CSF'21: Formal security analysis of MPC-in-the-head zero-knowledge protocols

Nikolaj Sidorenco[1] Sabine Oechsner[3] Bas Spitters[12]

{sidorenco, spitters}@cs.au.dk, s.oechsner@ed.ac.uk

[1]Department of Computer Science
Aarhus University, Denmark

[2]Concordium Blockchain Research Center
Aarhus University

[3]Laboratory for Foundations of Computer Science
University of Edinburgh

https://eprint.iacr.org/2021/437

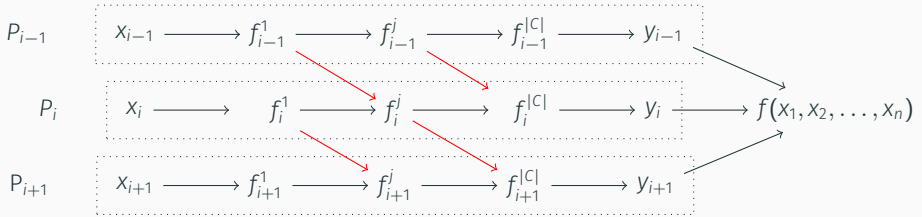
Zero-Knowledge Protocols

- Two parties: A Prover and a Verifier.
- Prover wants to convince the Verifier that he knows a secret x.
- The Verifier should not learn anything about the secret.
- The secret is part of a relation:
  - $\mathsf{R} = \{(x, y) : f(x) = y\}$

We require that the Verifier always accepts if the Prover knows the secret and follows the protocol.

Multi-Party Computation

- N parties want to jointly and securely compute a N-ary function $f(x_1, \ldots, x_N)$.
  - Each party supplies a private input x_i.
  - Everyone learns $f(x_1, \ldots, x_n)$. Parties should not learn any private inputs.
- f is a circuit. Let f_i^j denote the function party i uses to compute gate j.
- Needs to be correct and private.
  - Correctness: $\text{MPC}(f, x_1, \ldots, x_N) = f(x_1, \ldots, x_N)$
  - Privacy: Parties should not learn each others inputs.
- We only consider semi-honest security.



What is MPC-in-the-head

- A technique to construct zero-knowledge proofs for any NP-relation [IKOS07]
- Uses MPC and commitment schemes as the underlying primitives.
  - Key insight is that the Prover can emulate all parties.
- Initially mostly of theoretical interest.
- Recent efficiency gains in implementations.
  - Can in some scenarios offer ZK proofs competitive with SNARKS.
  - Post-quantum secure signature scheme [CDGORR17].
  - ZKBoo [GMO16], Picnic [CDGORR17], Banquet [BGKOSZ21], BBQ [GMOS19], and others.

Zero-Knowledge from MPC

Prover

$\mathsf{R} = \{(x, y) : f(x) = y\}$

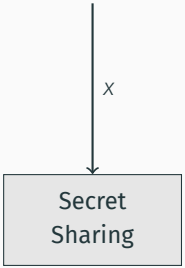
Verifier

Zero-Knowledge from MPC

Prover

$\mathsf{R} = \{(x, y) : f(x) = y\}$

Verifier

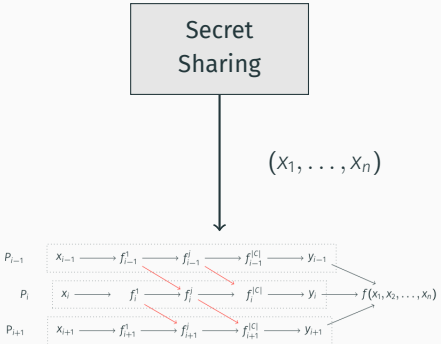


Zero-Knowledge from MPC

Prover

$\mathsf{R} = \{(x, y) : f(x) = y\}$

Verifier



Zero-Knowledge from MPC

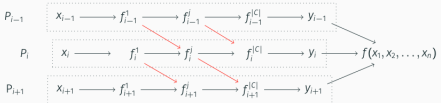
Prover

$\mathsf{R} = \{(x, y) : f(x) = y\}$

Verifier

Commit to views $V_1, \ldots, V_N$

$(V_1, \ldots, V_N)$



Zero-Knowledge from MPC

Prover

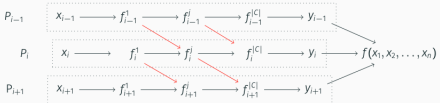
$\mathsf{R} = \{(x, y) : f(x) = y\}$

Verifier

Commit to views $V_1, \ldots, V_N$

Send challenge

$(V_1, \ldots, V_N)$



Zero-Knowledge from MPC

Prover

$\mathsf{R} = \{(x, y) : f(x) = y\}$

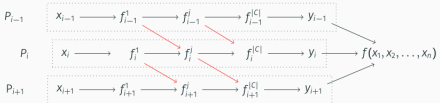
Verifier

Commit to views $V_1, \ldots, V_N$

Send challenge

Send subset of views

$(V_1, \ldots, V_N)$



Zero-Knowledge from MPC

Prover

$\mathsf{R} = \{(x, y) : f(x) = y\}$

Verifier

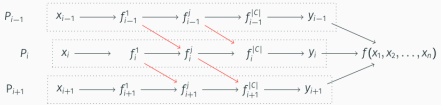
Commit to views $V_1, \ldots, V_N$

$(V_1, \ldots, V_N)$

Send challenge

Send subset of views

validate commitment and verify views



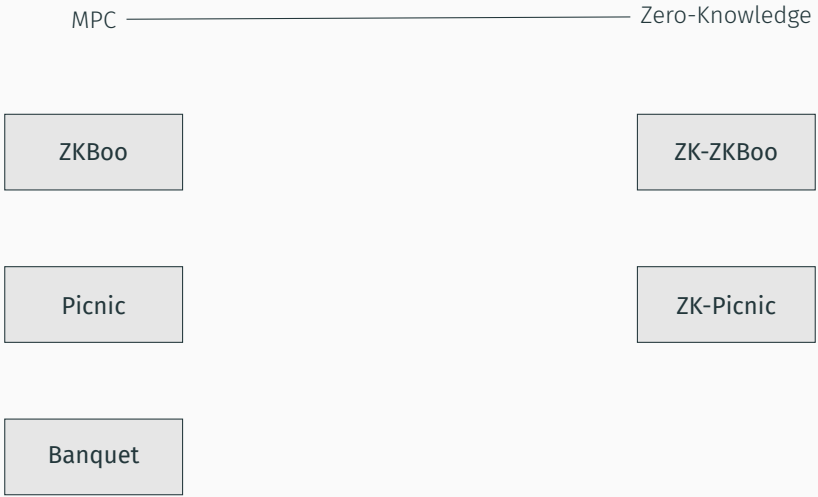
Motivation

- This flavour of the MPC-in-the-head approach consists of two components:
  1. A MPC protocol
     - with a method for validating the computations of a subset of the parties in the MPC protocol.
  2. A general purpose transformation protocol mapping MPC to zero-knowledge

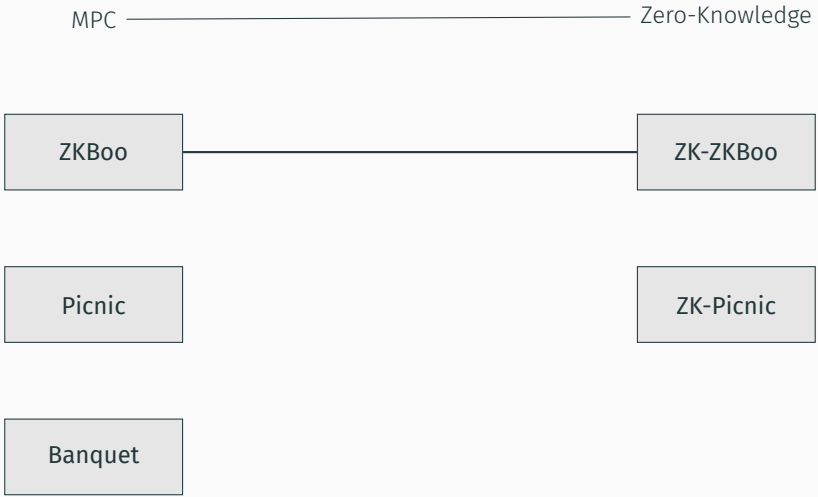
Recent works originating from this approach [GMO16, CDGORR17, KKW18] prove security of the construction as a whole.

Can our security proofs reflect this inherent modularity?

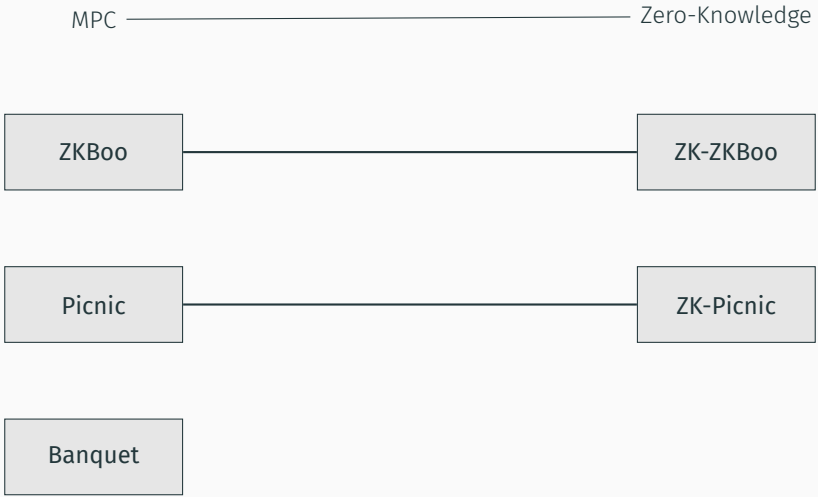
Why do we need modular security proofs?



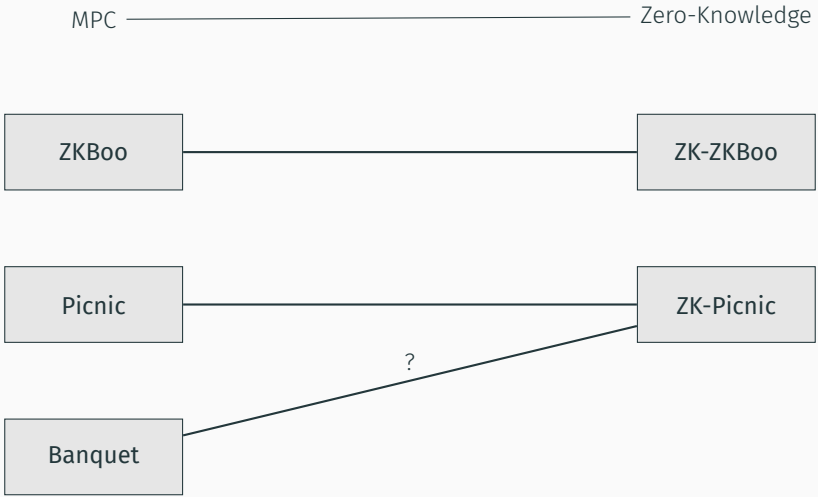
Why do we need modular security proofs?



Why do we need modular security proofs?



Why do we need modular security proofs?



Modular security proofs

- There are some subtle challenges
  - Does verification of the views imply a fixed input?
  - Can correctness or privacy ensure that the Verifier will accept the views?

Zero-Knowledge from MPC

$\mathsf{R} = \{(x, y) : f(x) = y\}$

Prover **Verifier**

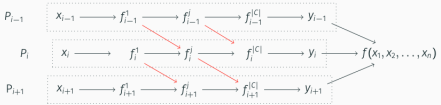
Commit to views $V_1, \ldots, V_N$

$(V_1, \ldots, V_N)$

Send challenge

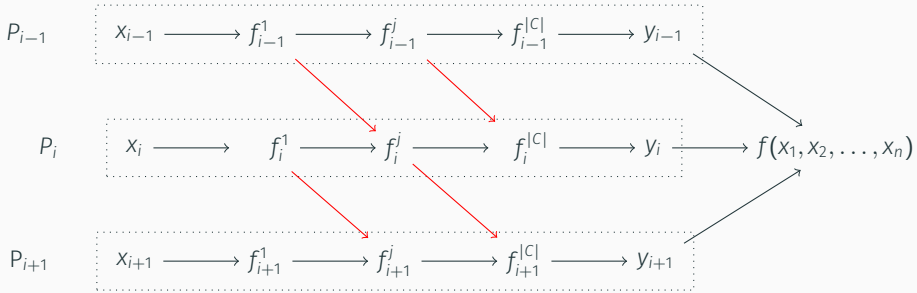
Send subset of views

validate commitment and verify views



Communication Pattern (of ZKBoo)

Verification needs to reason about intermediate computations.



Modular security proofs

- There are some subtle challenges
  - Does verification of the views imply a fixed input?
  - Can correctness or privacy ensure that the Verifier will accept the views?
- We overcome these challenges by extending the definition of MPC protocols.
- Our class of MPC protocols are a collection of procedures.
  - With extended security properties based on the procedures.

New Security definitions

Definition (Verifiability)

A MPC protocol is verifiable if the given validation procedure always outputs true for any (valid) subset of views produced by the MPC protocol.

Verifiability ensure the Verifier always accept if the Prover knows the secret and follows the protocol.

Formalisation

These modular security proofs facilitated our formalization in EasyCrypt.

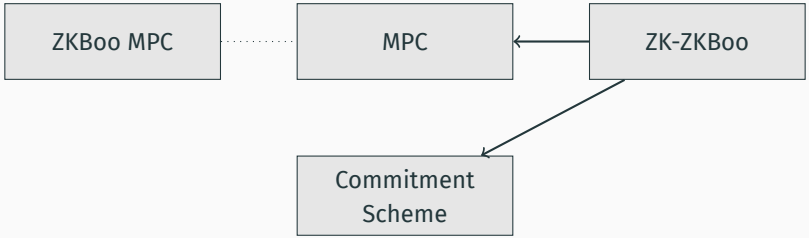
- We define our extended MPC notion in the proof assistant.
- we formalize the ZKBoo MPC protocol and its zero-knowledge transformation.
- Until now MPC and ZK were only formalized separately.
- First time a MPC formalisation has been used as a building block.

With formal verification we can precisely state our security requirement on protocol implementations. Moreover, we can prove security results in our chosen model of computation.

Formalisation Structure

A MPC protocol is a generic interface exposing functions required by the MPC-in-the-head transformation.

The Zero-Knowledge instantiates the MPC protocol with the required types.



Conclusion

- Modularity is needed to manage complexity.
  - Supports proof effort.
  - Allows us to freely compose decomposition protocols with zero-knowledge transformations.
  - Side effect: defines a common interface between decomposition and zero-knowledge.
- Formal verification:
  - EasyCrypt Formalization of modular framework for MPC-ith and ZKBoo as a concrete instance.