Jigsaw: Doubly Private Smart Contracts

Sanjam Garg

UC Berkeley

Aarushi Goel

Purdue University

Dimitris Kolonelos

UC Berkeley

Rohit Sinha

Swirlds Labs

Workshop on Cryptographic Tools for Blockchains 2025, Madrid
3 May 2025

Outline

Background

Our Contributions

Jigsaw

Conclusions and Open Problems

Background

Smart Contract Platforms

Input data

x

Smart Contract

f

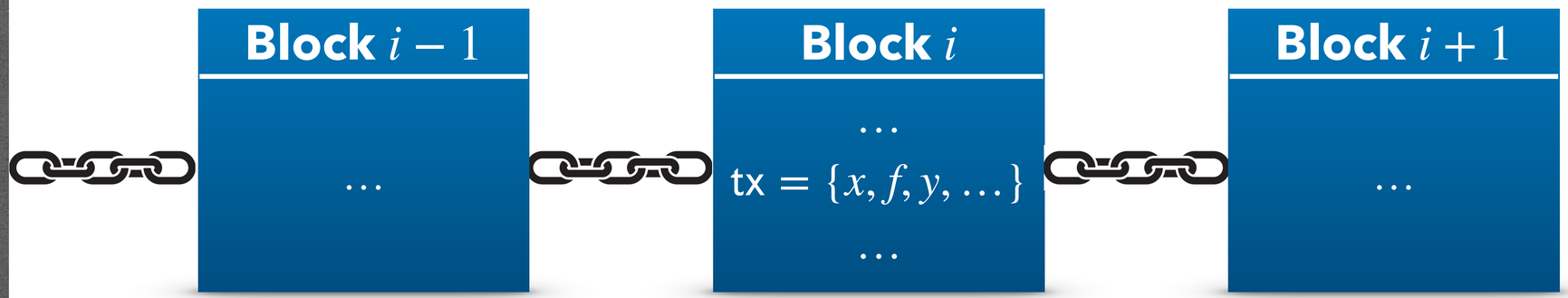
```
1   // SPDX-License-Identifier: MIT
2   pragma solidity ^0.8.0;
3
4   contract SimpleStorage {
5       uint private storedNumber;
6
7       // Function to set a number
8       function setNumber(uint _num) public {
9           storedNumber = _num;
10      }
11
12      // Function to get the stored number
13      function getNumber() public view returns (uint) {
14          return storedNumber;
15      }
16  }
17
```

Output data

$y = f(x)$

Decentralized Computing Machine

Blockchain



Example – DEX

$x_1 = \{2$ **BTC**, BTC↔ETH, 1:30$\}$

Smart Contract $f =$ **DEX**

$y_{1,1} = \{$ 60 **ETH** $\}$

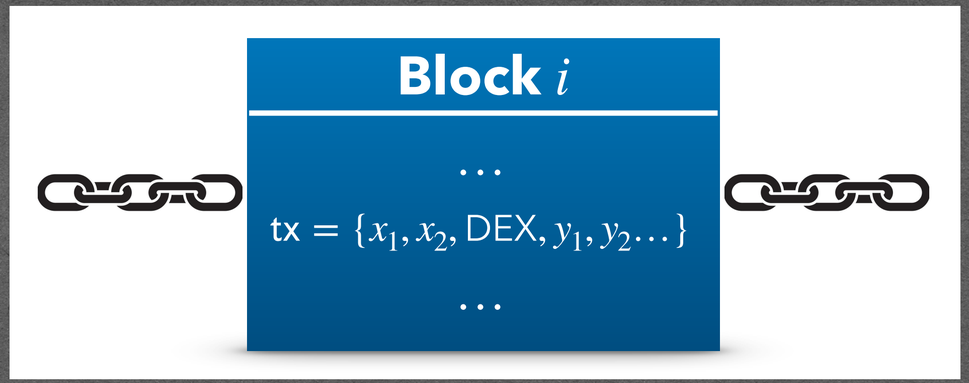
$y_{1,2} = \{$ 0 **BTC** $\}$

tx

$y_{1,2} = \{$ 2 **BTC** $\}$

$y_{2,2} = \{$10 **ETH** $\}$

$x_2 = \{70$ **ETH**, ETH↔BTC, 30:1$\}$



Privacy Leakage

Everything is public on Blockchain!

- Computation (functions)
- Data (input data, output data)

Privacy Leakage

Everything is public on Blockchain!

- Computation (functions)
- Data (input data, output data)

Well understood issue, with real-world de-anonymization attacks:

Academic:

Quantitative Analysis of the Full Bitcoin Transaction Graph

Dorit Ron and Adi Shamir

A Fistful of Bitcoins: Characterizing Payments Among Men with No Names

Sarah Meiklejohn Marjori Pomarole Grant Jordan
Kirill Levchenko Damon McCoy† Geoffrey M. Voelker Stefan Savage

Evaluating User Privacy in Bitcoin

Elli Androulaki[1], Ghassan O. Karame[2], Marc Roeschlin[1], Tobias Scherer[1], and Srdjan Capkun[1]

How to Peel a Million: Validating and Expanding Bitcoin Clusters

George Kappos[1], Haaroon Yousaf[1], Rainer Stütz[2], Sofia Rollet[2], Bernhard Haslhofer[3], and Sarah Meiklejohn[1]

Industry:

Chainalysis Crystal TRM ELLIPTIC

Privacy-Preserving Smart Contracts (PPSC)

zkSNARKs

Hawk [KMS+16], ZEXE [BCG+20], VERIZEXE [XCZ+23],

zkay [SBG+19], Zapper [SBV22], …

MPC (+zkSNARKs)

zkHAWK [BCT21], V-zkHAWK [BT22], Eagle [ByCDF23],…

FHE (+zkSNARKs)

Zeestar [SBBV22], SmartFHE [SWA23],…

TEE

Arbitrum [KGC+18], Ekiden [CZK+19],…

Privacy-Preserving Smart Contracts (PPSC)

zkSNARKs

Hawk [KMS+16], ZEXE [BCG+20], VERIZEXE [XCZ+23],

zkay [SBG+19], Zapper [SBV22], …

MPC (+zkSNARKs)

zkHAWK [BCT21], V-zkHAWK [BT22], Eagle [ByCDF23],…

FHE (+zkSNARKs)

Zeestar [SBBV22], SmartFHE [SWA23],…

TEE

Arbitrum [KGC+18], Ekiden [CZK+19],…

Specific Applications: Zerocash [BSCG+14] (Transactions), P2DEX [BDF21] (DEX),

Ratel [LSH+24] (MEV Prevention), …

Privacy-Preserving Smart Contracts (PPSC)

zkSNARKs

Hawk [KMS+16], ZEXE [BCG+20], VERIZEXE [XCZ+23],

zkay [SBC+19], Zapper [SBV22], …

This Work

MPC (+zkSNARKs)

zkHAWK [BCT21], V-zkHAWK [BT22], Eagle [ByCDF23],…

FHE (+zkSNARKs)

Zeestar [SBBV22], SmartFHE [SWA23],…

TEE

Arbitrum [KGC+18], Ekiden [CZK+19],…

Specific Applications: Zerocash [BSCG+14] (Transactions), P2DEX [BDF21] (DEX),

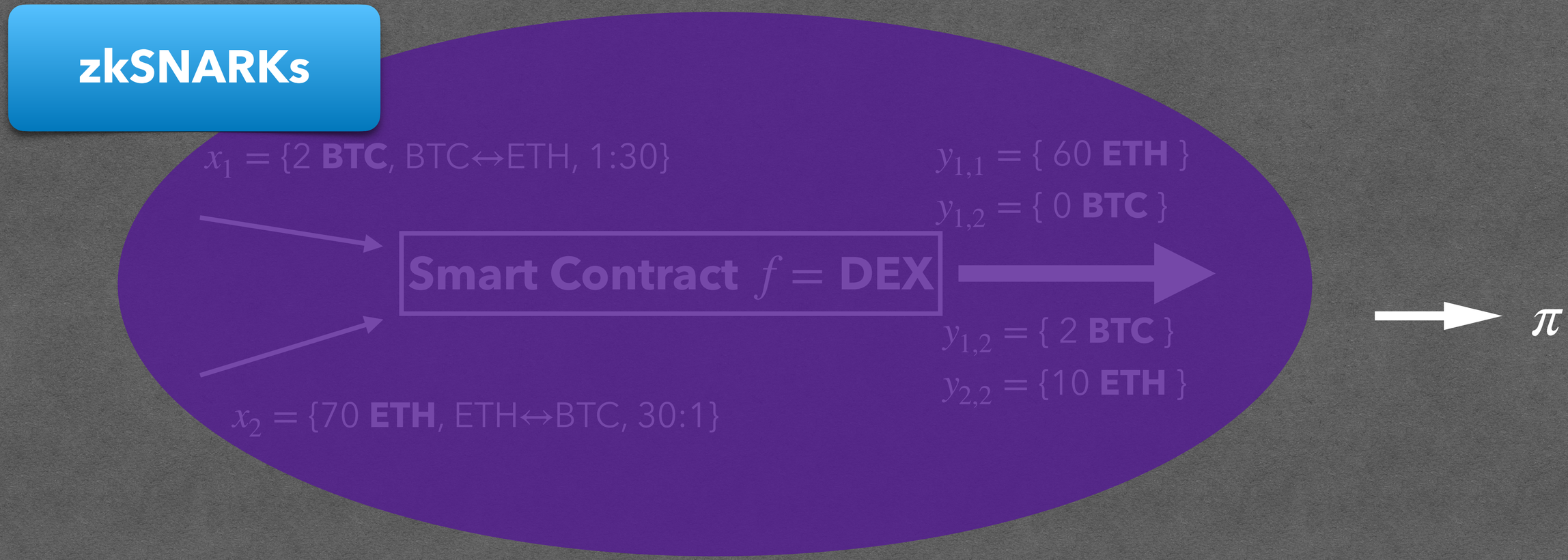
Ratel [LSH+24] (MEV Prevention), …

zkSNARK-based PPSC

Hawk [KMS+16], ZEXE [BCG+20], Zapper [SBV22], …

zkSNARK-based PPSC

Hawk [KMS+16], ZEXE [BCG+20], Zapper [SBV22], …

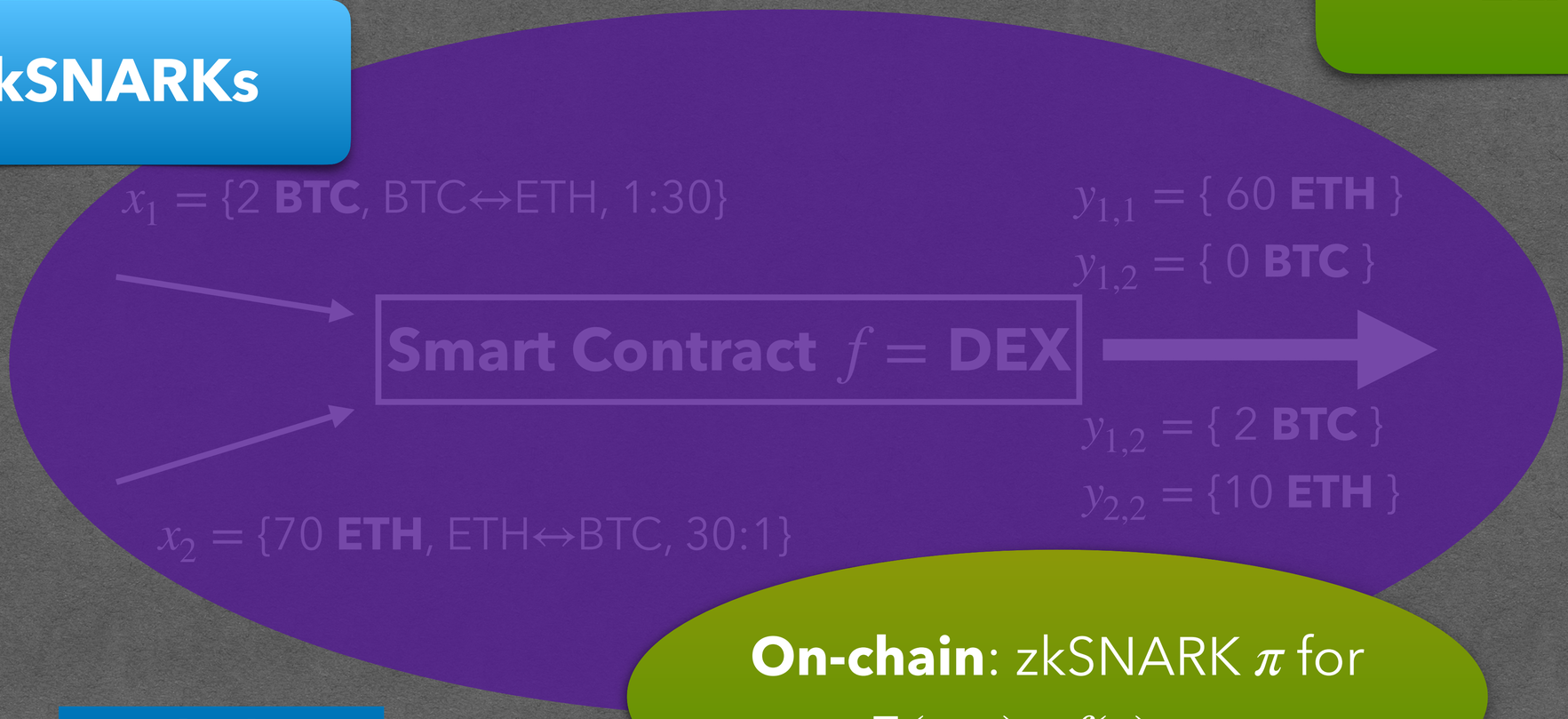


zkSNARK-based PPSC

Hawk [KMS+16], ZEXE [BCG+20], Zapper [SBV22], …

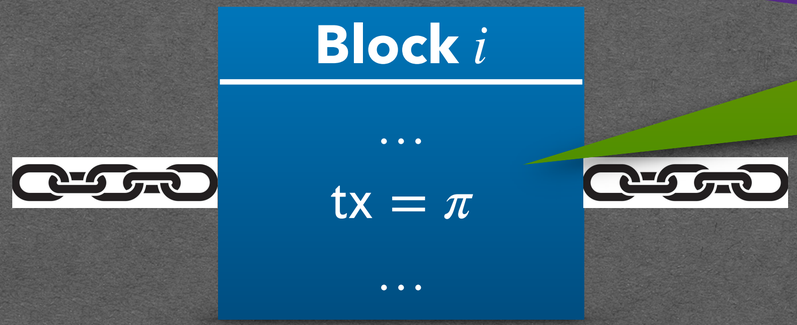
Hawk, Zapper: f public
ZEXE: f private

zkSNARKs



π

On-chain: zkSNARK π for
$\exists(x, y) : f(x) = y$



Off-Chain Privacy Leak

Off-Chain

On-Chain

$x_1 = \{2\ \textbf{BTC}, \text{BTC}\leftrightarrow\text{ETH}, 1{:}30\}$

$x_2 = \{70\ \textbf{ETH}, \text{ETH}\leftrightarrow\text{BTC}, 30{:}1\}$

$\text{Prove}(\text{DEX}, x_1, x_2, y_1, y_2)$

π

Private

Off-Chain Privacy Leak

Off-Chain

On-Chain

$x_1 = \{2$ **BTC**, BTC↔ETH, 1:30$\}$

$x_2 = \{70$ **ETH**, ETH↔BTC, 30:1$\}$

Prove(DEX, x_1, x_2, y_1, y_2)

π

Private

Off-Chain Privacy Leak

- A trusted off-chain entity learns the data.
- Non-affordable in applications: Trading, Auctions, DeFi,…

Our Contributions

Our Contributions

- **Doubly Private Smart Contracts** (DPSC) Framework
- **Jigsaw**: Cryptographic Construction of DPSC
- **Implemention**: <3s off-chain, 40-50x faster
- **Applications**

Doubly Private Smart Contracts Framework

Clients

Servers

(Privacy Provider Service)

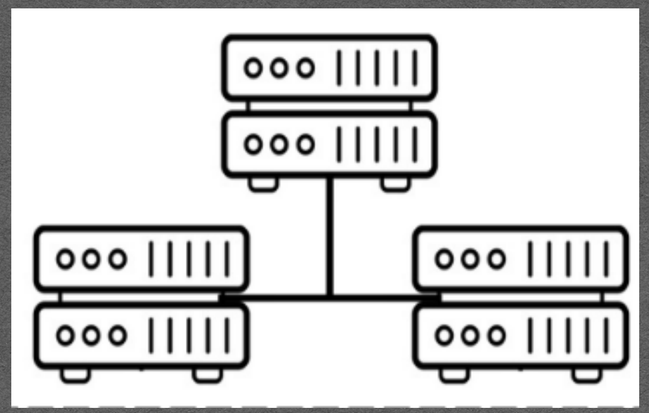
Blockchain

x_1

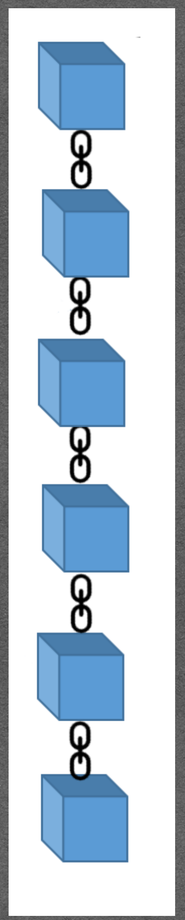
x_2

x_3

x_4



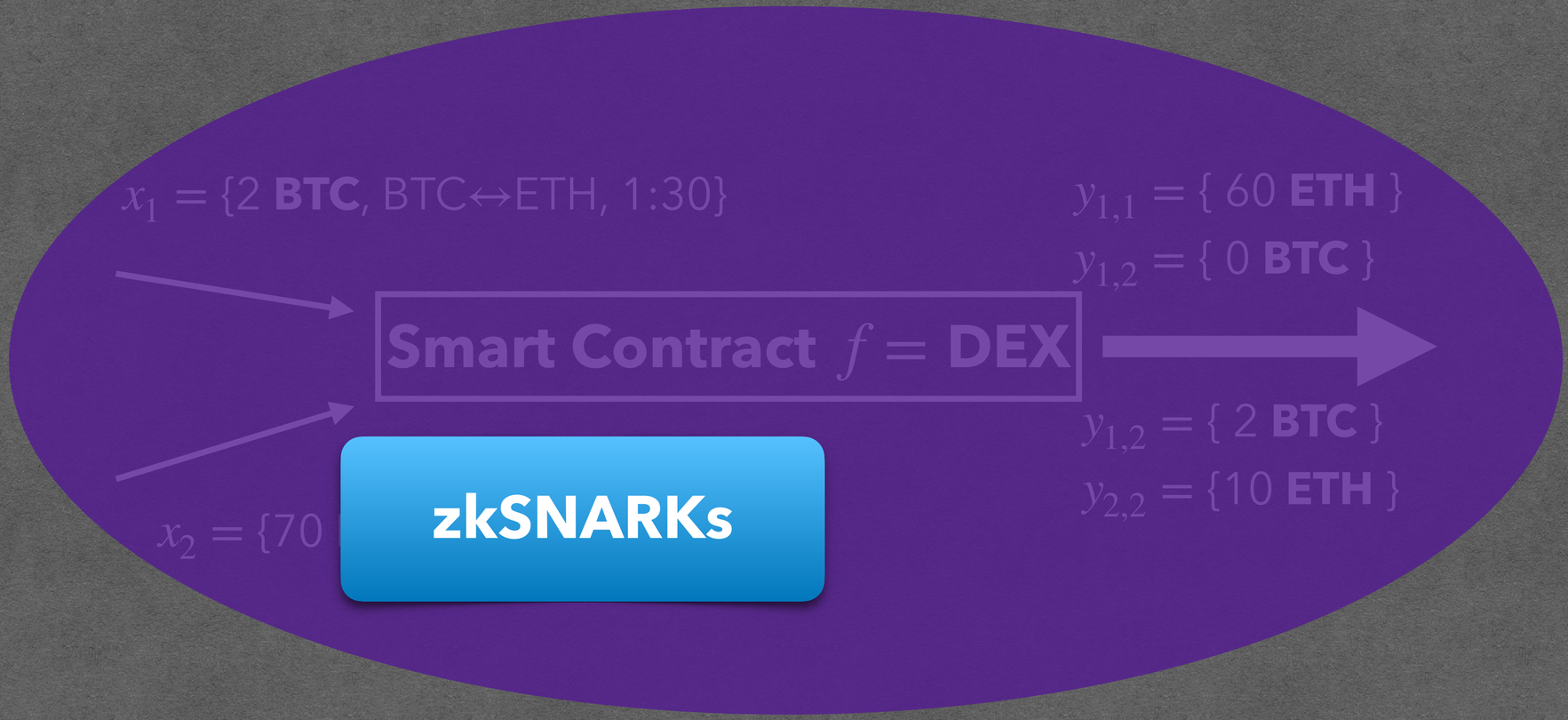
tx



1. **Integrity**
2. **Fire-and-Forget**
3. **Anonymity**
4. **Off-Chain Privacy**

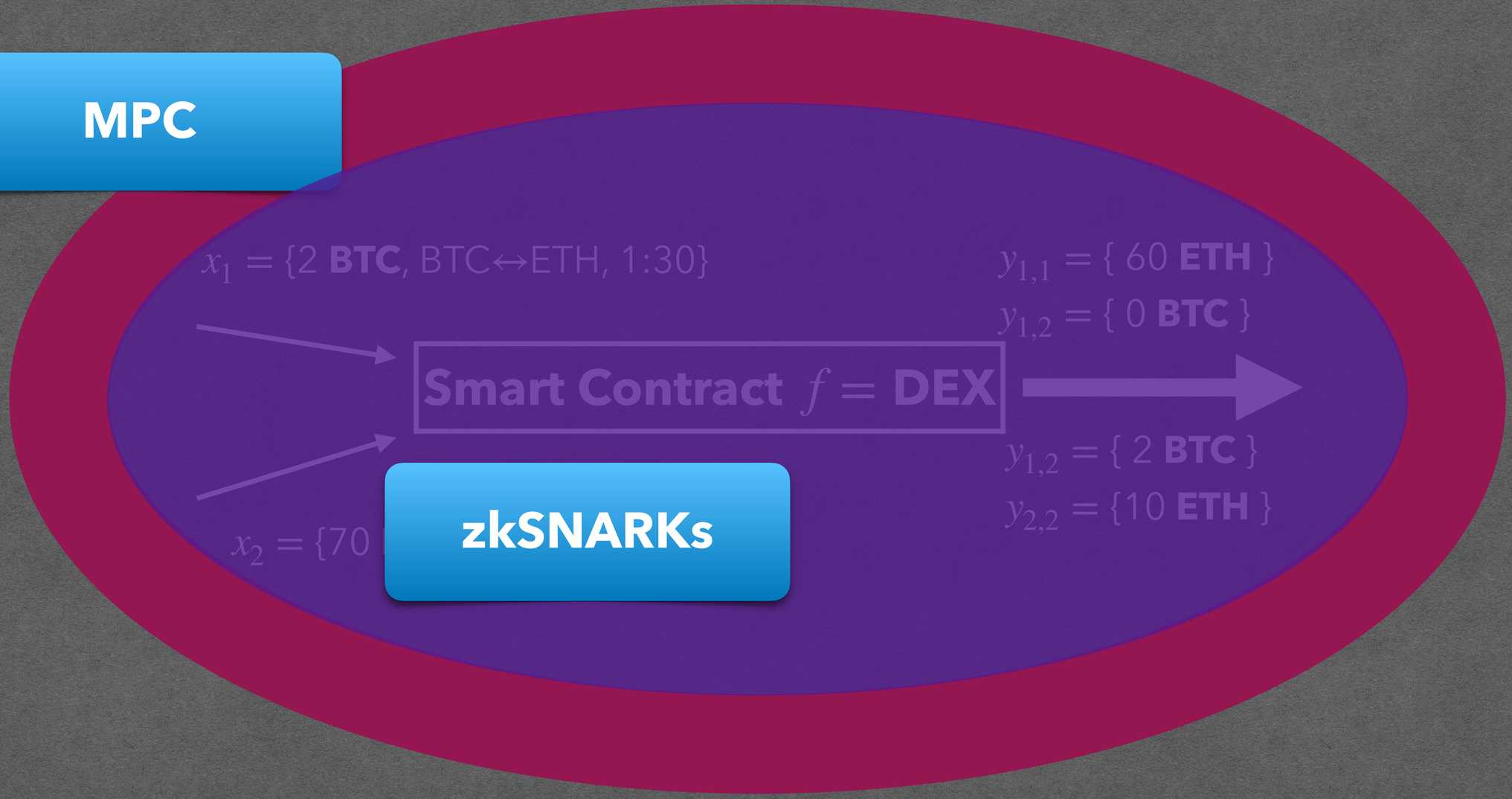
Our Cryptographic Approach

Add another layer of privacy: MPC over zkSNARKs



Our Cryptographic Approach

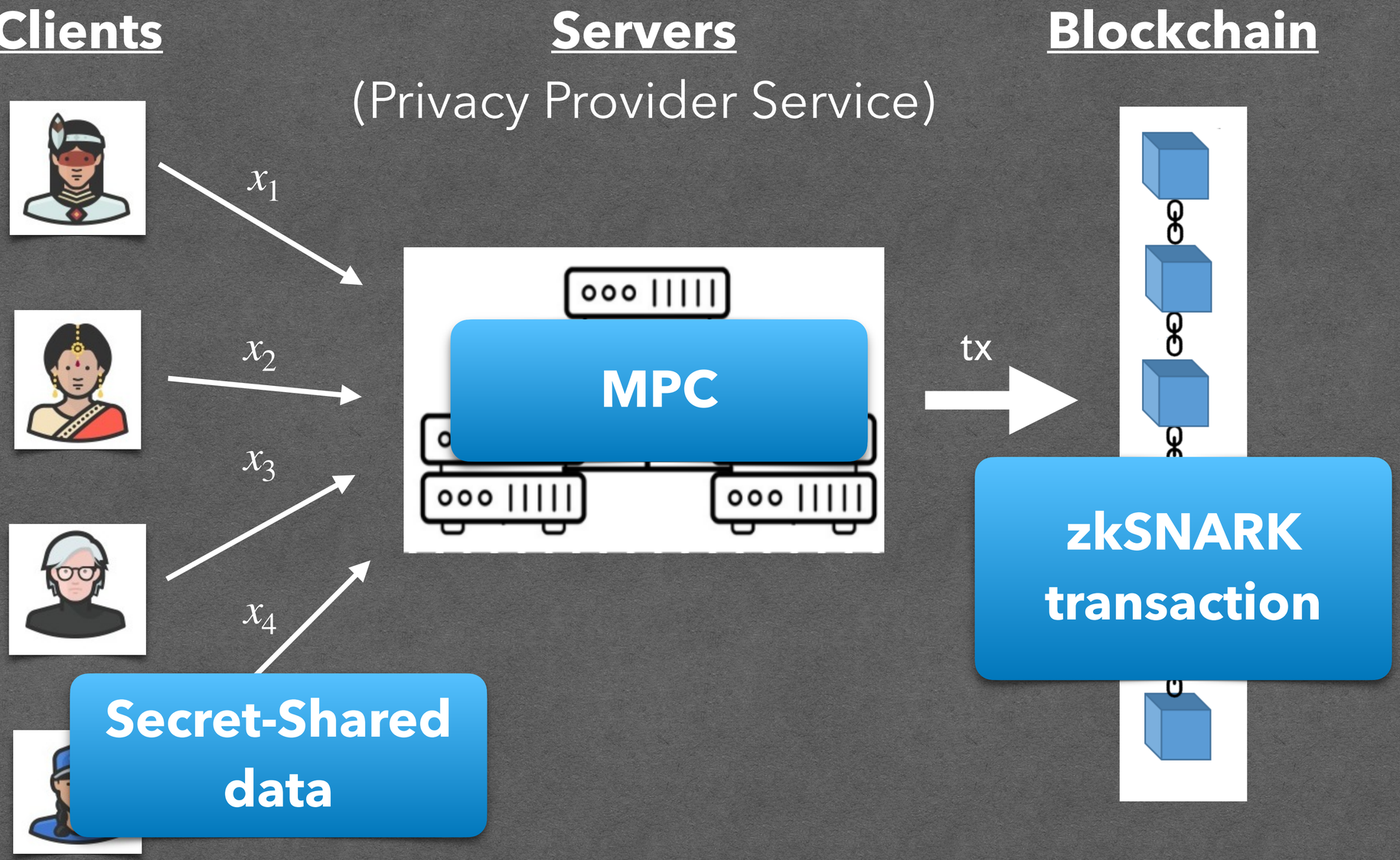
Add another layer of privacy: MPC over zkSNARKs



Jigsaw

Jigsaw from a bird's-eye view

Outsourcing in a privacy-preserving manner a zkSNARK computation



Jigsaw from a bird's-eye view

Outsourcing in a privacy-preserving manner a zk[illegible]tion

Clients

Server

(Privacy Provide[illegible]

[illegible]in

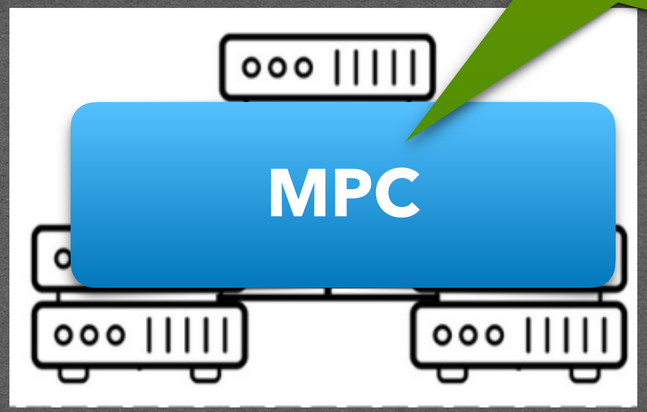
x_1

x_2

x_3

x_4

Secret-Shared data



Privacy Servers in place of trusted off-chain entity

tx

zkSNARK transaction

Zcash/ZEXE architecture

Jigsaw Architecture

Zcash/ZEXE data structures [BSCGGMTV14]

- **Record:** $\mathbf{r} = (\text{cm}, \text{apk}, \text{payload}, \text{sn}, \ldots)$

  $\text{cm} = \text{Com}(\text{apk}, \text{payload}, \ldots)$

  $\text{sn} = \text{PRF}_{\text{sk}}(\mathbf{r})$

- **Blockchain state:** $\text{root} = \text{MerkleCom}(\text{cm}_1, \text{cm}_2, \ldots, \text{cm}_n)$

- **Transaction:** $\text{tx} = (\text{sn}_{spent}, \text{cm}_{new}, \pi, f)$

π **zkSNARK for: (1)** sn_{spent} **valid** ($\text{cm}_{spent} \in \text{root}$**,...)**

(2) cm_{new} **well formed**

(3) $f(\text{payload}_{old}, \text{payload}_{new}) = 1$

Jigsaw Architecture

Zcash/ZEXE data structures [BSCGGMTV14]

- **Record:** $\mathbf{r} = (\text{cm}, \text{apk}, \text{payload}, \text{sn}, \ldots)$

  $\text{cm} = \text{Com}(\text{apk}, \text{payload}, \ldots)$

  $\text{sn} = \text{PRF}_{\text{sk}}(\mathbf{r})$

- **Blockchain state:** $\text{root} = \text{MerkleCom}(\text{cm}_1, \text{cm}_2, \ldots, \text{cm}_n)$

- **Transaction:** $\text{tx} = (\text{sn}_{spent}, \text{cm}_{new}, \pi, f)$

*__ZEXE__ also hides f

π **zkSNARK for: (1)** sn_{spent} **valid** ($\text{cm}_{spent} \in \text{root}$**,...)**

(2) cm_{new} **well formed**

(3) $f(\text{payload}_{old}, \text{payload}_{new}) = 1$

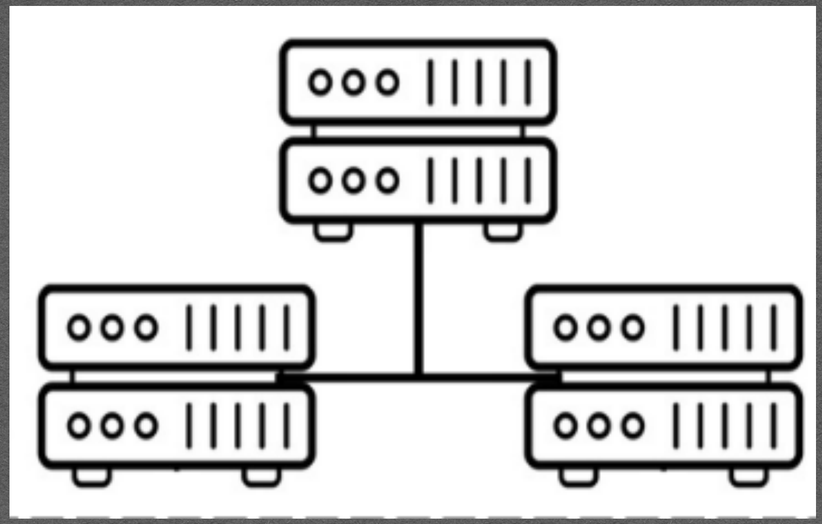
Example – DEX

$\mathbf{r}_1 = (\ldots(2,\textbf{BTC})\ldots)$

BTC↔ETH

1:30

Smart Contract f = DEX



$\tilde{\mathbf{r}}_{1,1} = (\ldots(60,\textbf{ETH})\ldots)$

$\tilde{\mathbf{r}}_{1,2} = (\ldots(0,\textbf{BTC})\ldots)$

tx

$\mathbf{r}_2 = (\ldots(70,\textbf{ETH})\ldots)$

ETH↔BTC

30:1

$\tilde{\mathbf{r}}_{2,1} = (\ldots(2,\textbf{BTC})\ldots)$

$\tilde{\mathbf{r}}_{2,2} = (\ldots(10,\textbf{ETH})\ldots)$

$\text{tx} = (\text{sn}_1,\ \text{sn}_2,\ \widetilde{\text{cm}}_{1,1},\ \widetilde{\text{cm}}_{1,2},\ \widetilde{\text{cm}}_{2,1},\ \widetilde{\text{cm}}_{2,2},\ \pi,\ \textbf{DEX})$

Challenges

1. **Interaction:** Output records computed by the Servers → Clients have to come back for their secret keys.

2. **Efficiency:** How does an MPC compute a zkSNARK?

Challenges

1. **Interaction:** Output records computed by the Servers → Clients have to come back for their secret keys.

Homomorphic commitments: Clients pre-generate dummy records with their secret keys – Servers 'correct' them homomorphically and post them on-chain

2. **Efficiency:** How does an MPC compute a zkSNARK?

Challenges

1. **Interaction:** Output records computed by the Servers → Clients have to come back for their secret keys.

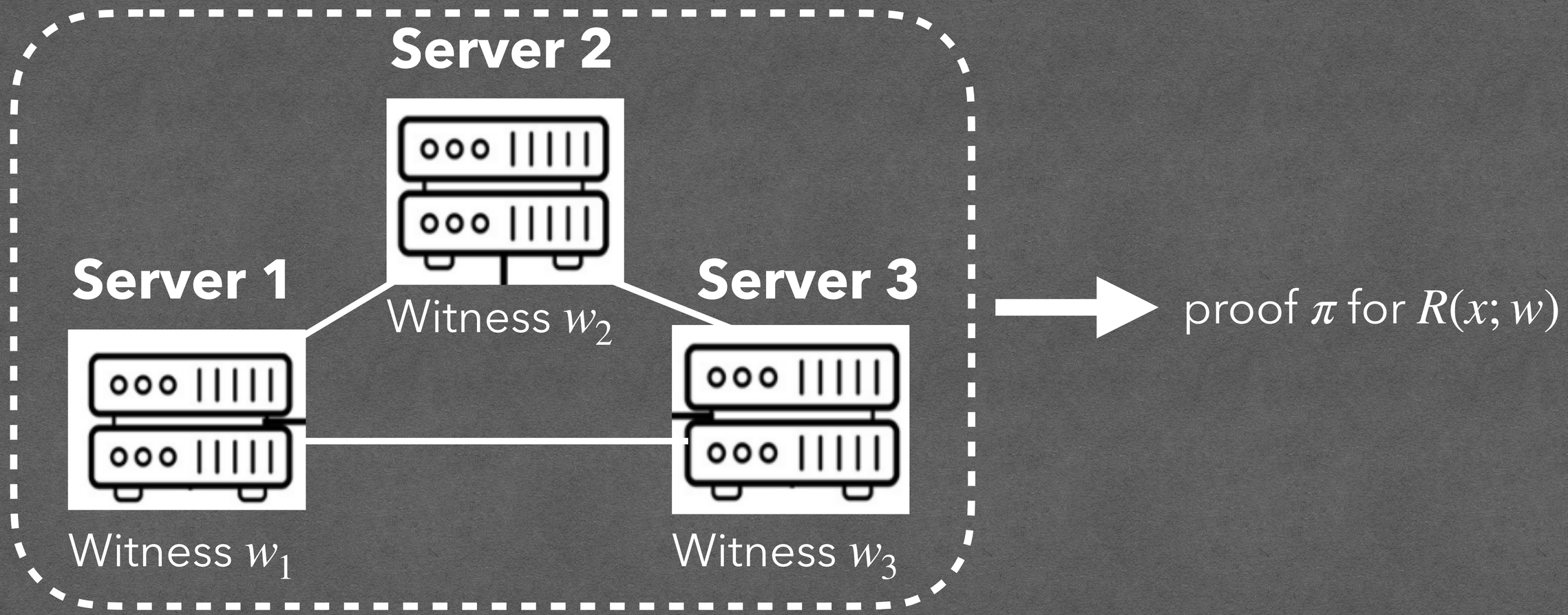
Homomorphic commitments: Clients pre-generate dummy records with their secret keys – Servers 'correct' them homomorphically and post them on-chain

2. **Efficiency:** How does an MPC compute a zkSNARK?

Collaborative zkSNARKs

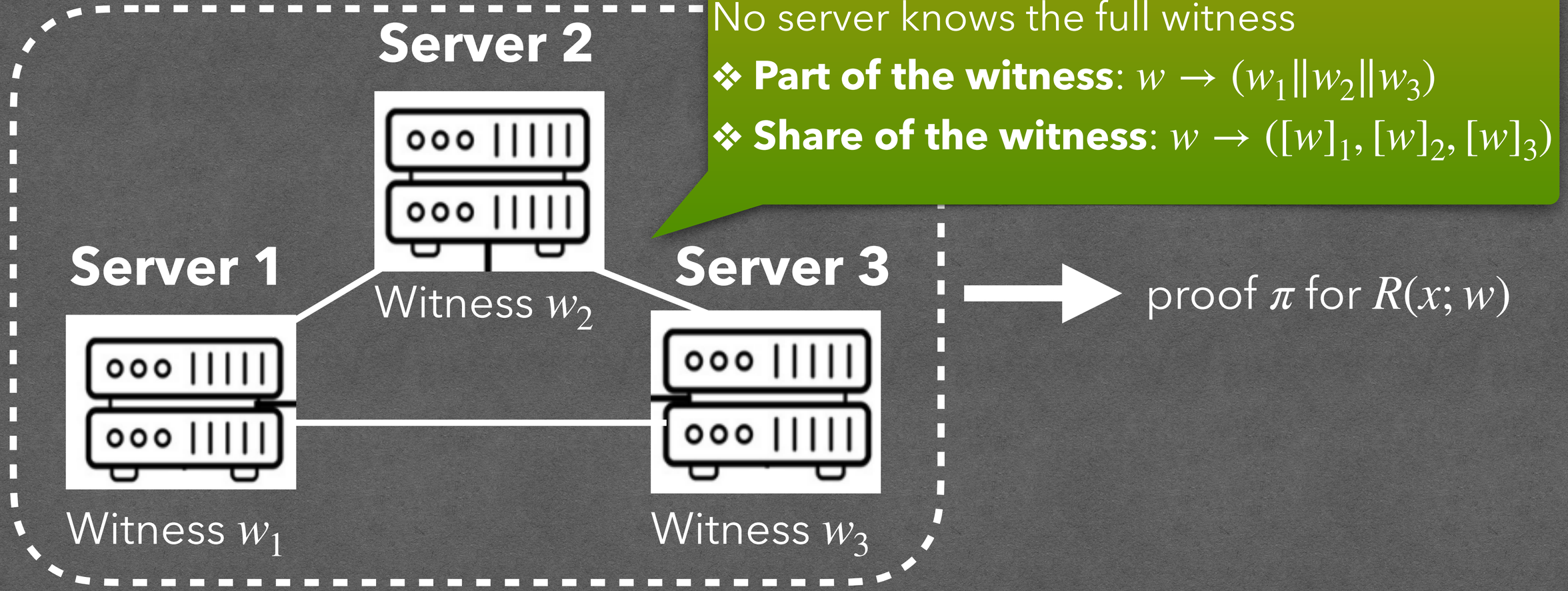
Collaborative zkSNARKs [OB22]

Collaborative zkNARKs: Efficient MPC for a zkSNARK Prover



Collaborative zkSNARKs [OB22]

Collaborative zkNARKs: Efficient MPC for a zkSNARK Prover



Collaborative zkSNARKs – Efficiency Limitations

1. Each Server's i computation is proportional to $\|w\|$ not $\|w_i\|$
2. Communication overhead: Multiplication depth is of essence

Traditional zkSNARKs

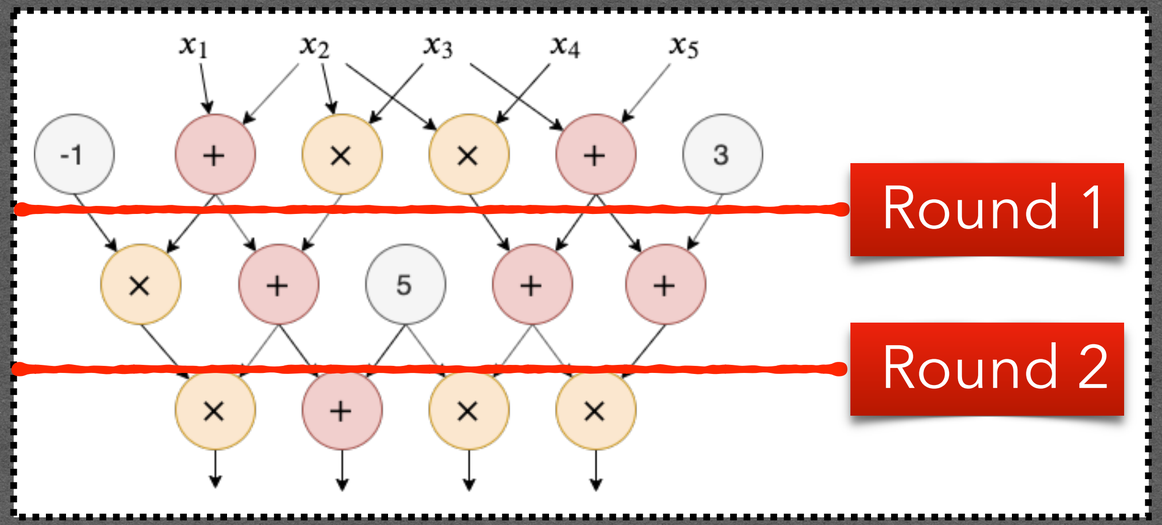
O(#gates) computation

Collaborative zkSNARKs

O(#gates) computation

+

O(mult. depth) communication



Collaborative zkSNARKs – Efficiency Limitations

1. Each Server's i computation is proportional to $\|w\|$ not $\|w_i\|$
2. Communication overhead: Multiplication depth is of essence

Traditional zkSNARKs

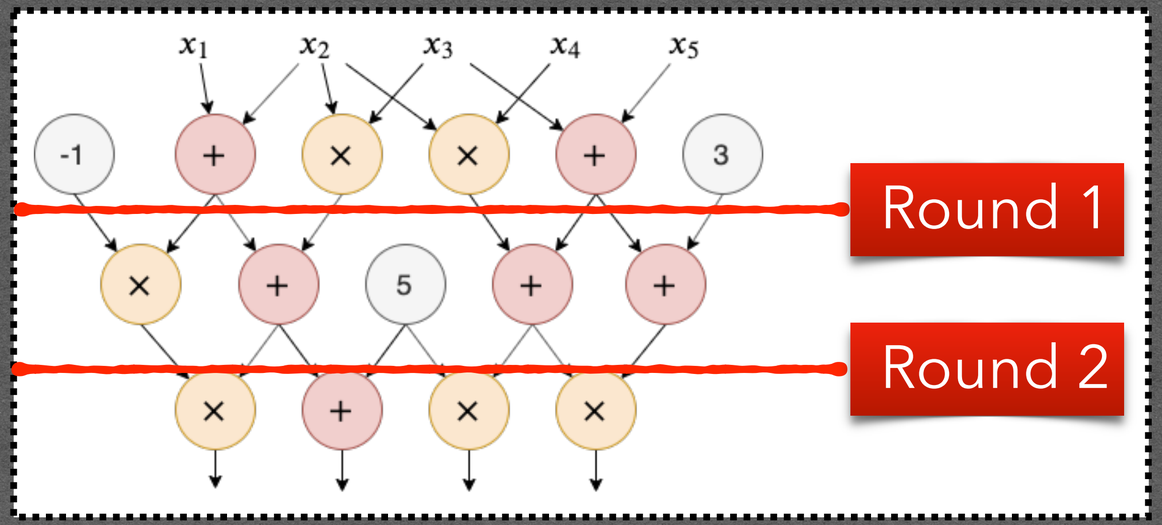
O(#gates) computation

Collaborative zkSNARKs

O(#gates) computation

+

O(mult. depth) communication



E.g. Poseidon Hash is not suitable for Collaborative SNARKs

Collaborative zkSNARKs – Efficiency Limitations

1. Each Server's i computation is proportional to $\|w\|$ not $\|w_i\|$
2. Communication overhead: Multiplication depth is of essence

Traditional zkSNARKs

O(#gates) computation

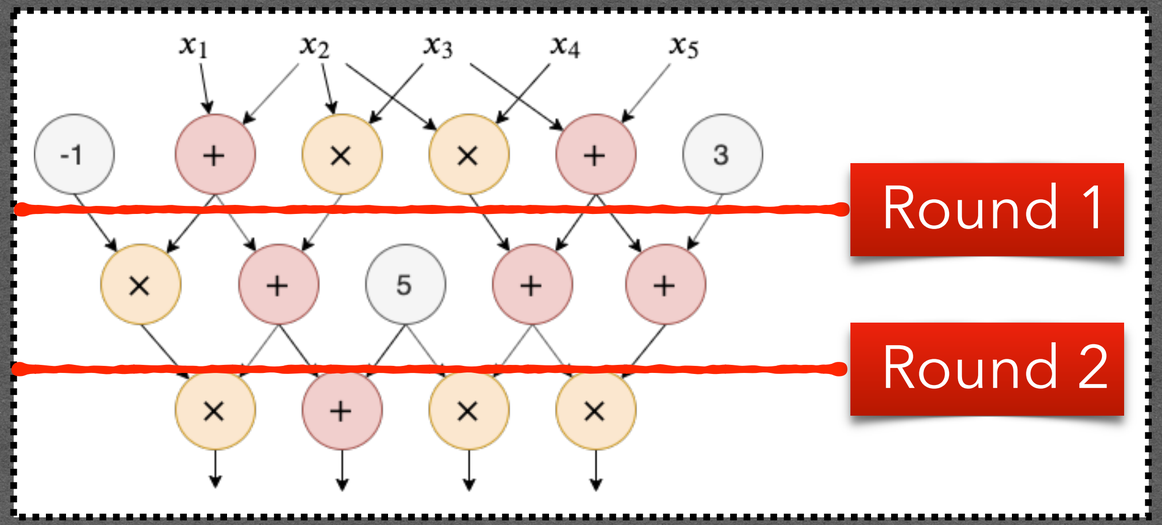
>2min for Merkle Tree opening

Collaborative zkSNARKs

O(#gates) computation

+

O(mult. depth) communication



E.g. Poseidon Hash is not suitable for Collaborative SNARKs

Jigsaw Core Technique (1)

π **zkSNARK for: (1)** sn_{spent} **valid (**$\mathsf{cm}_{spent} \in \mathsf{root}$**,...)**

(2) cm_{new} **well formed**

(3) $f(\mathsf{payload}_{old}, \mathsf{payload}_{new}) = 1$

Core Observation: The bulk of the work includes only local data

Jigsaw Core Technique (1)

π **zkSNARK for: (1)** sn_{spent} **valid (**$\mathsf{cm}_{spent} \in \mathsf{root}$**,...)**

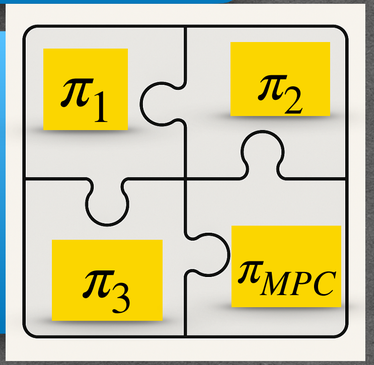
(2) cm_{new} **well formed**

(3) $f(\mathsf{payload}_{old}, \mathsf{payload}_{new}) = 1$

Core Observation: The bulk of the work includes only local data

Main idea:

- Each client computes a local zkSNARK for their data
- Servers compute a collaborative zkSNARK for what's left



Jigsaw Core Technique (1)

π **zkSNARK for: (1)** sn_{spent} **valid (**$\text{cm}_{spent} \in$ root**,...)**

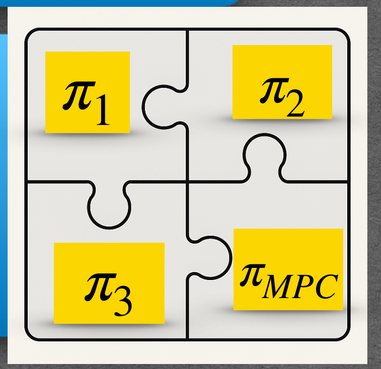
(2) cm_{new} **well formed**

(3) $f(\text{payload}_{old}, \text{payload}_{new}) = 1$

Core Observation: The bulk of the work includes only local data

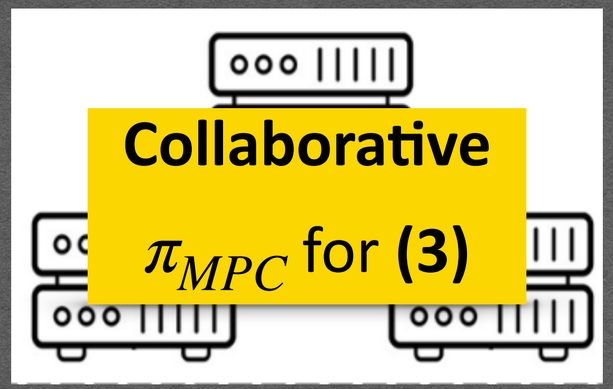
Main idea:

- Each client computes a local zkSNARK for their data
- Servers compute a collaborative zkSNARK for what's left



Local π_1 for **(1)+(2),** $[w_1]$

Local π_2 for **(1)+(2),** $[w_2]$



$(\pi_1, \pi_2, \pi_{MPC})$

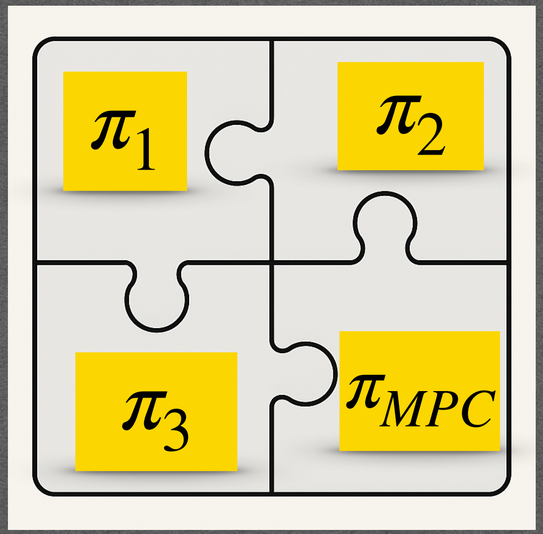
Jigsaw Core Technique (2)

Careful Decomposition of the relation:

Client i

Local zkSNARK π_i for:

- Merkle tree inclusions
- Commitments opening
- PRF computations



Servers

Collaborative zkSNARK π_i for:

- Execution of f
- A few field operations

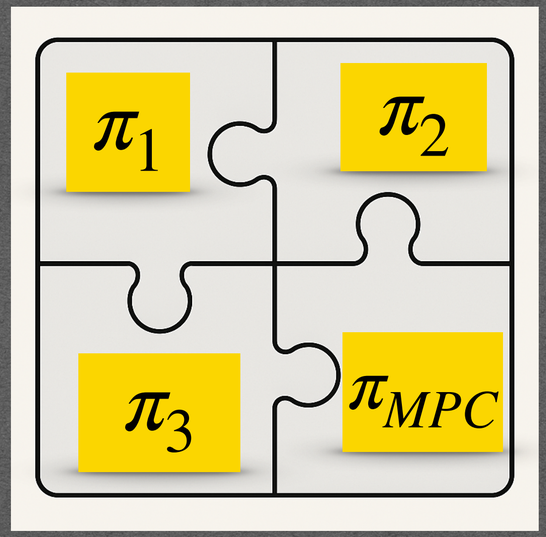
Jigsaw Core Technique (2)

Careful Decomposition of the relation:

Client i

Local zkSNARK π_i for:

- Merkle tree inclusions
- Commitments opening
- PRF computations



Servers

Collaborative zkSNARK π_i for:

- Execution of f
- A few field operations

Extremely simple for many applications (e.g. DEX, auction)

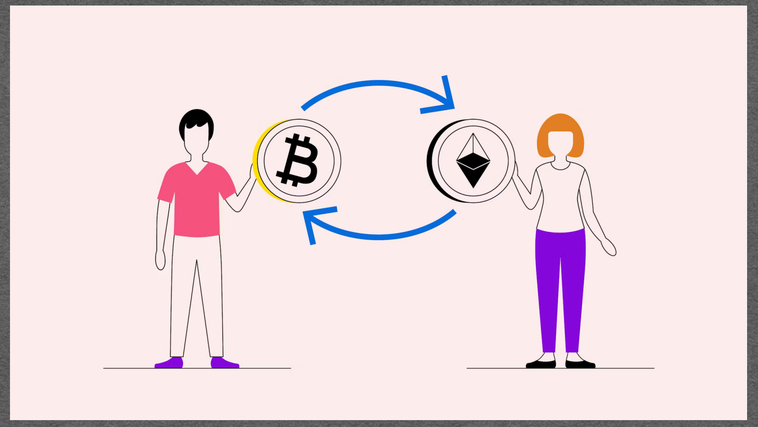
More Technical Subtleties

- **Commit-and-prove** zkSNARKs to ensure π_i, π_{MPC} are over the same data.
  ➡ Commit-and-prove PLONK variant.

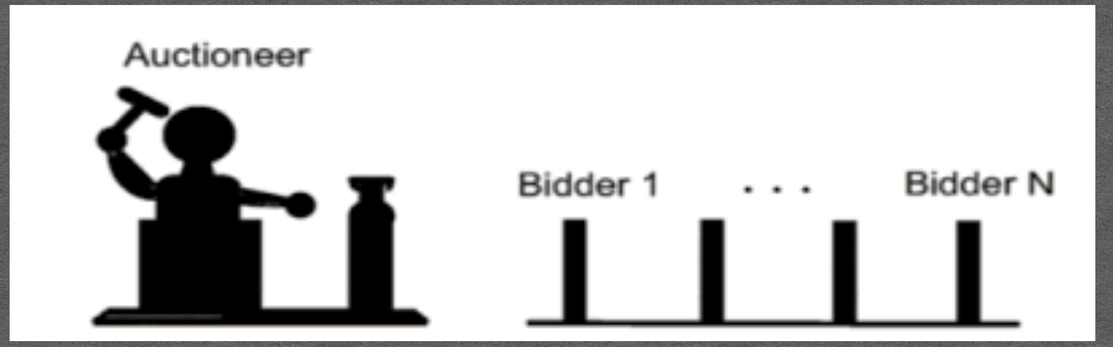
- **Signatures of Knowledge** to bind π_i with the intended f.

- **Proofs of correct secret-sharing** to prevent malicious clients.

Applications



Atomic Swaps/Trading



Sealed-bid Auctions

Lotteries

Voting

Implementation (1)

Local zkSNARK:

- (i) TurboPLONK [GW20] + (ii) Custom SNARK for CP-link + (iii) Custom SNARK of Correct Secret Sharing
- Macbook Pro with 8-core M2 CPU and 16 GB RAM
- Multicore implementation

Proving Time (sec): ~ 1.3 — 3.6

Implementation (2)

40-50x faster than generic Collaborative zkSNARKs

Collaborative zkSNARK:

- Taceo toolchain implementation
- 3 AWS c4.xlarge machines, with 4 vCPUs and 8 GB RAM each

Application	Parameter	Ext Witness Gen		Plonk Proof Gen	
		LAN	WAN	LAN	WAN
Atomic Swap	2 parties	0.26 s	1.08 s	1.14 s	1.84 s
Auction	50 bids	1.02 s	2.22 s	2.1 s	2.85 s
	100 bids	1.81 s	3.07 s	2.1 s	2.85 s
Lottery	100 entries	0.09 s	0.16 s	0.13 s	0.88 s
	1000 entries	0.1 s	0.17 s	0.25 s	1.03 s
Voting	10 voters	0.09 s	0.16 s	0.15 s	0.91 s
	100 voters	0.11 s	0.18 s	0.25 s	1.03 s
	1000 voters	0.17 s	0.26 s	1.38 s	2.21 s

Implementation (2)

40-50x faster than generic Collaborative zkSNARKs

Collaborative zkSNARK:

- Taceo toolchain implementation
- 3 AWS c4.xlarge machines, with 4 vCPUs and 8 GB RAM each

Application	Parameter	Ext Witness Gen		Plonk Proof Gen	
		LAN	WAN	LAN	WAN
Atomic Swap	2 parties	0.26 s	1.08 s	1.14 s	1.84 s
Auction	50 bids	1.02 s	2.22 s	2.1 s	2.85 s
	100 bids	1.81 s	3.07 s	2.1 s	2.85 s
Lottery	100 entries	0.09 s	0.16 s	0.13 s	0.88 s
	1000 entries	0.1 s	0.17 s	0.25 s	1.03 s
Voting	10 voters	0.09 s	0.16 s	0.15 s	0.91 s
	100 voters	0.11 s	0.18 s	0.25 s	1.03 s
	1000 voters	0.17 s	0.26 s	1.38 s	2.21 s

Verification:

Gas Cost (K): ~432 + 472*#clients

Conclusions

Conclusions and Future Work

Conclusions:

- ❖ More elaborate applications –> More elaborate privacy challenges.
- ❖ Off-Chain privacy essential.
- ❖ Collaborative zkSNARKs have the potential for real-world deployment.

Future Work:

- ❖ DPSC with Function Hiding.
- ❖ Fine-tune MPC properties (Guaranteed output delivery, …).
- ❖ Special purpose MPC.

Thank you!