A Scalable Smart Contracts Platform

Shehar Bano
Postdoc, InfoSec group
University College London

s.bano@ucl.ac.uk
@thatBano

https://github.com/chainspace
The Team

Mustafa Al-Bassam (UCL)
Alberto Sonnino (UCL)
Bano (UCL)
Dave Hrycyszyn (constructiveproof)
George Danezis (UCL)
Why Chainspace?

- Blockchains are cool — but scale badly
Why Chainspace?

- Blockchains are cool — but scale badly

- Transactions are recorded on chain
  - Inputs are therefore public

- Hard to operate on secret inputs
## Why Chainspace?

<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>
Why Chainspace?

**contribution I**

Scalable smart contract platform
Why Chainspace?

**contribution I**

Scalable smart contract platform

**contribution II**

Supporting privacy
System Overview

- How Chainspace works?
  - Nodes are organised into **shards**
  - Shards manage **objects**
Objects
Objects

- Hold state in Chainspace (e.g. Bank Account, Train Seat, Hotel Room).

- Object state is immutable.

- Objects may be in two meta-states, either active or inactive.
  - **Active objects** are available to be operated on through smart contract procedures.
  - **Inactive ones** are retained for the purposes of audit only.
Smart Contracts

- Contracts are special types of objects
- Contain executable information on how other objects be manipulated
  - Contracts contain **procedures** that define the logic by which a number of objects are processed
  - Procedures do not have to be pure functions, and may be randomized, keep state or have side effects
Composition of Smart Contracts

- A contract procedure may call a procedure of another smart contract.
- Allows the creation of a library of smart contracts from different authors that act as utilities for other higher-level contracts.
Object-to-Shard Mapping

**Transaction**
- p: procedure
- w: inputs
- r: references
- lpar: local parameters
- x: outputs
- lret: local returns
- dep: dependencies

**Shard**
- nodes
- objects
- status (active)

**Diagram:**
- Users (p): procedure (w), references (r)
- Local parameters (lpar), outputs (x), local returns (lret), dependencies (dep)
- Shards with objects and status (active)
Smart Contracts map Objects to Shards

- How to map objects to shards?

The smart contracts decide!
Transactions

- Instantiation of smart contracts
- Once a transaction is accepted in Chainspace
  - all input objects ‘die’ (become inactive)
  - all output objects are ‘born’ (become active)

<table>
<thead>
<tr>
<th>Object 1</th>
<th>Balance: 10</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice’s wallet</td>
<td></td>
</tr>
<tr>
<td>Object 1: active</td>
<td></td>
</tr>
<tr>
<td>Object 2: nonexistant</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Object 2</th>
<th>Balance: 5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice’s wallet</td>
<td></td>
</tr>
<tr>
<td>Object 1: inactive</td>
<td></td>
</tr>
<tr>
<td>Object 2: active</td>
<td></td>
</tr>
</tbody>
</table>
Contents

1. System Overview
2. Privacy by Design
3. Scalability
4. Security Properties
5. Performance
Checkers

- Every smart contract has a checker

user side

node side

smart contract

checker
Checkers

- Checkers are pure functions (i.e., deterministic, and have no side-effects), and return a Boolean value.

- A checker requires **no secret inputs**
  - only has sufficient information to check transaction validity (e.g., a zero-knowledge proof)
Privacy by Design

- Transaction in classic blockchains

user

node

contract

secret data

input state

output state
Privacy by Design

- Chainspace transaction

user

- input objects
- secret data

execution

node

- output objects
Privacy by Design

- Chainspace transaction

**User**
- Input objects
- Secret data
- Execution
- Output objects

**Node**
- Input & output objects, proof of correctness
Privacy by Design

- Chainspace transaction

**user**

- input objects
- secret data

**execution**

- output objects

**node**

- input & output objects, proof of correctness

- checker

- ✔️ or ✗
Example: Private E-petitions

- Legitimate user casts a vote without identifying herself

Cast a vote

Check zero-knowledge proof that the vote corresponds to an actual user who possesses a valid ID (secret data)
Contents

.1. System Overview

.2. Privacy by Design

.3. Scalability

.4. Security Properties

.5. Performance
Consensus

How do nodes agree whether to accept or reject a transaction?
Inter-Shard Consensus
Inter-Shard Consensus

- **Byzantine Fault Tolerant (BFT) protocol** which guarantees:
  - **Safety**: All honest members of a shard of size $3f + 1$, agree on a specific common sequence of actions, despite some $f$ malicious nodes within the shard.
  - **Liveness**: When agreement is sought, a decision or sequence will eventually be agreed upon.
Intra-Shard Consensus

Transaction

- p: procedure
- w: inputs
- r: references
- lpar: local parameters
- x: outputs
- lret: local returns
- dep: dependencies

user

Shard

<table>
<thead>
<tr>
<th>objects</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1</td>
<td>active</td>
</tr>
<tr>
<td>o2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

Shard

<table>
<thead>
<tr>
<th>objects</th>
<th>status</th>
</tr>
</thead>
<tbody>
<tr>
<td>o1</td>
<td>active</td>
</tr>
<tr>
<td>o2</td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>
Intra-Shard Consensus

Travel agent

Book hotel room #12 in shard 1

Book train seat #33 in shard 2
Intra-shard Consensus

- Atomic commit protocol
- A transaction is only accepted if all the concerned shards agree, otherwise it is rejected
S-BAC Consensus Protocol

How nodes reach consensus?

The S-BAC Protocol

Byzantine Agreement  Atomic Commit

user

lock  unlock

Shard 1
(manage o1)

Shard 2
(manage o2)

Shard 3
(manage o3)

BFT

BFT

BFT

BFT

BFT

BFT

BFT

BFT

BFT
S-BAC enables Scalability

The Wisdom behind S-BAC

Only shards managing o1 and o2 are reaching consensus

Shard 1 and Shard 2 can work in parallel

user

Shard 1 (manage o1)

Shard 2 (manage o2)

Shard 3 (manage o3)
Contents

.1. System Overview

.2. Privacy by Design

.3. Scalability

.4. Security Properties

.5. Performance
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.

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

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

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

  **Non-Repudiation**
  
  Misbehaviour is detectable: there are evidences of misbehaviour pointing to the faulty parties or shards.
Hash-DAG Structure

- Objects and transactions naturally form a directed acyclic graph (DAG)
- Directed graph: transactions take as input active objects, render them inactive, and create a new set of active output objects
- No cycles: Each object may only be created by a single transaction

**Train seat #33**

<table>
<thead>
<tr>
<th>Object 1</th>
<th>Object 2</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Train seat #33</strong></td>
<td><strong>Train seat #33</strong></td>
</tr>
<tr>
<td>Free</td>
<td>Taken by Alice</td>
</tr>
</tbody>
</table>

Object 1: active
Object 2: nonexistant

**Chainspace transaction**

Object 1: inactive
Object 2: active
Hash-DAG Structure

- Every transaction $T$ has an id
- $id_T$ is $\text{Hash(all input info except outputs)}$
Hash-DAG Structure

- Every transaction $T$ has an id
  - $id_T$ is Hash(all input info except outputs)

- Every object $O$ has an id
  - $id_O$ is Hash($O \| id_T$)
Hash-DAG Structure

- Given $O$, and $id_O$, it is possible to verify all transactions and previous (now inactive) objects and references that contribute to the existence of $O$
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.
    - **Integrity** (Honest Shard)
      Only valid & non-conflicting transactions will be executed.
    - **Encapsulation**
      A smart contract cannot interfere with objects created by another contract (except if defined by that contract).
    - **Non-Repudiation**
      Misbehaviour is detectable: there are evidences of misbehaviour pointing to the faulty parties or shards.
Node Hash-Chains

- Each node in a shard forms a Merkle tree containing all transactions that have been accepted or rejected.
Node Hash-Chains

- Periodically, nodes within a shard consistently agree to seal a checkpoint, as a block of transactions into their hash chains

```
Block 1 ───── Block 2 ───── Block 3
```
Node Hash-Chains

- Periodically, nodes within a shard consistently agree to seal a checkpoint, as a block of transactions into their hash chains.
Node Hash-Chains

- Periodically, nodes within a shard consistently agree to seal a checkpoint, as a block of transactions into their hash chains.
Node Hash-Chains

- Periodically, nodes within a shard consistently agree to seal a checkpoint, as a block of transactions into their hash chains

![Diagram of hash chains]

- Auditing involves re-executing transactions and comparing the result with the hash-chain
Performance

- What did we implement?

S-BAC protocol implemented in Java

Based on BFT-SMaRt
Performance

- What did we implement?

  - 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?

  - S-BAC protocol implemented in Java
  - Python contract simulator
  - Everything is released as open source software

- Helps developers
- Simulation of the checker
- No need for full deployment

- Based on BFT-SMaRt

[https://github.com/chainspace]
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
How the number of shards influences the TPS?

TPS scales linearly with the number of shards.
Performance

How is the trade off between TPS and latency?

![Probability vs Latency](image)

Low latency even when the system is heavy loaded
Chainspace: A Sharded Smart Contracts Platform

Mustafa Al-Bassam1, Alberto Sonnino2, Shehar Bano3, Dave Hrycyszyn1 and George Danezis1
1 University College London, United Kingdom
2 University of Northumbria, Newcastle, United Kingdom
3 University of Edinburgh, United Kingdom

Abstract—Chainspace is a decentralized infrastructure, known as a distributed ledger, that supports user-defined smart contracts and executes user-supplied transactions on their objects. The current execution of smart contract transactions is scalable by all: the system is scalable, by ordering state and the execution of transactions; and scaling, in the sense of guaranteeing integrity, the distributed architecture, the Byzantine Fault Tolerance (BFT), and extremely high-scalability. While BFT fail, ending calculations are to place to reuse multiple participating. We present the design, security, and details of Chainspace, as well through evaluating an implementation of the system about its scaling and other features; we discuss a number of privacy features through modern zero-knowledge techniques.

1. Introduction

Chainspace is a distributed ledger platform for high-security, high-availability, smart contracts and its transaction semantics, to provide higher availability, in the sense of guaranteeing integrity, the distributed architecture, the Byzantine Fault Tolerance (BFT), and extremely high-scalability. While BFT fail, ending calculations are to place to reuse multiple participating. We present the design, security, and details of Chainspace, as well through evaluating an implementation of the system about its scaling and other features; we discuss a number of privacy features through modern zero-knowledge techniques.

Unlike other scalable but permissioned smart contract platforms, such as Hyperledger Fabric [Kach] or Besu [BCCG16], Chainspace aims to be an open system, in which anyone can author a smart contract, or a node. By providing infrastructure on which smart contracts can be executed, we can provide security features through modern zero-knowledge techniques.

Promising to fully optimize all parts of this paper has not been accomplished. For instance, the authors’ work on smart contracts and their security features is based on a blockchain platform named CSCoin, as a system smart contract to allow for accounting between those parties. However, the security model of Chainspace, is different from traditional unpermissioned blockchains, that rely on proof-of-work and global consensus of states, such as Ethereum. In Chainspace, smart contract authors determine the parts of the infrastructure that are trusted to maintain the system’s state, and also the parties that can authorize transactions. This is key to supporting privacy-friendly smart-contracts. The paper makes the following contributions:

- It presents Chainspace, a system that can scale arbitrarily as the number of nodes increases, with Byzantine failures, and is both fully and publicly audited.
- It presents a novel distributed atomic commit protocol, called BFT-SAC, for sharding generic smart contract transactions across multiple Byzantine nodes, and correctly coordinating those nodes to ensure safety, liveness and security properties.
- It introduces a distinction between parts of the smart contract that execute a computation, and those that check the computation and dismiss how that distinction key to supporting privacy-friendly smart-contracts.
- It provides a full implementation and evaluates the performance of the Byzantine distributed atomic commit protocol, BFT-SAC, on a real distributed set of nodes and under varying transaction loads.
- It presents a number of key system and application smart contract and evaluates their performance. The contracts for privacy-friendly smart-contracts and privacy-friendly proofs illustrate and validate support for high-scalability and high-security applications.

Outline: Section II presents an overview of Chainspace; Section III presents the distributed application interface; Section IV presents the design of internal data structures; Section V argues the correctness and security; Section VI presents an evaluation of the core protocols and smart contract performance; Section VII presents limitations of the system; Section VIII presents a per-contract basis, and also allows for horizontal scalability.

This paper makes the following contributions:

- It presents Chainspace, a system that can scale arbitrarily as the number of nodes increases, with Byzantine failures, and is both fully and publicly audited.
- It presents a novel distributed atomic commit protocol, called BFT-SAC, for sharding generic smart contract transactions across multiple Byzantine nodes, and correctly coordinating those nodes to ensure safety, liveness and security properties.
- It introduces a distinction between parts of the smart contract that execute a computation, and those that check the computation and dismiss how that distinction key to supporting privacy-friendly smart-contracts.
- It provides a full implementation and evaluates the performance of the Byzantine distributed atomic commit protocol, BFT-SAC, on a real distributed set of nodes and under varying transaction loads.
- It presents a number of key system and application smart contract and evaluates their performance. The contracts for privacy-friendly smart-contracts and privacy-friendly proofs illustrate and validate support for high-scalability and high-security applications.

Embed this section into the main paper: Section II presents an overview of Chainspace; Section III presents the distributed application interface; Section IV presents the design of internal data structures; Section V argues the correctness and security; Section VI presents an evaluation of the core protocols and smart contract performance; Section VII presents limitations of the system; Section VIII presents a per-contract basis, and also allows for horizontal scalability.

This paper makes the following contributions:

- It presents Chainspace, a system that can scale arbitrarily as the number of nodes increases, with Byzantine failures, and is both fully and publicly audited.
- It presents a novel distributed atomic commit protocol, called BFT-SAC, for sharding generic smart contract transactions across multiple Byzantine nodes, and correctly coordinating those nodes to ensure safety, liveness and security properties.
- It introduces a distinction between parts of the smart contract that execute a computation, and those that check the computation and dismiss how that distinction key to supporting privacy-friendly smart-contracts.
- It provides a full implementation and evaluates the performance of the Byzantine distributed atomic commit protocol, BFT-SAC, on a real distributed set of nodes and under varying transaction loads.
- It presents a number of key system and application smart contract and evaluates their performance. The contracts for privacy-friendly smart-contracts and privacy-friendly proofs illustrate and validate support for high-scalability and high-security applications.

Outline: Section II presents an overview of Chainspace; Section III presents the distributed application interface; Section IV presents the design of internal data structures; Section V argues the correctness and security; Section VI presents an evaluation of the core protocols and smart contract performance; Section VII presents limitations of the system; Section VIII presents a per-contract basis, and also allows for horizontal scalability.

This paper makes the following contributions:
Future Work

1. How to recover from malicious shards?

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

4. How to incentivise nodes?
Conclusions

**contribution I**

Scalable smart contract platform

sharding

**contribution II**

Supporting privacy

execution / checker
Thanks
Q/A

@thatBano
s.bano@ucl.ac.uk
http://sheharbano.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.