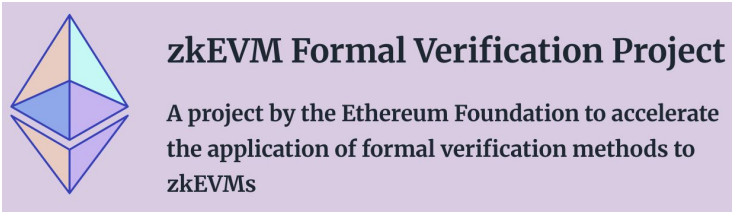
zkLib - Verified Proof Systems in Lean

Quang Dao (CMU)

Joint work with Devon Tuma (UMinnesota) & Gregor Mitscha-Baude (zkSecurity)

ZKProof 7, Sofia

Formal Verification for SNARKs



This work!

- Interactive (Oracle) Proofs-based SNARKs
- Linear-PCP-based SNARKs (e.g. Groth16)
- Verified verifier's implementation for Groth16 / Plonk
- DSLs for formally verified circuits
- (Semi-)automated solvers for under constrained bugs

Backend — Implementation — Frontend

zkLib: Formally Verifying SNARK Backends in Lean

Goals:

- Mechanize the security proofs of current SNARKs (IOP-based)
- Extracting verified implementations from verified SNARK specifications

Desiderata:

- SNARKs should be specified & proven secure in a ***modular*** and ***compositional*** manner
- Develop a ***core language*** & ***program logic*** for Interactive Oracle Reductions

  $\Rightarrow$ aimed to clarify existing constructions & streamline security proofs

Talk Outline

1. Anatomy of IOP-based SNARKs

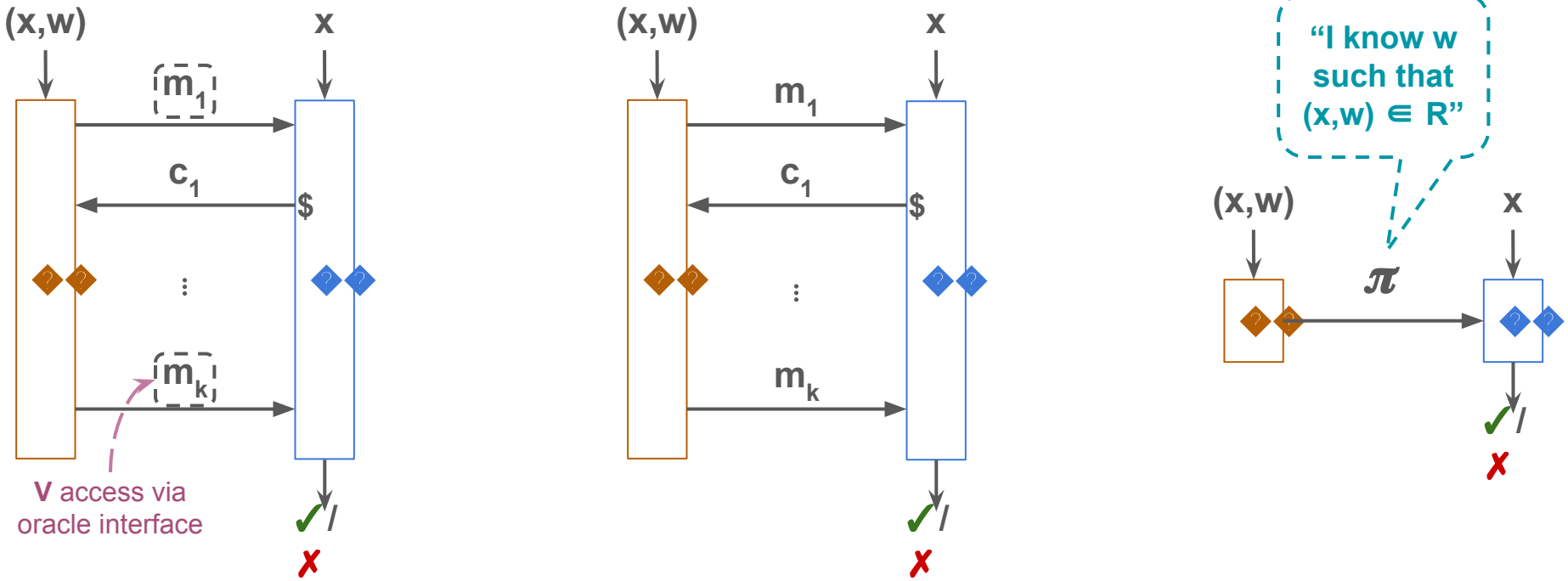
2. (WIP) Program logic for IORs

3. zkLib’s current development, and next steps

Talk Outline

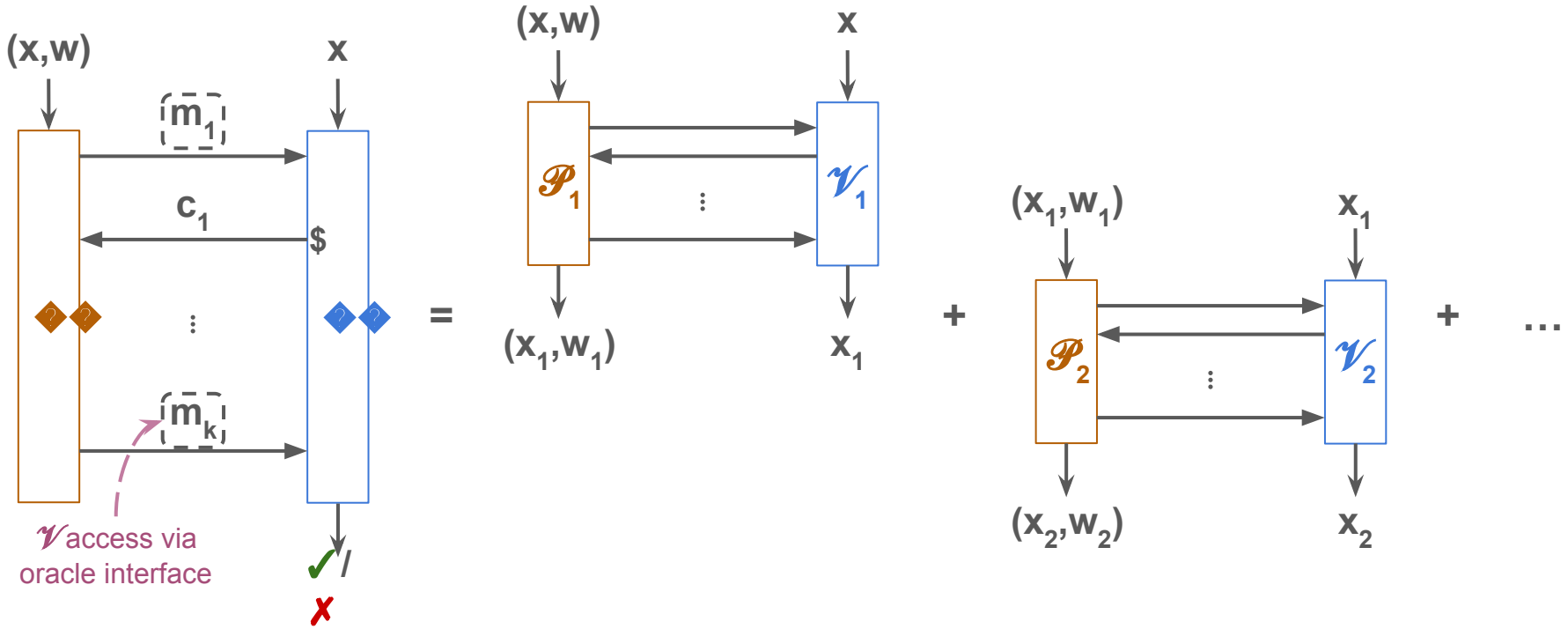
1. Anatomy of IOP-based SNARKs
2. (WIP) Program logic for IORs
3. zkLib’s current development, and next steps

What are IOP-based SNARKs?

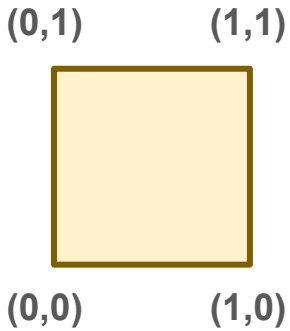


How are IOPs constructed?

Via composing a series of **interactive oracle reductions (IORs)**



IOR Example: The Sum-Check Protocol

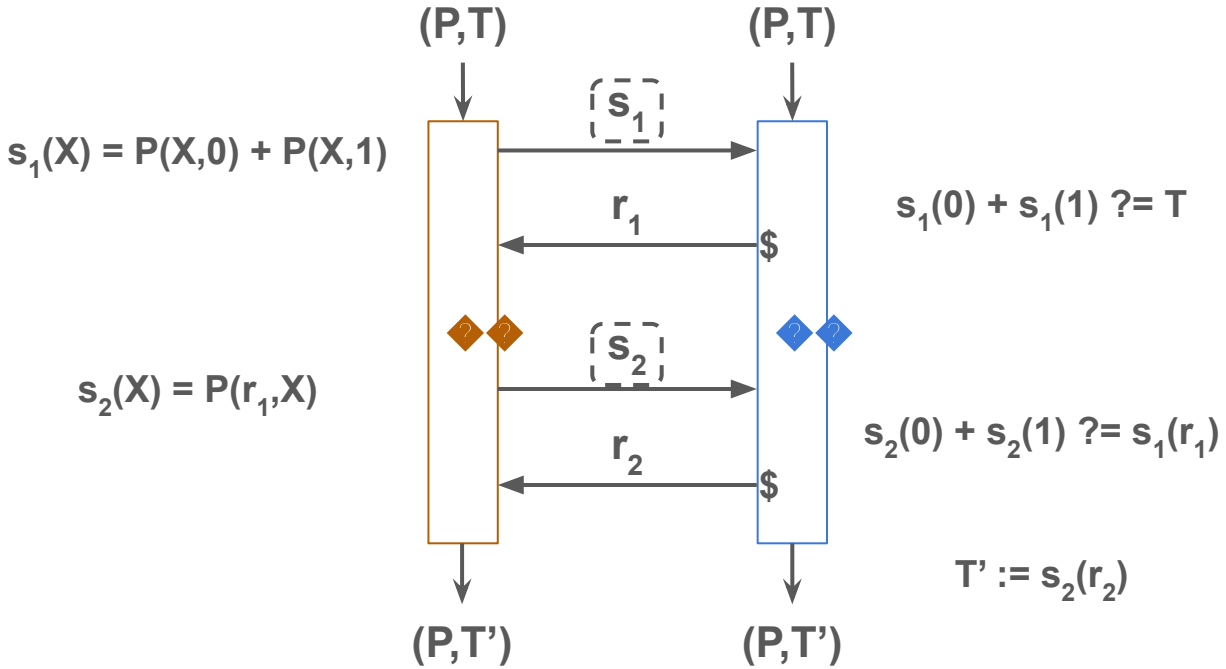


Relation R_{in} :

$P(0,0) + P(1,0) + P(0,1) + P(1,1) = T$

Relation R_{out} :

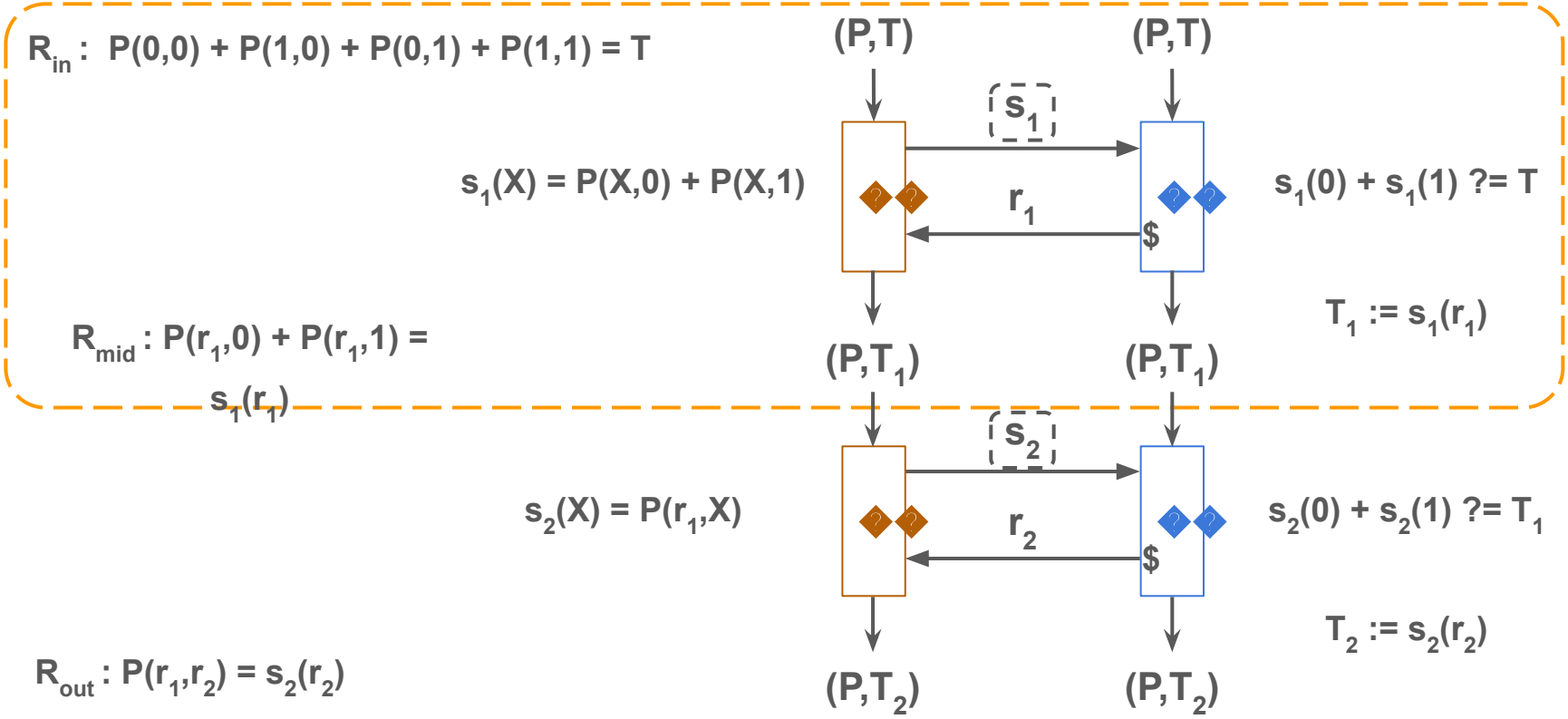
$P(r_1,r_2) = s_2(r_2)$



Talk Outline

1. Anatomy of IOP-based SNARKs
2. (WIP) Program logic for IORs
3. zkLib's current development, and next steps

The Sum-Check Protocol, Revisited



The Sum-Check Protocol, Revisited

Pr[fail] :=

$V(\cdot) = \checkmark \wedge R_{out}(\cdot,\cdot) =$ ✗

R_{in} : $P(0,0) + P(1,0) + P(0,1) + P(1,1) = T$

Pr[fail] = 0

$s_1(X) = P(X,0) + P(X,1)$

$s_1(0) + s_1(1) \mathrel{?}= T$

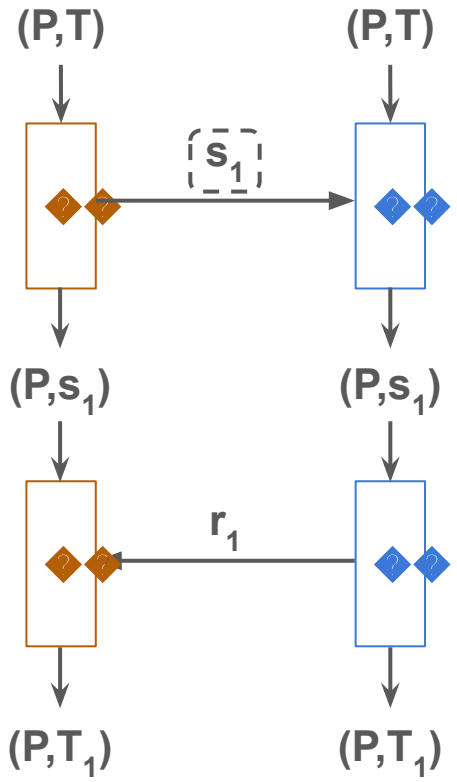
R_{mid}' : $P(X,0) + P(X,1) = s_1(X)$

Pr[fail] ≤ d / |F|

Round-by-round guarantees come for free!

R_{mid} : $P(r_1,0) + P(r_1,1) = s_1(r_1)$

$T_1 := s_1(r_1)$



The Sum-Check Protocol, Revisited

R_{in} : $P_1(0) + P_1(1) = T$

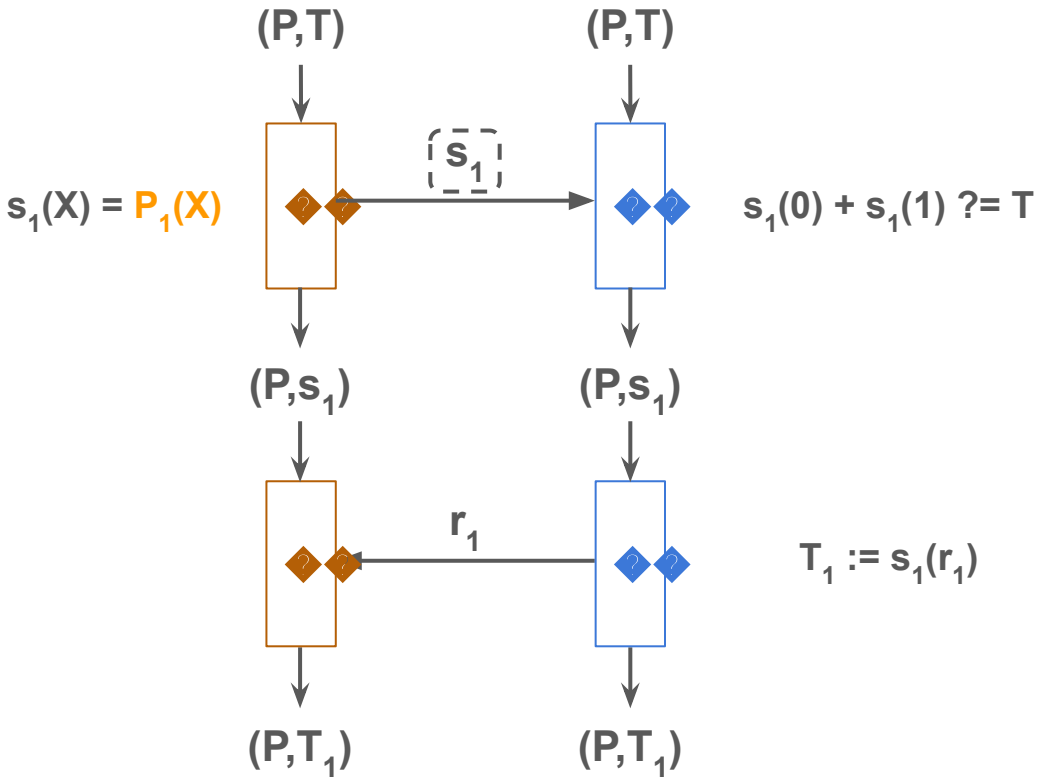
$P_1(X) := P(X,0) + P(X,1)$

is a **virtual** polynomial

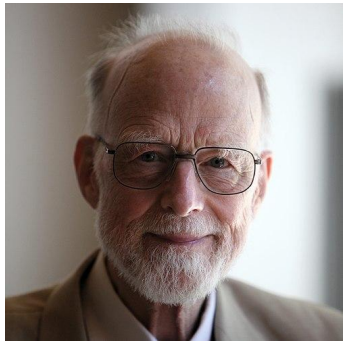
R_{mid}' : $P_1(X) = s_1(X)$

How do we formalize virtual protocols?

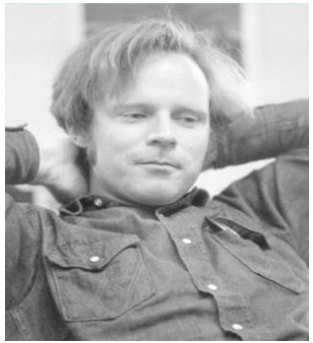
R_{mid} : $P_1(r_1) = s_1(r_1)$



Connection to Hoare-style Program Verification



Tony Hoare



Robert Floyd

Hoare Triples: {P} C {Q}

"Program C, when executed with precondition P, attains postcondition Q"

Hoare Rules:

$$\textsc{Consequence}\quad \frac{F \vdash F' \qquad \vdash \{F'\}\ S\ \{G'\} \qquad G' \vdash G}{\vdash \{F\}\ S\ \{G\}}$$

$$\textsc{Loop}\quad \frac{\vdash \{F \wedge c\}\ S\ \{F\}}{\vdash \{F\}\ \textbf{while}\ c\ \textbf{do}\ S\ \{F \wedge \neg c\}}$$

$$\textsc{Sequencing}\quad \frac{\vdash \{F\}\ S_1\ \{F'\} \qquad \vdash \{F'\}\ S_2\ \{F''\}}{\vdash \{F\}\ S_1; S_2\ \{F''\}}$$

$$\textsc{Conditional}\quad \frac{\vdash \{F \wedge c\}\ S\ \{F'\} \qquad \vdash \{F \wedge \neg c\}\ S'\ \{F'\}}{\vdash \{F\}\ \textbf{if}\ c\ \textbf{then}\ S\ \textbf{else}\ S'\ \{F'\}}$$

$$\textsc{Assignment}\quad \frac{}{\vdash \{F[v \mapsto t]\}\ v := t\ \{F\}}$$

IOR Triples: {R_1} <P,V> {R_2}

"Reduction <P,V>, when executed on inputs satisfying R_1, results in outputs satisfying R_2"

Can we build a program logic for IORs?

Talk Outline

1. Anatomy of IOP-based SNARKs

2. (WIP) Program logic for IORs

3. zkLib’s current development, and next steps

Immediate goals for zkLib

Currently, we have:

1. Definitions of Interactive Oracle Reductions & computable polynomial data types
2. Specification & proof of security for a single round of sum-check

In a few months, we plan to formalize:

1. Composition of IORs, and proofs that they preserve security
2. Executable spec & security proof for the Spartan Polynomial IOP
3. Statements about R-S codes, and specifications of FRI & Plonk

Longer-term goals for zkLib

1. Compilation steps:

   IOPs =[Commitments]=> IPs =[Fiat-Shamir]=> SNARKs

2. Mechanize rewinding knowledge soundness & zero-knowledge

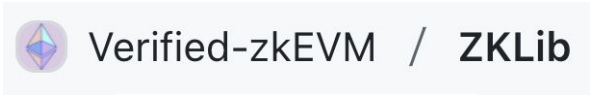
3. Improve verifier's performance (compiled from Lean), *and/or*

   Establish functional equivalence with extracted impl' from Rust

4. Tutorials & onboarding documentation

Summary

- Formally verifying SNARKs is important - and should be done right now!
- We are building **zkLib**, a Lean framework to mechanize your favorite (IOP-based) SNARKs
- **Key ideas:**
  - Reductions as the main building block
  - Leverage existing program verification ideas



Thank you!

(zk)SNARKs: Expectation vs. Reality

- Scaling blockchains
- zkML
- zkIdentity
- Authenticating images
- …

- Under-constrained circuits
- Incorrect Fiat-Shamir
- Insecure protocols
- Missing security proofs
- …

What will be in zkLib in 1-3 months?

1. Specification of ways to compose IORs (sequential composition & virtualization) + proofs that transformations preserve security
2. Case study: specification & proof of the Spartan Polynomial IOP (also: Ligero PCS as a Vector IOP, if time)
3. Correspondence proofs for polynomial data types & executable implementation of Spartan using those types
4. Blueprints for R-S codes, FRI, STIR, WHIR, Plonk, etc.