TACEO

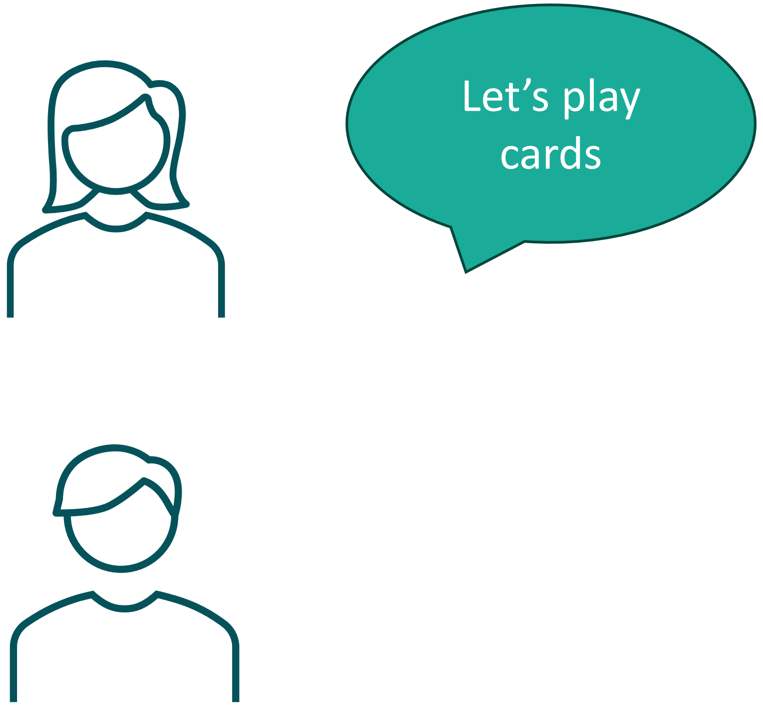
[Don‘t] share your data.

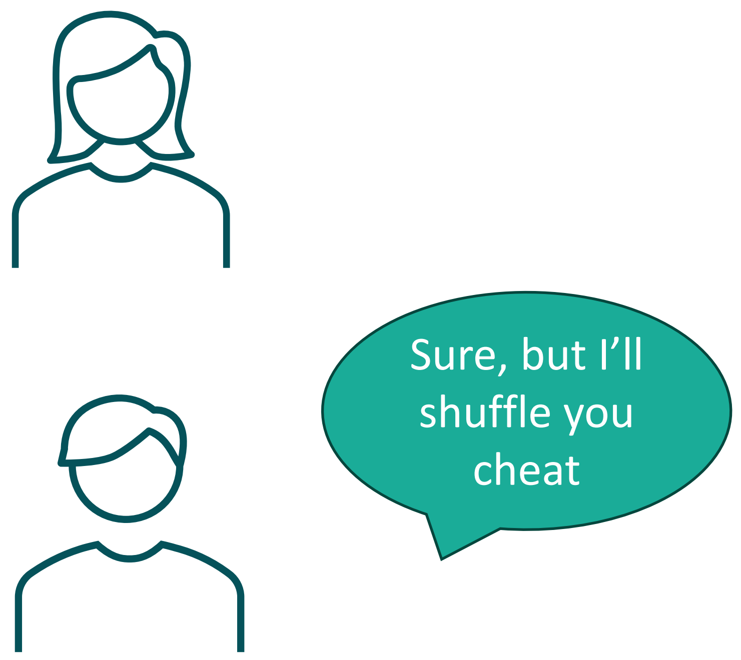
TACEO.IO

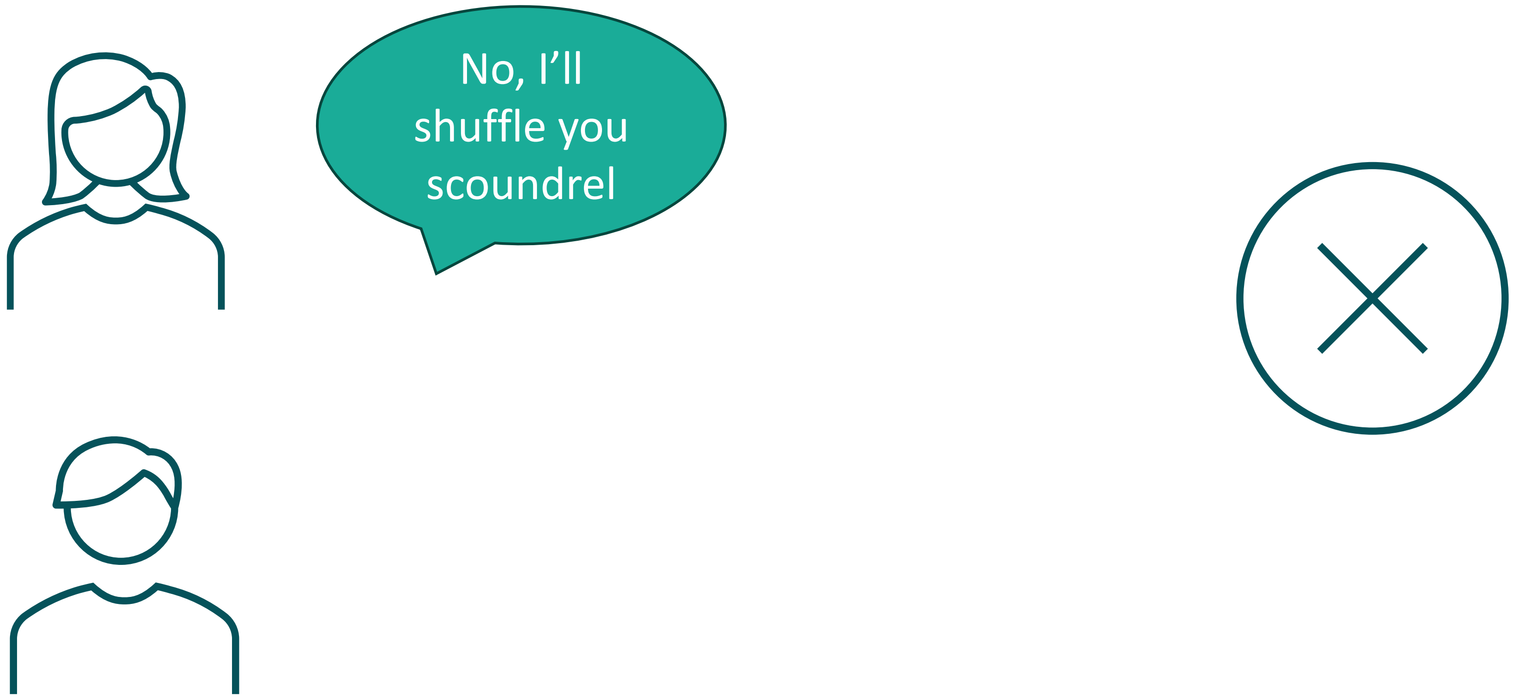
github.com/TaceoLabs/noir_workshop_0625/

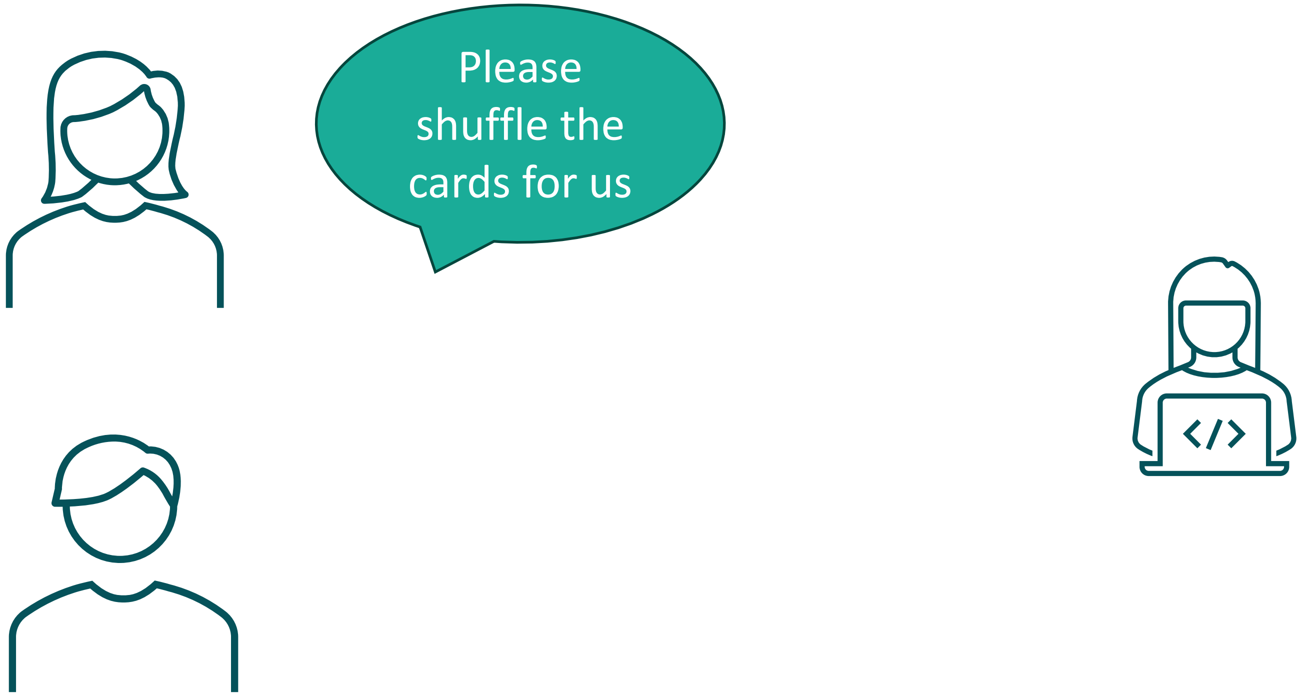
Building a coSNARK-powered DApp with coNoir

Shuffling Cards with MPC

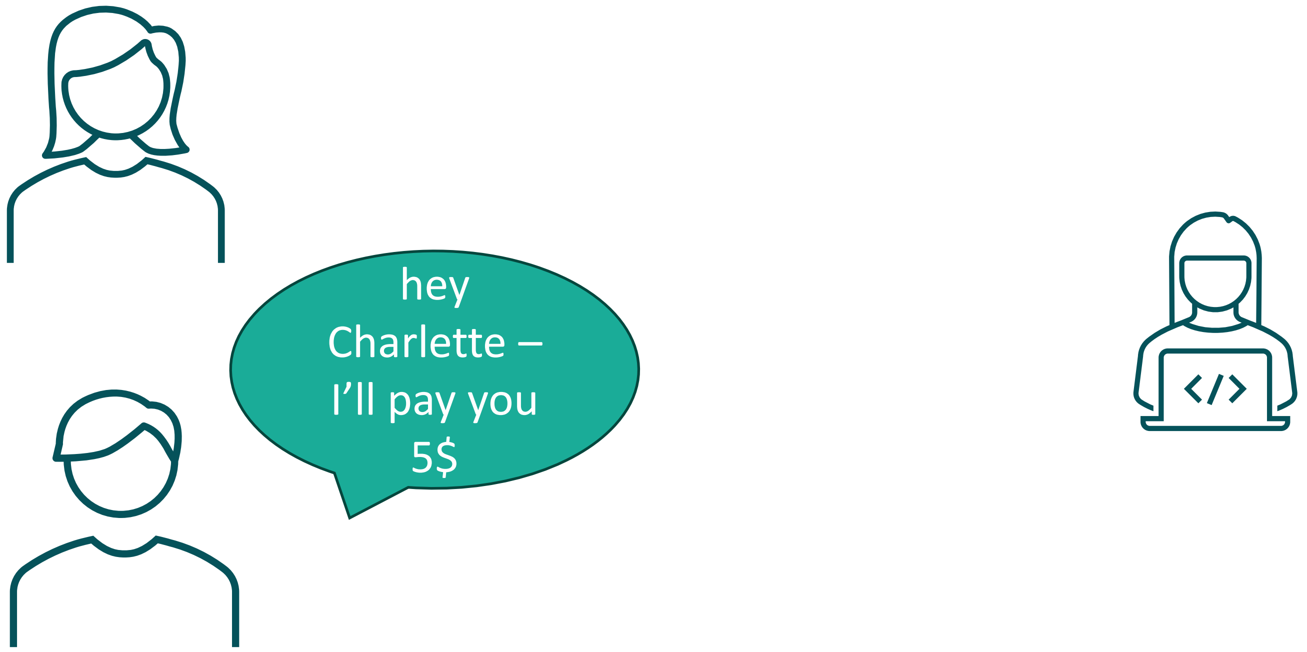

Let’s play cards


Sure, but I’ll shuffle you cheat


No, I'll shuffle you scoundrel


Please shuffle the cards for us

TACEO

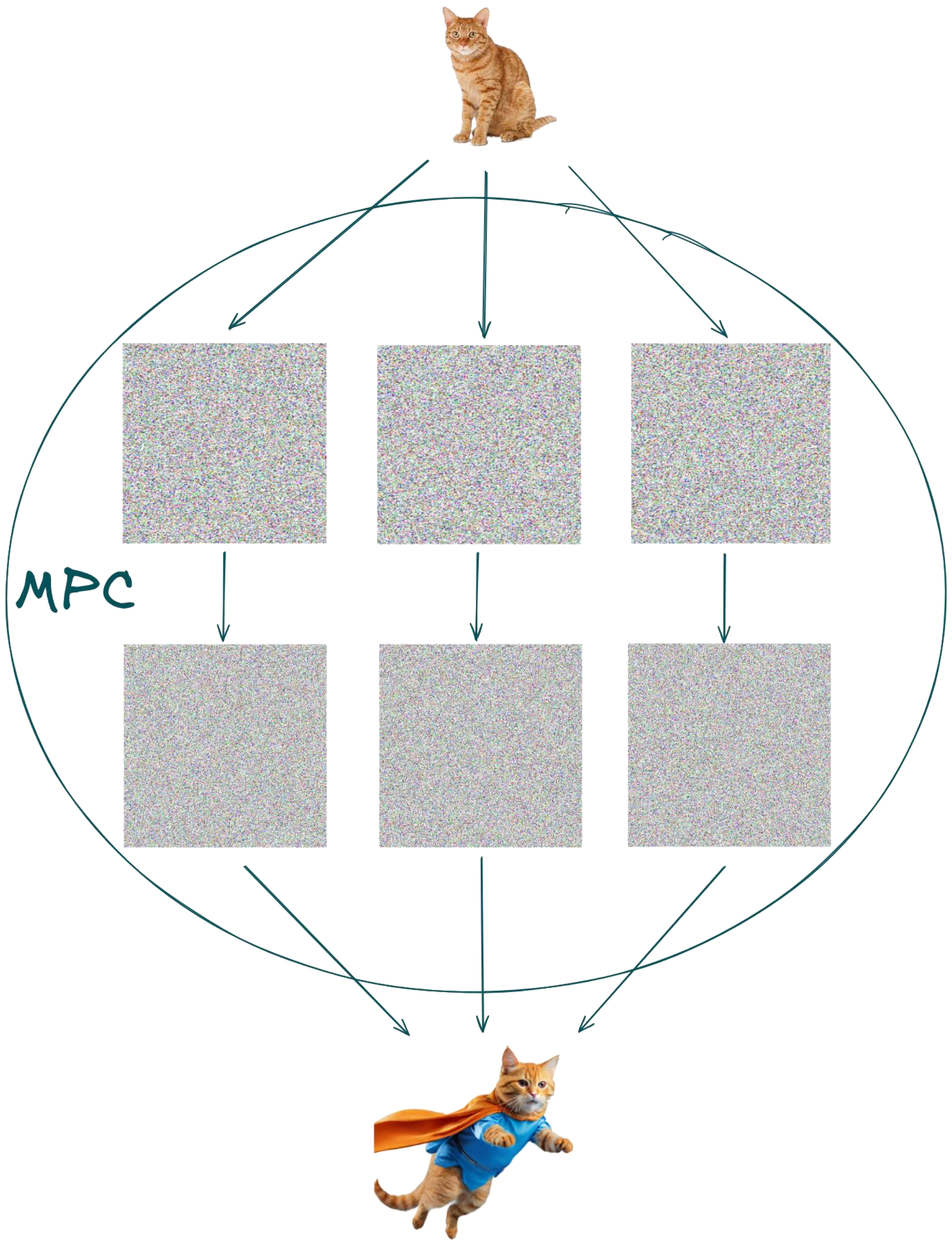

hey Charlette – I'll pay you 5$

TACEO

TACEO

What now?

Private State only gets you that far

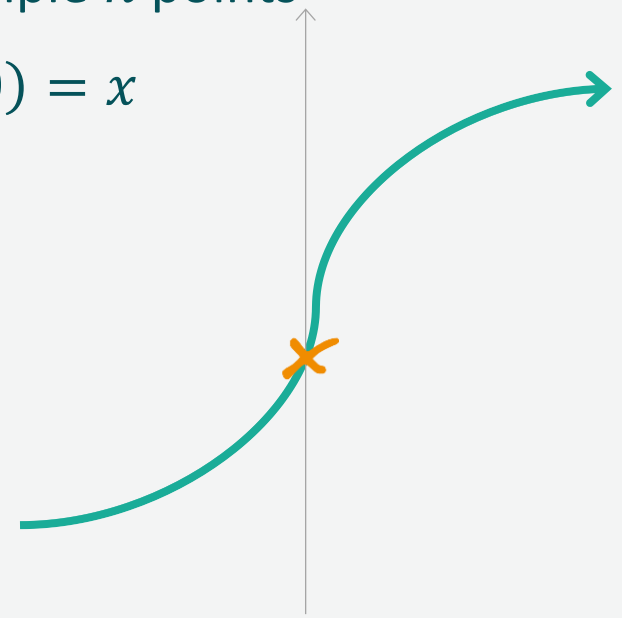

MPC

TACEO

How to share a secret?

Shamir Secret Sharing

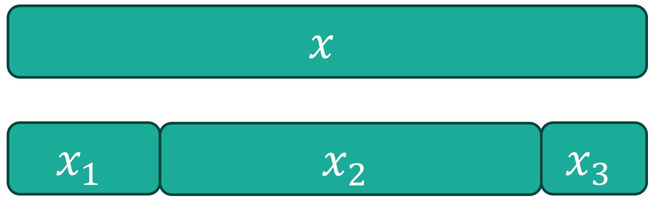
- Polynomial $f(X)$ of degree d
- Sample n points
- $f(0) = x$



Additive Secret Sharing

- Split x in n parts

$$\sum_{n} x_i = x$$

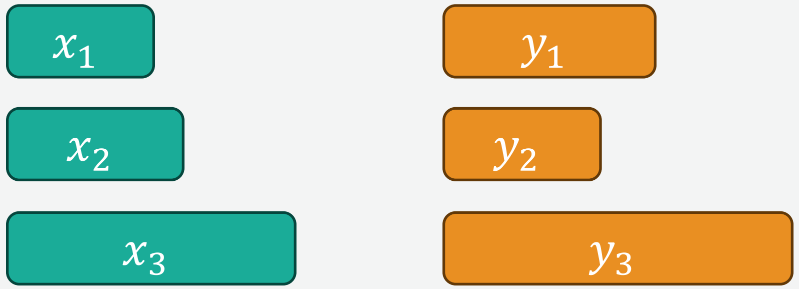


TACEO

Computing on secrets

Addition

Two secrets x and y

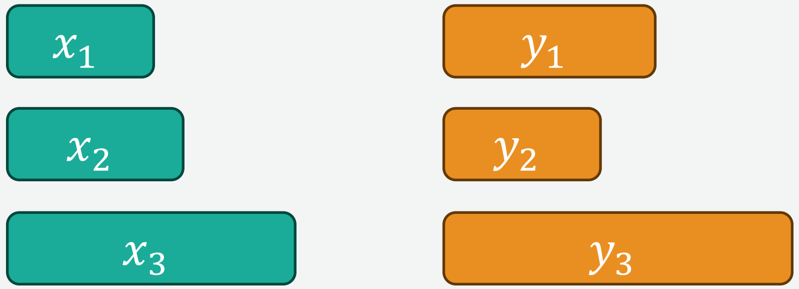


TACEO

Computing on secrets

Addition

Two secrets x and y

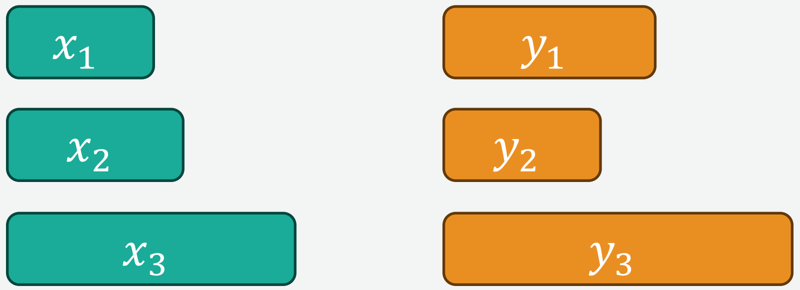


$$\begin{matrix} x_1 + y_1 \\ x_2 + y_2 \\ x_3 + y_3 \end{matrix} \quad = x + y$$

Computing on secrets

Addition

Two secrets x and y



$$\begin{matrix} x_1 + y_1 \\ x_2 + y_2 \\ x_3 + y_3 \end{matrix} \quad = x + y$$

Multiplication

?

Non-linear Operations

- Beaver triples
  - Generate helper triples $([\mathrm{a}], [\mathrm{b}], [c])$ and $ab = c$
  - Open $[a + x] = A$ and $[b + y] = B$
  - Compute $A[y] - B\,[a] + [c] = [xy]$

Non-linear Operations

- Beaver triples
  - Generate helper triples $([\mathrm{a}], [\mathrm{b}], [c])$ and $ab = c$
  - Open $[a + x] = A$ and $[b + y] = B$
  - Compute $A[y] - B\ [a] + [c] = [xy]$
- Replicated Secret Sharing (2 out of 3 sharing)
  - Parties hold two shares instead of one
  - Every party computes $x_a y_a + x_a y_b + x_b y_a + m$, where $m = [0]$
  - Reshare result

Great – what now?

zkSNARKs 101

Succinct proof for computational integrity

Keeping secret input hidden (potentially)

MPC 101

Mutually untrusting parties jointly compute a function

Inputs and intermediate values private (secret-shared)

TACEO

What are coSNARKs?

Zero-Knowledge

MPC

TACEO

What are coSNARKs?

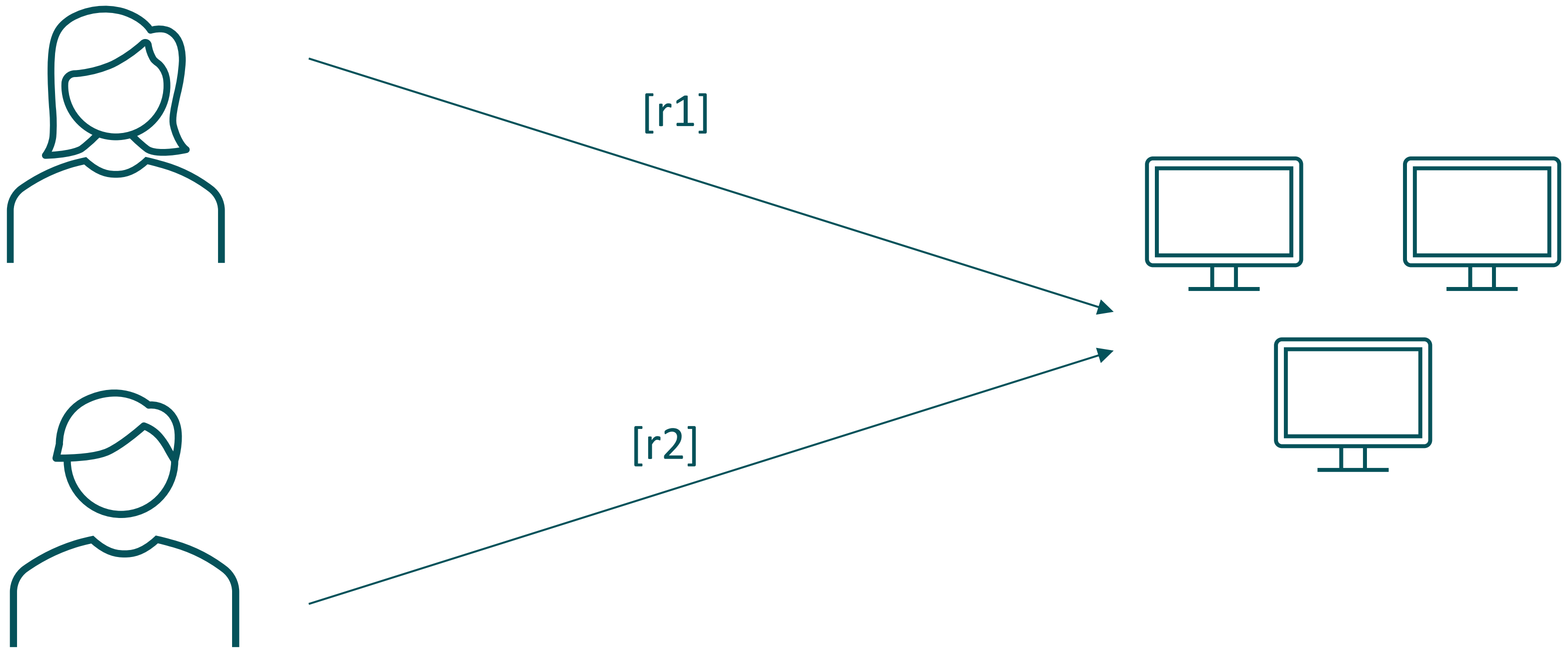
Zero-Knowledge + MPC = co-SNARKs

Co-SNARK 101

Mutually untrusting parties jointly compute a function

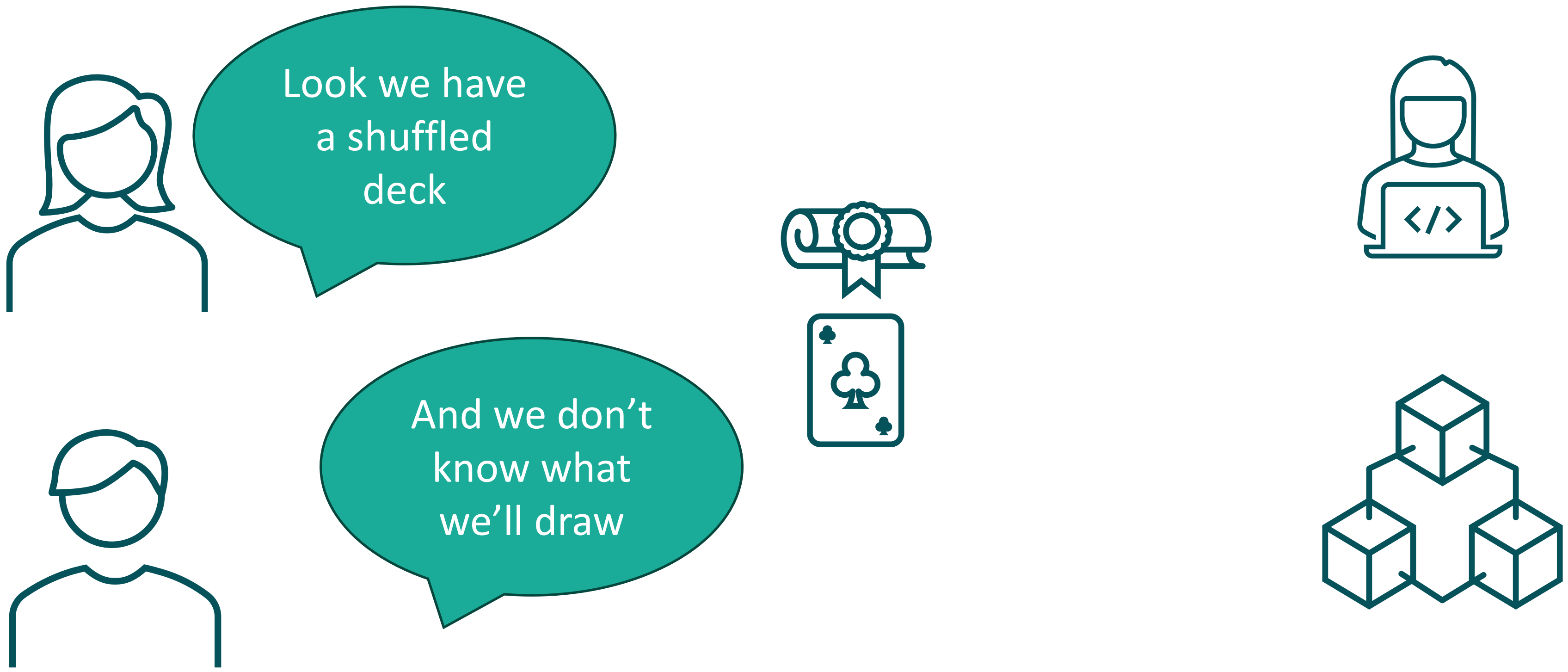
Succinct proof for computational integrity

Keeping secret inputs and intermediate values hidden


[r1]
[r2]

Here is a shuffled deck and a proof

Look we have a shuffled deck
And we don't know what we'll draw


Look we have a shuffled deck
And we don't know what we'll draw

We are done, right?

Let’s think this through

Let’s think this through

Playing a game of cards

- Shuffled the deck

Let’s think this through

Playing a game of cards

- Shuffled the deck
- Only distribute cards to respective players (verifiable encryption)

TACEO

Let’s think this through

Playing a game of cards

- Shuffled the deck
- Only distribute cards to respective players (verifiable encryption)

→ No one knows the structure of deck!

Let’s think this through

Playing a game of cards

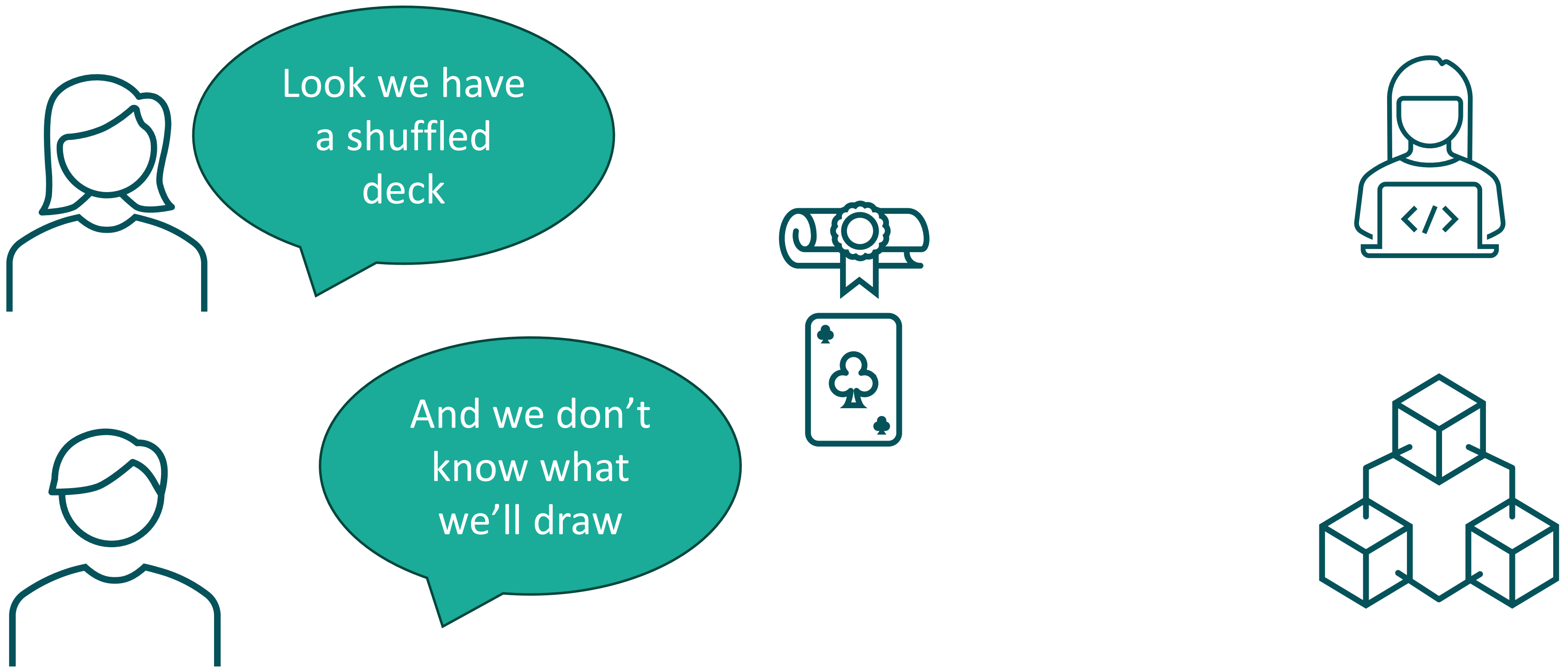
- Shuffled the deck
- Only distribute cards to respective players (verifiable encryption)

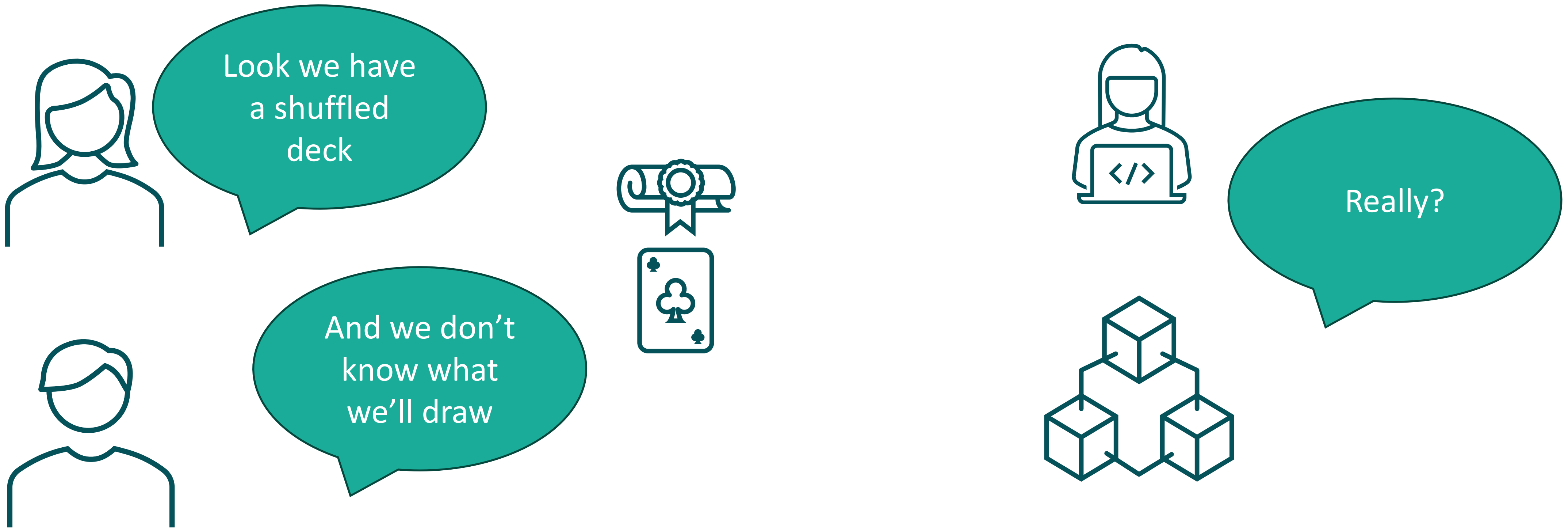
→ No one knows the structure of deck!

Private Shared State

- Not simply composing Alice and Bobs private state
- We generate new private state that nobody knows
- But: No input verification so far!

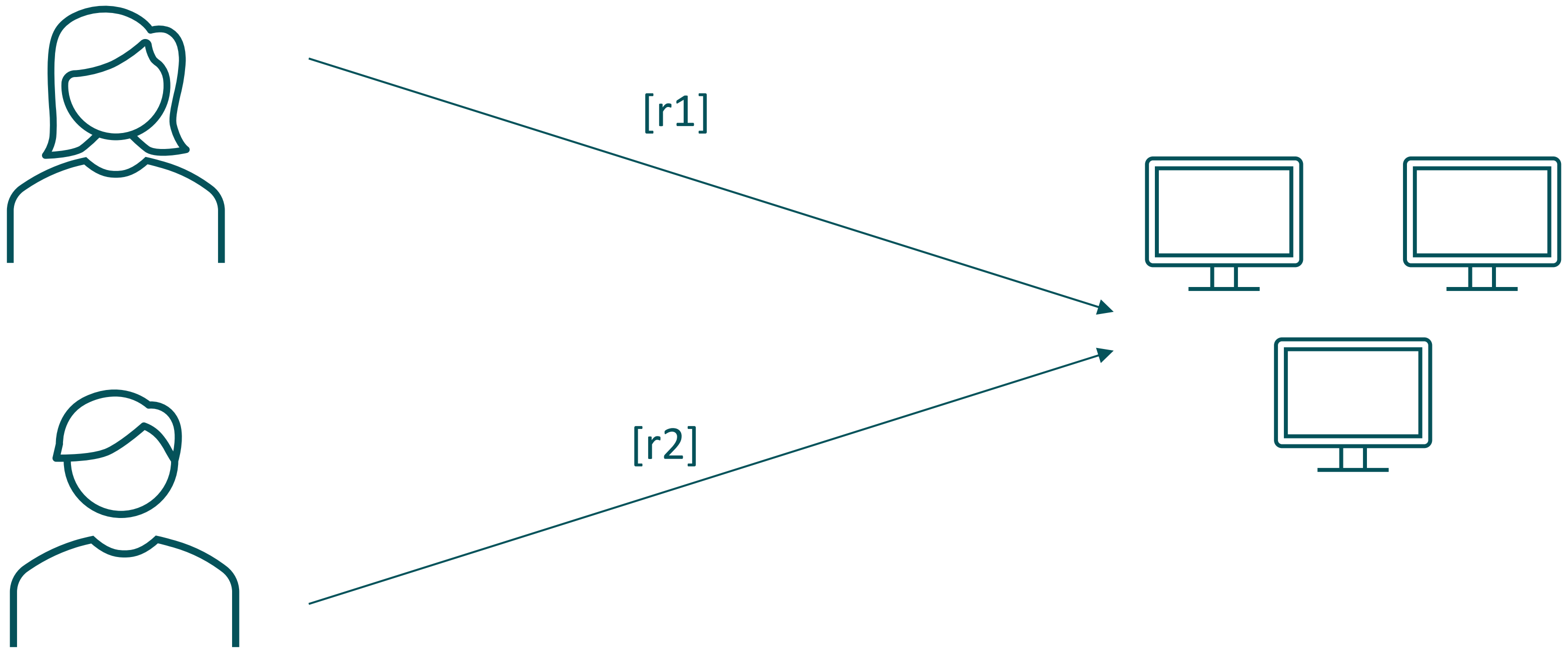
TACEO


Look we have a shuffled deck
And we don't know what we'll draw

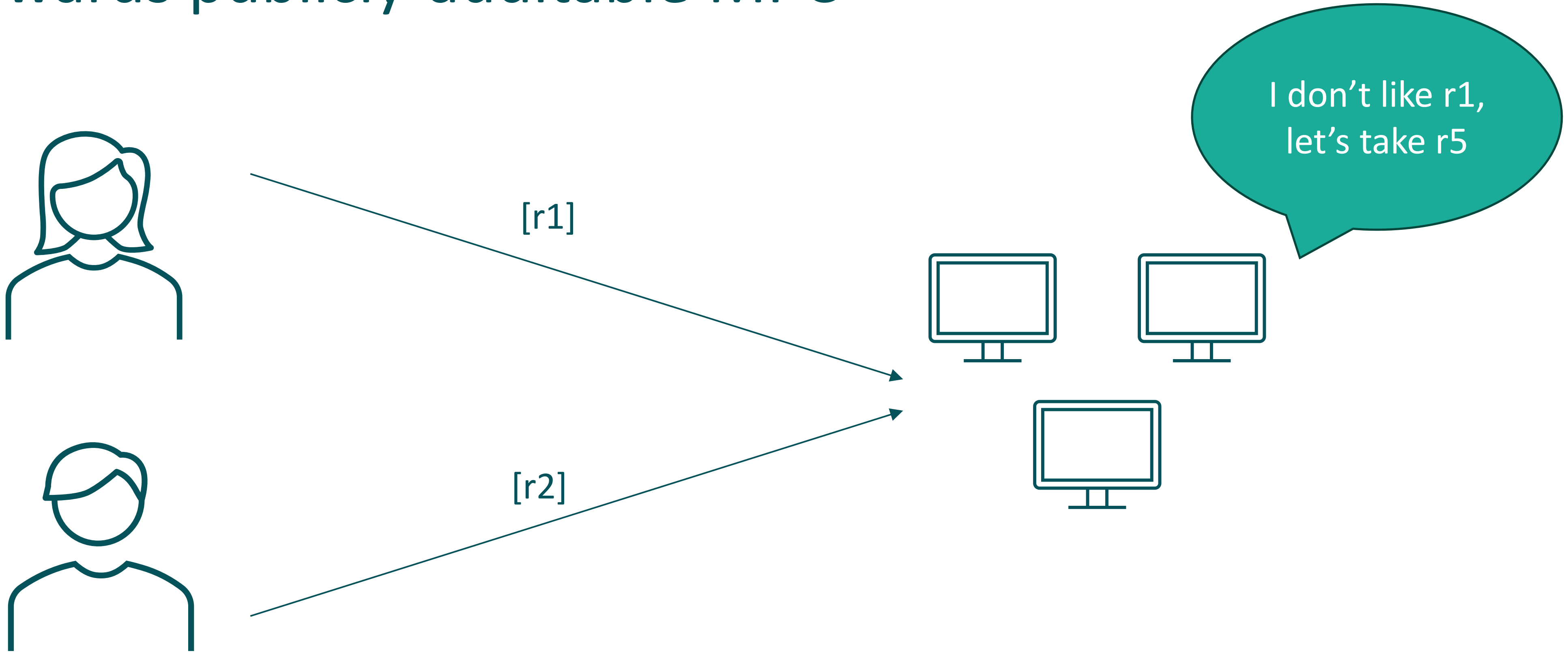

Look we have a shuffled deck
And we don't know what we'll draw
Really?

TACEO

Towards publicly-auditable MPC



Towards publicly-auditable MPC



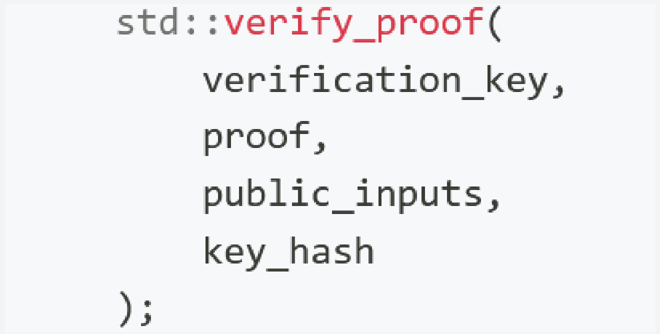
=> Solution: Always bind data to public commitments

Towards publicly-auditable MPC

Interop with Aztec

UltraHonk Proof System

- Currently supported by coNoir
- Recurse into smart contract



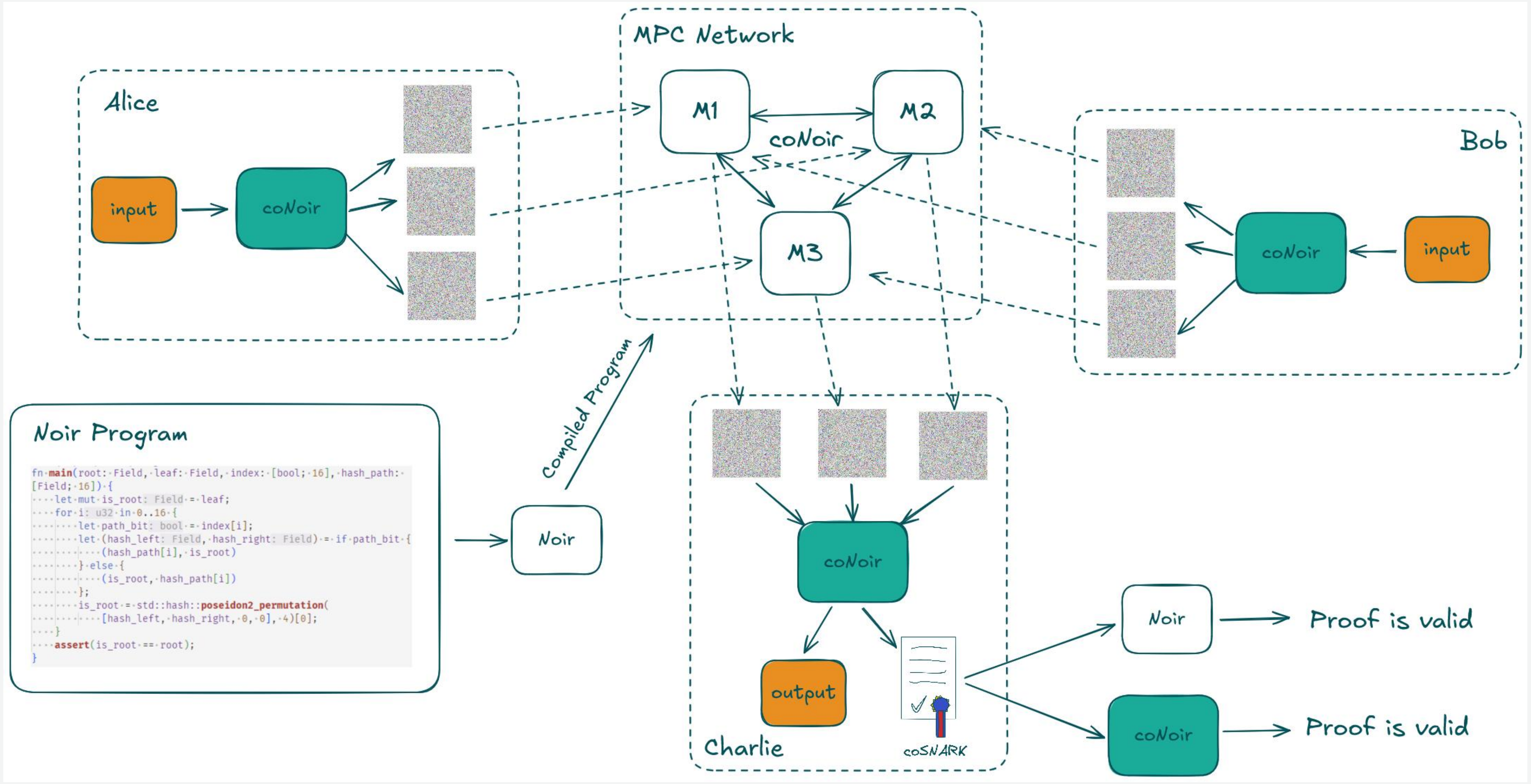
```
    std::verify_proof(
        verification_key,
        proof,
        public_inputs,
        key_hash
    );
```

Full Client-IVC Proof System

- Prove whole Aztec transactions in MPC
- Create Aztec keys that are secret-shared and own Private State

github.com/TaceoLabs/noir_workshop_0625/

TACEO



```
fn main(root: Field, leaf: Field, index: [bool; 16], hash_path:
[Field; 16]) {
    let mut is_root: Field = leaf;
    for i: u32 in 0..16 {
        let path_bit: bool = index[i];
        let (hash_left: Field, hash_right: Field) = if path_bit {
            (hash_path[i], is_root)
        } else {
            (is_root, hash_path[i])
        };
        is_root = std::hash::poseidon2_permutation(
            [hash_left, hash_right, 0, 0], 4)[0];
    }
    assert(is_root == root);
}
```

FOLLOW US!

GitHub	Twitter / X	LinkedIn	TACEO GmbH

Am Eisernen Tor 5 8010

Graz | Austria

office@taceo.io