Chainspace: A Sharded Smart Contract Platform

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

* University College London
** constructiveproof.com

NDSS Symposium 2018
Motivation

- Blockchains are cool — but scale badly
Motivation

- Blockchains are cool — but scale badly
Motivation

- Blockchains are cool — but scale badly
Motivation

- Blockchains are cool — but scale badly
- Hard to operate on secret inputs

Transactions are recorded on chain

Inputs are therefore public
## Motivation

### Related works

<table>
<thead>
<tr>
<th></th>
<th>Smart Contract</th>
<th>Scalable</th>
<th>Privacy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ethereum</td>
<td>✓</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Hawk</td>
<td>✓</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>ZCash</td>
<td>✗</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Omniledger</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>RSCoin</td>
<td>✗</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>
Introduction

What is chainspace?

Scalable smart contract platform
Introduction

What is chainspace?

contribution I

Scalable smart contract platform

contribution II

Supporting privacy
Contents

.1. System Overview

.2. Scalability

.3. Privacy by Design

.4. Security Properties

.5. Performance
System Overview

- How Chainspace works?
  - Nodes are organised into **shards**
  - Shards manage **objects**
  - Objects can be used only once
Scalability

How nodes reach consensus?

The S-BAC Protocol

- Byzantine Agreement
- Atomic Commit
Scalability

How nodes reach consensus?

The S-BAC Protocol

Byzantine Agreement + Atomic Commit

user

Shard 1
/manage o1/

Shard 2
/manage o2/

Shard 3
/manage o3/
Scalability

How nodes reach consensus?

The S-BAC Protocol

Byzantine Agreement + Atomic Commit

- User
- Shard 1 (manage o1)
- Shard 2 (manage o2)
- Shard 3 (manage o3)

lock

The prepare(T) message is implicit, and sequences it. The BFT-Initiator takes the lead in sequencing across shards without requiring a global consensus. This is achieved through a two-phase process where each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The S-BAC protocol manages its inputs and references. If two transactions have disjoint sets of concerned nodes, they cannot conflict, and are executed in parallel or in any order. If two transactions have common input objects, only one of them is accepted by all nodes. This is subject to the BFT-Initiator's decision, which is driven by the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds. The S-BAC protocol also supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abort, T)' holds across shards. The protocol supports a two-phase process to recover from a malicious BFT-Initiator that suppresses transactions. As previously mentioned, some of the messages in the S-BAC protocol might be lost due to unreliability or asynchrony, so they are handled by a designated node in each shard called the BFT-Initiator. Specifically, the BFT-Initiator drives the composed S-BAC protocol by sending 'prepare(T)' and then 'accept(abort, T)' or 'accept(accept, T)' to all nodes within the shard. Next, the BFT-Initiator of each shard assesses whether overall 'All proposed(accept, T)' or 'Some proposed(abol
Scalability

How nodes reach consensus?

The S-BAC Protocol

Byzantine Agreement + Atomic Commit

user

Shard 1
(manage o1)

Shard 2
(manage o2)

Shard 3
(manage o3)

lock

unlock
Scalability

- How nodes reach consensus?

The S-BAC Protocol

- Byzantine Agreement
- Atomic Commit

User

Shard 1 (manage o1)

Shard 2 (manage o2)

Shard 3 (manage o3)

lock

unlock
Scalability

The Wisdom behind S-BAC

- Only shards managing o1 and o2 are reaching consensus
- Shard 1 and shard 2 can work in parallel

![Diagram showing Scalability]
Privacy by Design

- Transaction in classic blockchains

user

node
Privacy by Design

Transaction in classic blockchains
Privacy by Design

What are Chainspace Smart Contracts?

- **user side**
- **node side**
- **execution**
- **checker**
Privacy by Design

- Chainspace transaction

user

input objects

secret data

execution

node

output objects
Privacy by Design

- Chainspace transaction

user → execution

input objects → secret data → output objects

node

input & output objects
Privacy by Design

- Chainspace transaction

**User**
- input objects
- secret data

**Execution**
- output objects

**Node**
- input & output objects

**Checker**
- ✔️ or ✗
Privacy by Design

- Private data never leave the client!
Privacy by Design

- Private data never leave the client!

![Diagram showing private data and execution within a system with a user and a node.](image)
Privacy by Design

- Private data never leave the client!
Security Properties

- What does Chainspace guarantee?
  - **Honest Shard**: among $3f+1$ nodes, at most $f$ are malicious.
  - **Malicious Shard**: over $f$ dishonest nodes.
  - Chainspace properties:
Security Properties

- What does Chainspace guarantee?
  - **Honest Shard**: among $3f+1$ nodes, at most $f$ are malicious.
  - **Malicious Shard**: over $f$ dishonest nodes.
  - Chainspace properties:

  **Transparency**

  Anyone can authenticate the history of transactions and objects that led to the creation of an object.
Security Properties

What does Chainspace guarantee?
- **Honest Shard**: among $3f+1$ nodes, at most $f$ are malicious.
- **Malicious Shard**: over $f$ dishonest nodes.

Chainspace properties:

**Transparency**

Anyone can authenticate the history of transactions and objects that led to the creation of an object.

**Encapsulation**

A smart contract cannot interfere with objects created by another contract (except if defined by that contract).
Security Properties

- What does Chainspace guarantee?
  - **Honest Shard**: among $3f+1$ nodes, at most $f$ are malicious.
  - **Malicious Shard**: over $f$ dishonest nodes.
  - Chainspace properties:

  **Transparency**
  Anyone can authenticate the history of transactions and objects that led to the creation of an object.

  **Encapsulation**
  A smart contract cannot interfere with objects created by another contract (except if defined by that contract).

  **Integrity**
  (Honest Shard)
  Only valid & non-conflicting transactions will be executed.
Security Properties

What does Chainspace guarantee?
- **Honest Shard**: among $3f+1$ nodes, at most $f$ are malicious.
- **Malicious Shard**: over $f$ dishonest nodes.
- Chainspace properties:

  - **Transparency**: Anyone can authenticate the history of transactions and objects that led to the creation of an object.
  - **Encapsulation**: A smart contract cannot interfere with objects created by another contract (except if defined by that contract).
  - **Integrity** (Honest Shard): Only valid & non-conflicting transactions will be executed.
  - **Non-Repudiation**: Misbehaviour is detectable: there are evidences of misbehaviour pointing to the faulty parties or shards.
Performance

- What did we implement?
Performance

- What did we implement?

Measured and tested on Amazon AWS
Performance

What did we implement?

- Measured and tested on Amazon AWS
- S-BAC protocol implemented in Java
- Based on BFT-SMaRt
Performance

What did we implement?

- Measured and tested on Amazon AWS
- S-BAC protocol implemented in Java
- Based on BFT-SMaRt
- Python contract simulator
  - Helps developers
  - Simulation of the checker
  - No need for full deployment
Performance

What did we implement?

- Measured and tested on Amazon AWS
- S-BAC protocol implemented in Java
- Python contract simulator
  - Helps developers
  - Simulation of the checker
  - No need for full deployment

Everything is released as open source software

https://github.com/chainspace

Based on BFT-SMaRt
Performance

How the number of shards influences the TPS?

TPS scales linearly with the number of shards.
Performance

- How does the size of the shard influence the TPS?

![Graph showing TPS vs Nodes per Shard]

**TPS decreases slowly**
Performance

How the number of inputs influence the TPS?

![Graph showing TPS vs Number of Inputs]

TPS decreases slowly and then flattens out
Performance

How is the trade off between TPS and latency?

Low latency even when the system is heavy loaded
Cross shard transactions

Smart metering contract

Platform for decision making

counts benchmarking and evaluation

What else is in the paper?
Future Works

1. How to recover from malicious shards?

2. How can a smart contract creator avoid dishonest shards?

3. How to configure shards?

4. How to incentivise nodes?
Conclusions

What did we talked about?

**contribution I**

Scalable smart contract platform

**contribution II**

Supporting privacy
Conclusions

- **Main take-aways**

  - **sharding** ➔ **scalability**
  - **execution / checker** ➔ **privacy by design**
Thank you for your attention
Questions?

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

https://github.com/chainspace

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.