram of Coconut Smart Contract Library]

- User provides contract info
- User provides attributes
- User provides credentials
- Contract info is processed
- Attributes are requested
- Credentials are issued
- Credentials are verified
- Authorities process the transaction

**Figure 5: Overview of the CoCoNut petition architecture.**
Applications

- **Coin tumbler**

![Diagram of the coin tumbler process](image)
Applications

- Privacy-preserving petitions

![Diagram showing the process of creating and signing petitions](image)
Performance

- What is out there?
Performance

- What is out there?

The Coconut cryptographic library

Python & Timing benchmark

Everything is released as open source software
Performance

What is out there?

- The Coconut cryptographic library
  - Python & Timing benchmark
- Smart contract library
Performance

What is out there?

- The Coconut cryptographic library
  - Python & Timing benchmark

Applications

- Coin tumbler
- E-Petition
  (CRD proxy distribution)

Smart contract library
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?

<table>
<thead>
<tr>
<th>Operation</th>
<th>$\mu$ [ms]</th>
<th>$\sqrt{\sigma^2}$ [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrepareBlindSign</td>
<td>2.633</td>
<td>$\pm$ 0.003</td>
</tr>
<tr>
<td>BlindSign</td>
<td>3.356</td>
<td>$\pm$ 0.002</td>
</tr>
<tr>
<td>Unblind</td>
<td>0.445</td>
<td>$\pm$ 0.002</td>
</tr>
<tr>
<td>AggCred</td>
<td>0.454</td>
<td>$\pm$ 0.000</td>
</tr>
<tr>
<td>ProveCred</td>
<td>1.544</td>
<td>$\pm$ 0.001</td>
</tr>
<tr>
<td>VerifyCred</td>
<td>10.497</td>
<td>$\pm$ 0.002</td>
</tr>
</tbody>
</table>

sign

verify

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?

<table>
<thead>
<tr>
<th>Transaction</th>
<th>Complexity</th>
<th>Size [B]</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Signature on public attribute:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 request credential</td>
<td>$O(n)$</td>
<td>32</td>
</tr>
<tr>
<td>2 issue credential</td>
<td>$O(n)$</td>
<td>132</td>
</tr>
<tr>
<td>3 verify credential</td>
<td>$O(1)$</td>
<td>162</td>
</tr>
<tr>
<td><strong>Signature on private attribute:</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1 request credential</td>
<td>$O(n)$</td>
<td>516</td>
</tr>
<tr>
<td>2 issue credential</td>
<td>$O(n)$</td>
<td>132</td>
</tr>
<tr>
<td>3 verify credential</td>
<td>$O(1)$</td>
<td>355</td>
</tr>
</tbody>
</table>

Signing scales linearly, verifying is constant time
Performance

- Did you evaluate it in the real world?

pick 10 locations across the world
Performance

Did you evaluate it in the real world?

![Graph showing client latency vs threshold parameter]

**client latency VS number of authorities**
Performance

- Did you evaluate it in the real world?

![Graph showing client latency vs number of authorities]

**client latency VS number of authorities**

---

Table 1: Communication complexity and transaction size for the Coconut credentials scheme when signing one public and one private attribute.

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Operation</th>
<th>Communication Complexity</th>
<th>Transaction Size [B]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Public Attribute</td>
<td>Keygen</td>
<td>AggregateKeys</td>
<td>2.392</td>
</tr>
<tr>
<td></td>
<td>Randomize</td>
<td>AggregateSign</td>
<td>0.017</td>
</tr>
<tr>
<td></td>
<td>PrepareBlindSign</td>
<td>BlindSign</td>
<td>0.004</td>
</tr>
<tr>
<td></td>
<td>ShowBlindSign</td>
<td>BlindVerify</td>
<td>0.000</td>
</tr>
</tbody>
</table>

Table 2: Execution times for the cryptographic primitives described in Section 3. Measured over 10,000 runs.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Mean [ms]</th>
<th>std [ms]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature on public</td>
<td>132</td>
<td>0.000</td>
</tr>
<tr>
<td>Signature on private</td>
<td>32</td>
<td>0.001</td>
</tr>
</tbody>
</table>

---

As expected, we observe that the further the authorities are from the client, the higher the latency due to network delays. The average latency from 10 authorities, and latency is defined as the time to receive the signature for the last of the authorities. The dots correspond to the average latency and the error-bars represent the normalized standard deviation.

---

We evaluate the Coconut smart contract library implemented in Chainspace, as well as the privacy-preserving e-petition (Section 5.1) and the coin tumbler (Section 5.2) applications that use this library. As expected, we observe that the further the authorities are from the client, the higher the latency due to network delays. The average latency from 10 authorities, and latency is defined as the time to receive the signature for the last of the authorities. The dots correspond to the average latency and the error-bars represent the normalized standard deviation.
Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers

Alberto Sonnino
University College London

Mustafa Al-Bassam
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 cash. 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 disclosures.

The lack of full-featured selective disclosure credentials impacts platforms that support "smart contracts", such as Ethereum [40], Hyperledger [14] and Chainspace [9]. 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 authorities in charge of maintaining a blockchain, or a side chain [9] 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 credential and the

Applications evaluation and benchmarking
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
Suggestion of discussion topics

- **What else?**
  
  **Consensus.** Why sharded systems? What are the alternative to scale? Intra-shard consensus? Challenges of cross-shard consensus?

  **Trusted hardware.** Can it be useful in the context of blockchains? TEE + PETs: what can they do together? What are the challenges?

  **Privacy-preserving technologies.** Why do we need blockchains for that? Blockchains + PETs: what can they do together? What are the challenges?
Thank you for your attention
Questions?

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

https://github.com/asonnino/coconut

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

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

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