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
Byzantine
Hard to build
Expensive

No failure
Conventional
Cheap
The Authors

Alberto Sonnino

Mustafa Al-Bassam

Bano Shehar

Sarah Meiklejohn

George Danezis
Challenges in blockchains

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

write the contract

some attributes

... 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?

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
What are we trying to do?

- Why is it hard?

In a decentralized 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

How does Coconut work?

When the client aggregates partial credentials from 10 authorities, a consolidated credential is composed of exactly two group credentials. In order to sign the petition, the users compute a value and this signature acts as the citizen credentials to sign any e-petition and e-voting campaign. Successively, any third party can also be a ChainSpace node (to make clear the potential charge of CoCoNut when deeply related to blockchains), since we have actually built all of this to have credentials in smart contracts.
System Overview

- How does Coconut work?

1. request
2. issue
3. aggregate & randomize
System Overview

- How does Coconut work?

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

authorities
System Overview

- Threshold authorities

![Diagram of authorities](attachment:image.png)
System Overview

- Threshold authorities

Users need to collect only \( t \) shares

honest authorities

\( n \)

\( t \)
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 Diagram]

- 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

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

- General purpose library

![Diagram of Coconut Smart Contract Library]

---

**Table I: Performances evaluation.**

<table>
<thead>
<tr>
<th>Function</th>
<th>Time (μs)</th>
<th>Standard Deviation (μs)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Create</td>
<td>2.392</td>
<td>0.002</td>
</tr>
<tr>
<td>Request</td>
<td>7.136</td>
<td>0.001</td>
</tr>
<tr>
<td>Issue</td>
<td>10.497</td>
<td>0.005</td>
</tr>
<tr>
<td>Verify</td>
<td>13.289</td>
<td>0.002</td>
</tr>
<tr>
<td>Sign</td>
<td>3.356</td>
<td>0.001</td>
</tr>
<tr>
<td>Verify</td>
<td>5.164</td>
<td>0.002</td>
</tr>
</tbody>
</table>

**Table II: Communication complexity and the transaction size.**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Communication Complexity</th>
<th>Transaction Size</th>
</tr>
</thead>
<tbody>
<tr>
<td>PrepareBlindSign</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
<tr>
<td>ShowBlindSign</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
<tr>
<td>Randomize</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
<tr>
<td>ShowRandomize</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
<tr>
<td>GetSignature</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
<tr>
<td>AskSignature</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
<tr>
<td>VerifySignature</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
<tr>
<td>Sign</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
<tr>
<td>Verify</td>
<td>$O(n)$</td>
<td>$</td>
</tr>
</tbody>
</table>

**Figure 5: Overview of the Malet petition architecture**

- can also be Chainspace nodes (to make clear the potential
- build from the same value
- options, scores
- from the fact
- a signature on a hidden message. This comes from the fact
- to issue a blind and long-term signature on the citizen's private
- prevent a citizen to vote twice during the same campaign (prevent double spend-
- Ledger

**Figure 6: Interactions and credentials usage in the petition process.**

- UUID, owner, vk
- Sign
- Verify

**Figure 7: Communication complexity and transaction size.**

**Figure 8: Interactions and credentials usage in the petition process.**

- UUID, owner, vk
- Sign
- Verify

Alberto: @Bano, test system on AWS (https://github.com/asonnino/coconut)
Applications

- Coin tumbler

![Diagram of Coin Tumbler Process]

---

Alberto: @George, Describe how the CoCoNut authorities can also be Chainspace nodes (to make clear the potential for using chains as an open-source project on GitHub).

The signature scheme has been implemented in Python using the two crypo libraries petlib [1] and bplib [2]. The parameters of CoCoNut when deeply related to blockchains), since we actually built all of this to have credentials in smart contracts.

BlindSign {PK_{_{creation}}, \{C_{_{creation}}, 0\}} \rightarrow \{\hat{C}, \hat{m}\}

Verify \{\hat{C}, \hat{m}\}, \{PK_{_{creation}}, \{C_{_{creation}}, 0\}\}

Sign: \{PK_{_{creation}}, \{C_{_{creation}}, 0\}\} \rightarrow \{\hat{m}\}

BlindSign: \{PK_{_{creation}}, \{C_{_{creation}}, 0\}\} \rightarrow \{\hat{C}, \hat{m}\}

+ 0.004
+ 0.006
+ 0.001
+ 0.008
+ 0.001
+ 0.002

Table I: Performances evaluation.

<table>
<thead>
<tr>
<th>Operation</th>
<th>µs ±</th>
<th>YES</th>
</tr>
</thead>
<tbody>
<tr>
<td>get signature</td>
<td>6.714</td>
<td>YES</td>
</tr>
<tr>
<td>verify signature</td>
<td>0.004</td>
<td>YES</td>
</tr>
<tr>
<td>build from the same value</td>
<td>0.006</td>
<td>NO</td>
</tr>
<tr>
<td>check the time it takes to hear back from</td>
<td>0.001</td>
<td>YES</td>
</tr>
<tr>
<td>order to a spent message on which the user wish to obtain a signature. Note</td>
<td>0.002</td>
<td>YES</td>
</tr>
<tr>
<td>that the execution of each procedure described in section section II.</td>
<td>0.001</td>
<td>YES</td>
</tr>
<tr>
<td>ensure that each entry nodes that are publicly known. Second, a censor</td>
<td>0.002</td>
<td>YES</td>
</tr>
<tr>
<td>the complexity is expressed as the number of each exchange involved in the signature scheme, as</td>
<td>0.001</td>
<td>YES</td>
</tr>
<tr>
<td>ensures that each entry nodes that are publicly known. Second, a censor</td>
<td>0.002</td>
<td>YES</td>
</tr>
<tr>
<td>the size of each exchange involved in the signature scheme, as</td>
<td>0.001</td>
<td>YES</td>
</tr>
</tbody>
</table>

Bano: is this private (can everyone see current scores)?

Bano: state if this may in-
Applications

- Privacy-preserving petitions

![Diagram]

1. Proof of identity
2. Credentials
3. Create petition
4. Sign petition

happens only once

- Happens every campaign

Table II: Communication complexity and transaction size.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Communication complexity</th>
<th>Transaction size</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keygen</td>
<td>0.545</td>
<td>17.937</td>
</tr>
<tr>
<td>BlindSign</td>
<td>0.445</td>
<td>17.497</td>
</tr>
<tr>
<td>AggregateSign</td>
<td>0.005</td>
<td>16.256</td>
</tr>
<tr>
<td>BlindVerify</td>
<td>0.003</td>
<td>15.972</td>
</tr>
<tr>
<td>AggregateKeys</td>
<td>0.002</td>
<td>15.272</td>
</tr>
<tr>
<td>AggregateThSign</td>
<td>0.004</td>
<td>15.147</td>
</tr>
</tbody>
</table>

We implement a censorship-resistant system based on CoCoNut when deeply related to blockchains, since we leverage the properties of the CoCoNut protocol to build a system that is resistant to censorship. The system is implemented using the Bano protocol, and has been built from a signed private key. The signature scheme has been implemented in python and is available as an open-source project on GitHub.
Performance

• What is out there?

https://github.com/asonnino/coconut

Everything is released as open source software

Applications

Coin tumbler

E-Petition (CRD proxy distribution)
Performance

- What is out there?

The Coconut cryptographic library

Python & Timing benchmark

https://github.com/asonnino/coconut

Everything is released as open source software

Applications

Coin tumbler
E-Petition (CRD proxy distribution)
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><strong>sign</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>PrepareBlindSign</td>
<td>2.633</td>
<td>± 0.003</td>
</tr>
<tr>
<td>BlindSign</td>
<td>3.356</td>
<td>± 0.002</td>
</tr>
<tr>
<td>Unblind</td>
<td>0.445</td>
<td>± 0.002</td>
</tr>
<tr>
<td>AggCred</td>
<td>0.454</td>
<td>± 0.000</td>
</tr>
<tr>
<td>ProveCred</td>
<td>1.544</td>
<td>± 0.001</td>
</tr>
<tr>
<td><strong>verify</strong></td>
<td></td>
<td></td>
</tr>
<tr>
<td>VerifyCred</td>
<td>10.497</td>
<td>± 0.002</td>
</tr>
</tbody>
</table>

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>Signature on public attribute:</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>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Transaction</th>
<th>complexity</th>
<th>size [B]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Signature on private attribute:</td>
<td></td>
<td></td>
</tr>
<tr>
<td>issue ① request credential</td>
<td>$O(n)$</td>
<td>516</td>
</tr>
<tr>
<td>issue ② issue credential</td>
<td>$O(n)$</td>
<td>132</td>
</tr>
<tr>
<td>verify 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?

![Client Latency vs Threshold Parameter](image)

**client latency VS number of authorities**
Performance

- Did you evaluate it in the real world?

---

**Performance**

- Client-perceived latency for Coconut threshold credentials

- Client requests a partial credential

- Client latent:
  - Public attribute
  - Private attribute

- Communication complexity and transaction size for the Coconut credentials scheme

- Table 2: Communication complexity and packets size.

- Table 1: Transaction complexity size [B]

- **Figure 7:** Client Latency [ms] VS Threshold parameter

- **Europe** (close to client)

- **Tokyo & Sidney**

- **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**

---

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

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 甜 for blockchains

Randomizable Multi-use & unlinkability
Thank you for your attention
Questions?

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

https://github.com/asonnino/coconut
The ugly
How coconuts are made

- **Issue credentials**

  take an attribute: \( m \)

  compute: \( c_m = g^m_1 h^0_1 \) 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: \( l_i = \left( \prod_{i=1, j \neq i}^{t} (0 - j) \right) \left( \prod_{i=1, j \neq i}^{t} (i - j) \right)^{-1} \mod p \)

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

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