A set of nodes
... Some of which are bad
1. make transaction
Blockchains

1. make transaction

2. submit transaction
Blockchains

1. make transaction

2. submit transaction

3. sequence and verify
Blockchains

1. make transaction
2. submit transaction
3. sequence and verify
4. store
Low throughput
High latency
Slow finality
Poor privacy
Chainspace & Byzcuit

A scalable backbone with integrated privacy support
Chainspace
State sharding

shard 1

shard 2

shard 3
Byzcuit
Cross-shard consensus protocol
Privacy by Design

procedure

checker
Privacy by Design

procedure
(make zk-proof)

checker
(verify zk-proof)
High throughput
Low latency
Fast finality
Good privacy
FastPay

A low-latency payment system
What we have so far

Total Latency:
slowest shard during phase 1
+ 
slowest shard during phase 2
+ 
all communications
Overview

Chainspace

Byzcuit

FastPay
Difference with blockchains

Blockchains

Byzantine Consensus

FastPay

Byzantine Consistent Broadcast
High throughput
Low latency
Fast finality
Good privacy
Coconut

Privacy-preserving credentials for smart contract applications
Coconut
Anonymous credentials in a blockchain setting

Coconut

Chainspace

Byzcuit

attributes
Coconut
Anonymous credentials in a blockchain setting
High throughput
Low latency
Fast finality
Good privacy
15. Fraud Proofs: Maximising Light Client Security and Scaling Blockchains with Dishonest Majorities  
Mustafa Al-Bassam, Alberto Sonnino, Vitalik Buterin, Ismail Khoffi, Financial Cryptography and Data Security (FC), 2021

14. EL PASSO: Privacy-preserving, Asynchronous Single Sign-On  
Zhiyi Zhang, Michał Król, Alberto Sonnino, Lixia Zhang, Etienne Rivière, International Symposium on Privacy Enhancing Technologies (PETs), 2021

Shehar Bano, Alberto Sonnino, Andrey Chursin, Dmitri Perelman, Zekun Li, Avery Ching, Dahlia Malkhi, ArXiv Preprint, 2020

12. FastPay: High-Performance Byzantine Fault Tolerant Settlement  
Mathieu Baudet, George Danezis, Alberto Sonnino, ACM Conference on Advances in Financial Technologies (AFT), 2020

11. Replay Attacks and Defenses against Cross-shard Consensus in Sharded Distributed Ledgers  
Alberto Sonnino, Shehar Bano, Mustafa Al-Bassam, George Danezis, IEEE European Symposium on Security and Privacy (EuroS&P), 2020

10. PASTRAMI: Privacy-preserving, Auditable, Scalable & Trust-worthy Auctions for Multiple Items.  
Michał Król, Alberto Sonnino, Argyrios G. Tasiopoulos, Ioannis Psaras, Etienne Rivièrè ACM/IFIP Middleware, 2020

9. Location, location, location: Revisiting Modeling and Exploitation for Location-based Side Channel Leakages  
Christos Andrikos, Lejla Batina, Lukasz Chmielewski, Liran Lerman, Vasillios Mavroudis, Kostas Papagiannopoulos, Guilherme Perin, Giorgos Rassias, Alberto Sonnino, Asiacrypt, 2019

8. FMPC: Secure Multiparty Computation from Fourier Series and Parseval’s Identity  
Alberto Sonnino, ArXiv Preprint, 2019

7. AStERISK: Auction-based Shared Economy Resolution System for blockchain  
Alberto Sonnino, Michał Król, Argyrios Tasiopoulos, Ioannis Psaras, Workshop on Decentralized IoT Systems and Security (DISS), 2019

6. SybilQuorum: Open Distributed Ledgers Through Trust Networks  
Alberto Sonnino, George Danezis, ArXiv Preprint, 2019

5. Proof-of-Prestige: A Useful Work Reward System for Unverifiable Tasks  
Michał Król, Alberto Sonnino, Mustafa Al-Bassam, Argyrios Tasiopoulos, Ioannis Psaras, IEEE International Conference on Blockchain and Cryptocurrency (ICBC), 2019

4. Coconut: Threshold Issuance Selective Disclosure Credentials with Applications to Distributed Ledgers  
Alberto Sonnino, Mustafa Al-Bassam, Shehar Bano, Sarah Meiklejohn, George Danezis, Proceedings of the Network and Distributed System Security Symposium (NDSS), 2019

3. SoK: Consensus in the Age of Blockchains  
Shehar Bano, Alberto Sonnino, Mustafa Al-Bassam, Sarah Azouvi, Patrick McCorry, Sarah Meiklejohn, George Danezis, ACM Conference on Advances in Financial Technologies (AFT), 2019

2. Airtnt: Fair Exchange Payment for Outsourced Secure Enclave Computations  
Mustafa Al-Bassam, Alberto Sonnino, Michał Król, Ioannis Psaras, ArXiv preprint, 2018

1. Chainspace: A Sharded Smart Contracts Platform  
Mustafa Al-Bassam, Alberto Sonnino, Shehar Bano, Dave Hrycyszyn, George Danezis, Proceedings of the Network and Distributed System Security Symposium (NDSS), 2018
EXTRA
State Sharding
Traditional

Sharding
State Sharding
An example transaction

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
State Sharding
An example transaction

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
State Sharding

Only two acceptable final states

- Shard 1
- Shard 2
- Shard 3

- X₁
- X₂
- Y₁
- Y₂
- Y₃
Cross-Shard Consensus
How do shards communicate with each other?
S-BAC Attacks
Attacks

Double spend any object

- Does not need to collude with any node
- Acts as client or passive observer
- Re-orders network messages (not always needed)
Attack against S-BAC

Double-spend $X_1$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$
Attack against S-BAC

Double-spend $X_1$

$T'(\widehat{x_1}, x_2) \rightarrow (y_1, y_2, y_3)$
Attack against S-BAC

Double-spend $X_1$

$T'(\tilde{x}_1, x_2) \rightarrow (y_1, y_2, y_3)$
Attack against S-BAC
Double-spend $X_1$

$T'(\overline{x}, x_2) \rightarrow (y_1, y_2, y_3)$

c
s1
BFT
s2
BFT
s3

lock $X_2$
Attack against S-BAC

Double-spend $X_1$

$T'(\overline{x_1}, x_2) \rightarrow (y_1, y_2, y_3)$

c

BFT

pre-abort($T'$)

s1

BFT

pre-accept($T'$)

s2

lock $X_2$

s3
Attack against S-BAC

Double-spend $X_1$

\[ T'(\overline{x}_1, x_2) \rightarrow (y_1, y_2, y_3) \]

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]

\[ \text{lock } X_2 \]
Attack against S-BAC
Double-spend $X_1$

$T'(\tilde{x}_1, x_2) \rightarrow (y_1, y_2, y_3)$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$

$\text{lock } X_2$

$c$

$s_1$

$s_2$

$s_3$

BFT

pre-accept($T$)

pre-abort($T'$)

pre-accept($T$)

from shard 1

BFT

BFT

BFT

BFT

BFT

BFT

abort($T$)
Attack against S-BAC

Double-spend $X_1$

$T'(\tilde{x}_1, x_2) \rightarrow (y_1, y_2, y_3)$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$

lock $X_2$

pre-accept($T'$) from shard 1

unlock $X_2$

pre-accept($T$)

pre-abort($T'$)

pre-accept($T$)

pre-abort($T$)

abort($T'$)

abort($T$)
Attack against S-BAC

Double-spend $X_1$

$T^*(x_1) \rightarrow (y^*_x)$

c client

shard 1

BFT

10
Attack against S-BAC
Double-spend $X_1$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$

client

shard 1

BFT

pre-abort(T)

shard 2

BFT

5

pre-accept(T)

shard 3

attacker
Attack against S-BAC

Double-spend $X_1$

$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$

client

shard 1

shard 2

shard 3

attacker
Double-spend $X_1$

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
**Attack against S-BAC**

Double-spend $X_1$

**Before attack**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$X_1$</td>
<td>10</td>
</tr>
<tr>
<td>$X_2$</td>
<td>5</td>
</tr>
</tbody>
</table>

**After attack**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>$Y^*$</td>
<td>10</td>
</tr>
<tr>
<td>$Y_2$</td>
<td>4</td>
</tr>
<tr>
<td>$Y_3$</td>
<td>10</td>
</tr>
</tbody>
</table>
What causes these issues?

**Issue 1.** Input shards cannot associate protocol messages to a specific protocol execution.

**Issue 2.** Output shards (that are not also input shards) do not experience the first phase of the protocol.
EXTRA
Atomix Attacks
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]

client

shard 1

shard 2

shard 3
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
$T(x_1, x_2) \rightarrow (y_1, y_2, y_3)$

delete $X_1, X_2$; create $Y_1, Y_2$
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
S-BAC

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
Atomix

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]

client

shard 1

shard 2

shard 3

BFT

inactivate X₁, X₂
Atomix

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
Atomix

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
Atomix

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]

Client

Shard 1

Shard 2

Shard 3

create \( Y_1 \)

create \( Y_2 \)

create \( Y_3 \)
Atomix

\[ T(x_1, x_2) \rightarrow (y_1, y_2, y_3) \]
Cross-Shard Consensus
How does it achieve linear scalability?
Byzcuit
Fix issue 1

Add sequence numbers per object
Byzcuit
Fix issue 2

Dummy objects for output shards

$X_1, S_{x_1}$
Shard 1

$X_2, S_{x_2}$
Shard 2

$D_3, S_{D_3}$
Shard 3
Byzcuit

\[ \{ S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3) \} \]
Byzcuit

\[ \{ S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3) \} \]

- **Check 1.** Are all inputs active / transaction well formed?

- **Check 2.** Is the sequence number \( S_T \) well formed?
  \[ S_T \geq \max\{S_{X1}, S_{X2}\} \]
Byzcuit

\[ \{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\} \]

client ➔ shard 1 ➔ shard 2 ➔ shard 3 ➔ TM

- If checks fail: \( S_{X1} \leftarrow S_T + 1 \)
- If checks fail: \( S_{X2} \leftarrow S_T + 1 \)
- If checks fail: \( S_{D3} \leftarrow S_T + 1 \)
Byzcuit

\[
\{ S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3) \}
\]
Byzcuit

\[ \{ S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3) \} \]
Byzcuit

\[ \{ S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3) \} \]
Byzcuit

\[ \{S_T, T(x_1, x_2, d_3) \rightarrow (y_1, y_2, y_3)\} \]

if (T, ST), inactivate X_1, X_2, D_3
create Y_1, Y_2, Y_3
Why is Byzcuit secure?

**Issue 1.** Input shards cannot associate protocol messages to a specific protocol execution.

Sequence numbers:
act as session ID

**Issue 2.** Output shards (that are not also input shards) do not experience the first phase of the protocol.

Dummy objects:
all shards experience the first phase of the protocol
EXTRA
FastPay Transfer
FastPay
How does it work?

sender
recipient
FastPay
How does it work?

1. transfer order
FastPay
How does it work?

1. transfer order
2. verify
FastPay
How does it work?

1. transfer order

sender

2. verify

recipient

3. signed transfer order
FastPay
How does it work?

1. transfer order
2. verify
3. signed transfer order
4. confirmation order
5. confirmation order
6. confirmation order
FastPay
How does it work?

1. transfer order
2. verify
3. signed transfer order
4. confirmation order
5. confirmation order
6. confirmation order
7. update
FastPay
Increasing capacity

1. 
2. 
3. 
4. 
5. 
6. 
7. 

[Diagram of FastPay system showing increasing capacity]
Byzantine Consistent Broadcast

Validity
No duplication
Integrity
Consistency
FastPay Smart Contract Interface
FastPay
From primary infrastructure to FastPay

1. funding transaction
FastPay
From primary infrastructure to FastPay

1. funding transaction

sender

smart contract

2. synchronization order
FastPay
From primary infrastructure to FastPay

1. funding transaction
2. synchronization order
3. verify & update
FastPay
From the primary infrastructure to FastPay

1. transfer order
FastPay
From the primary infrastructure to FastPay

1. transfer order

sender

2. verify

smart contract
FastPay
From the primary infrastructure to FastPay

1. transfer order
2. verify
3. signed transfer order
FastPay
From the primary infrastructure to FastPay

1. transfer order

2. verify

3. signed transfer order

4. confirmation order

sender

smart contract
FastPay
From the primary infrastructure to FastPay

1. transfer order

2. verify

3. signed transfer order

4. confirmation order

5. update

sender

smart contract
FastPay
From the primary infrastructure to FastPay

1. transfer order
2. verify
3. signed transfer order
4. confirmation order
5. update
6. redeem transaction
7. verify & update