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

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

* University College London

LNETM 2018
The authors

Alberto Sonnino

Mustafa Al-Bassam

Bano Shehar

George Danezis
People love blockchains

✅ Fancy
✅ Involve money
✅ Look complicated
✅ About crypto magic
Big challenges in blockchains

- Poor privacy
- Scalability
- Governance
- Regulations
Big challenges in blockchains

- Poor privacy
- Scalability
- Governance
- Regulations

write the contract

send it to the blockchain
Big challenges in blockchains

- Poor privacy
- Scalability
- Governance
- Regulations

write the contract

send it to the blockchain

anyone can verify
Big challenges in blockchains

- Poor privacy
- Scalability
- Governance
- Regulations

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?

- Why is it hard?

Transactions are recorded on chain

In a decentralised setting
What are we trying to do?

Why is it hard?

Transactions are recorded on chain

In a decentralised setting

Attributes & signing key should be secret
What are we trying to do?

Why is it hard?

- Transactions are recorded on chain
- Attributes & signing key should be secret
- Credentials showing should be unlinkable

In a decentralised setting
So we built 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
How Coconut works?

1. request

authorities
System Overview

- How Coconut works?

1. Request
2. Issue

authorities
System Overview

- How Coconut works?

1. Request
2. Issue
3. Aggregate & Randomize

authorities
System Overview

How Coconut works?

1. **request**
2. **issue**
3. **aggregate & randomize**
4. **show**
System Overview

- Threshold authorities
System Overview

- Threshold authorities

Users need to collect only $t$ shares
Coconut Credentials Scheme

- Cryptographic primitives
  - initialisation
    - Setup
    - KeyGen
    - AggregateKey
  - get signature
    - ShowBlindSign ↔ BlindSign
  - break linkability
    - Randomize
    - AggregateSign
  - verify
    - PrepareBlindSign ↔ BlindVerify
Coconut Credentials Scheme

Where does Coconut come from?

Coconut

BLS Signature

Waters Scheme
Coconut Credentials Scheme

- Where does Coconut come from?

- What does it 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

- Issuing & showing protocols

| user | authority \(_i\) | verifier |
Coconut Credentials Scheme

- Issuing & showing protocols

\[
\text{user} \rightarrow \text{authority}_i \rightarrow \text{verifier}
\]

1. \((c_m, c, \pi_s)\)

- Encryption of \(m\)
- Proof
Coconut Credentials Scheme

- Issuing & showing protocols

\[ (c_m, c, \pi_s) \]

\[ (\tilde{\sigma}_i) \]
Coconut Credentials Scheme

- Issuing & showing protocols

**Diagram:**

1. User sends \((c_m, c, \pi_s)\) to authority \(i\).
2. Authority \(i\) sends \((\tilde{\sigma}_i)\) to user.

Repeat \(n\) times.
Coconut Credentials Scheme

- Issuing & showing protocols

1. \((c_m, c, \pi_s)\)
2. \((\tilde{\sigma}_i)\)
3. \((\kappa, \nu, \sigma, \pi_v)\)

\[\text{repeat } n \text{ times}\]

Verification process:

- The user provides credentials to the authority.
- The authority issues a challenge to the user.
- The user responds with a proof to the verifier.

The unlinkability property means that the verifier cannot link multiple executions of the protocol between each other.
Smart Contract Library

- Chainspace Coconut library

Diagram:
- 1. Contract info
- 2. Contract info
- 3. Attributes
- 4. Attributes
- 5. Credentials
- 6. Credentials
- 7. Credentials

Ledger

Authorities

Signature on clear message:

Signature on hidden message:

Table II: Communication complexity and transaction size.

<table>
<thead>
<tr>
<th>Operation</th>
<th>Clear Message</th>
<th>Hidden Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>Keygen</td>
<td>0.002</td>
<td>0.004</td>
</tr>
<tr>
<td>Verify</td>
<td>0.002</td>
<td>0.004</td>
</tr>
<tr>
<td>AggregateSign</td>
<td>0.005</td>
<td>0.006</td>
</tr>
<tr>
<td>AggregateThSign</td>
<td>0.005</td>
<td>0.006</td>
</tr>
<tr>
<td>Sign</td>
<td>0.006</td>
<td>0.007</td>
</tr>
<tr>
<td>Signature on hidden message</td>
<td>0.007</td>
<td>0.008</td>
</tr>
</tbody>
</table>

Table III: Table II shows the communication complexity and the transaction size. (measured in milliseconds, ms).
Applications

- Privacy-preserving petitions

First, the user produces a zk-proof of knowledge set of possible values (similar to cash denominations).
Applications

- Privacy-preserving petitions

![Diagram](image.png)
Applications

- Privacy-preserving petitions

1. Proof of identity
2. Credentials

happens only once

3. Create petition

happens every campaign

petition creator

vote

Ledger
Applications

- Privacy-preserving petitions

![Diagram showing the process of creating and signing a petition.]

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

**happens only once**

**happens every campaign**

**TABLE I: Performances evaluation.**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of authorities</th>
<th>Transaction complexity size [B]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregateKeys</td>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td>PrepareBlindSign</td>
<td>2</td>
<td>0.000</td>
</tr>
<tr>
<td>BlindSign</td>
<td>7</td>
<td>0.001</td>
</tr>
<tr>
<td>AggregateSign</td>
<td>1</td>
<td>0.000</td>
</tr>
<tr>
<td>GetSignature</td>
<td>1</td>
<td>0.000</td>
</tr>
</tbody>
</table>

**TABLE II: Communication complexity and transaction size.**

<table>
<thead>
<tr>
<th>Operation</th>
<th>Number of authorities</th>
<th>Communication complexity [B]</th>
</tr>
</thead>
<tbody>
<tr>
<td>AggregateKeys</td>
<td>1</td>
<td>0.001</td>
</tr>
<tr>
<td>PrepareBlindSign</td>
<td>2</td>
<td>0.000</td>
</tr>
<tr>
<td>BlindSign</td>
<td>7</td>
<td>0.001</td>
</tr>
<tr>
<td>AggregateSign</td>
<td>1</td>
<td>0.000</td>
</tr>
<tr>
<td>GetSignature</td>
<td>1</td>
<td>0.000</td>
</tr>
</tbody>
</table>

The parameters $k$, $\alpha$, and $\beta$ are set accordingly, and the execution of each procedure described in section II.

**Unlinkability:**

The owner wishes to issue some long-term credentials to their citizens in order to allow any third party to organize a e-petition and e-voting campaign. Successively, any third party from keeping track of the user's transactions and that of any authority seeing the value. Without blindness, any authority monitoring the blockchain and detecting the user's private key $vk$ gets signature $\sigma(u)$ on the UUID, owner, $vk$, and build a zero-knowledge proof showing that $k$ parameter holds the citizen's votes (initialized to any e-petition and e-voting campaigns. Then, they add their vote to the options, append $k$ parameter uniquely identifies the petition, and represents the size of the $m$ parameter.

**Bano: what is $q$?**

As described in Section 4.1. As shown in Figure 5, the client has been built from a signed private key $k$.

**III. RELATED WORKS**

Although not many schemes have actually been implemented, the scheme on clear messages allows to see why it is cool stuff; not many schemes have actually been implemented. The highest transaction size appears when the user can also be Chainspace nodes (to make clear the potential charge of authorities issuing the credentials. In order to sign the petition, a set of authorities in any e-petition and e-voting campaign. If a client runs the library. We have released the code OpenSSL as the arithmetic backend. We have released the code.

**C. Mapping authorities to blockchain nodes**

In the case of a petition, the protocol changes slightly:

1. The vote is only 157 bytes.
2. The scores are set accordingly, and $\mu$ represents the size of the $\sigma(u)$ parameter.
3. The votes are only 132 bytes.
Performance

- What did we implement?
Performance

- What did we implement?

- The Coconut cryptographic library
- Python & Timing benchmark
Performance

- What did we implement?

- The Coconut cryptographic library

- Python & Timing benchmark

- Smart contract library
Performance

What did we implement?

- The Coconut cryptographic library
  - Python & Timing benchmark

Applications

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

What did we implement?

- The Coconut cryptographic library
- Smart contract library
- Python & Timing benchmark
- Applications
  - Coin tumbler
  - E-Petition
  - (CRD proxy distribution)

Everything is released as open source software

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

- What is the credentials size?

2 Group Elements
Performance

- What is the credentials size?

  2 Group Elements

No matter how many attributes...
Performance

- What is the credentials size?

  2 Group Elements

  No matter how many attributes...

  No matter how many authorities...
Abstract

We present Coconut, a novel selective disclosure credential scheme that can be used in 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 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 [5]. 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), instead of similar assumptions used in multiple credential issuing authorities (threshold aggregability). For example, a threshold signature scheme, combined with a simple coin-tumbler-like implementation, would be very desirable—a smart contract could conditionally issue new credentials depending on the state of the blockchain, or allow some claims about a user operating through the contract—such as their identity, attributes, or even the balance of their wallet. It is not possible, with current smart contract solutions that would allow a single party to issue an issuer, or would not provide appropriate re-randomization, blinded issuance and selective disclosure capabilities (as in the case of threshold signatures [5]).

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 any small subset of potentially corrupt authorities. Coconut incorporates both selective attributes being shown to a verifier, protecting privacy even in the case all authorities and verifiers collude. This scheme is based on a threshold issuance signature scheme, and provides partial thresholds to the aggregated issuers. Mapped to the context of permissioned and semi-permissioned blockchains, Coconut allows for collections of authorities in charge of maintaining a blockchain, or a sidechain [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 credential and the
Conclusion

What did we talked about?

**contribution I**

Coconut signature scheme

**contribution II**

Coconut smart contract library
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.