Shanghai Jiao Tong University

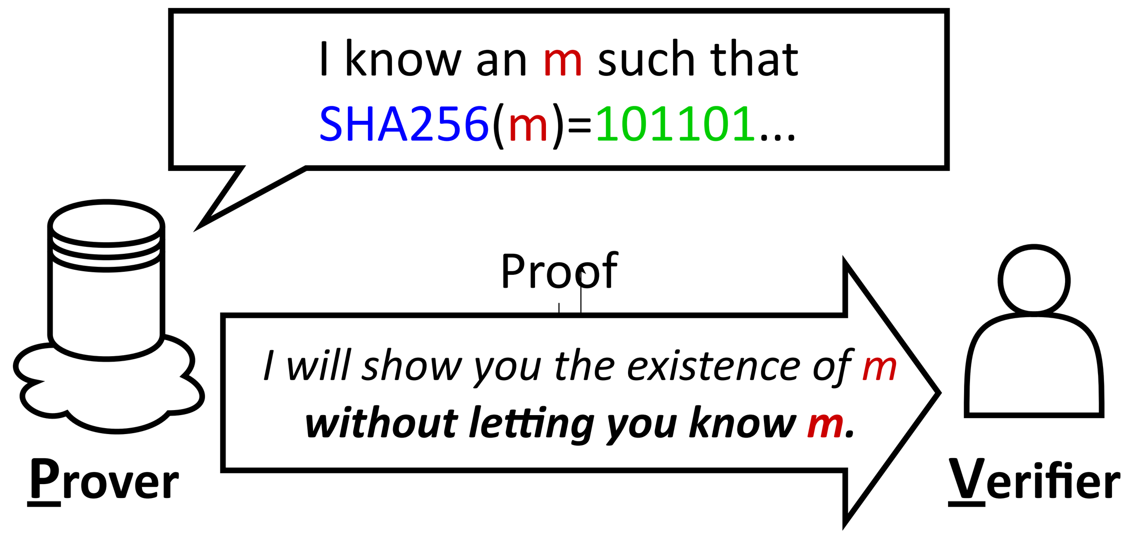
Youwei Zhong @ SPLASH SRC 2024
zhongyouwei@sjtu.edu.cn

A Parameterized Framework for the Formal Verification of Zero-Knowledge Virtual Machines

Supervised by Qinxiang Cao and Yuncong Hu
caoqinxiang@sjtu.edu.cn
huyuncong@sjtu.edu.cn

Zero-knowledge proof (ZKP)

Zero-knowledge proofs allow one party (**prover**) convince another party (**verifier**) that some given statement is true, *without revealing anything beyond the mere fact of that statement's truth.*



Zero-Knowledge Virtual Machine (zkVM):

a kind of virtual machine based on ZKP that allows for **verifiable computation**

Valid input: $\alpha_F := \{x \mid \exists \omega, \text{s.t. } F(x,\omega) = 1\}$

Completeness:

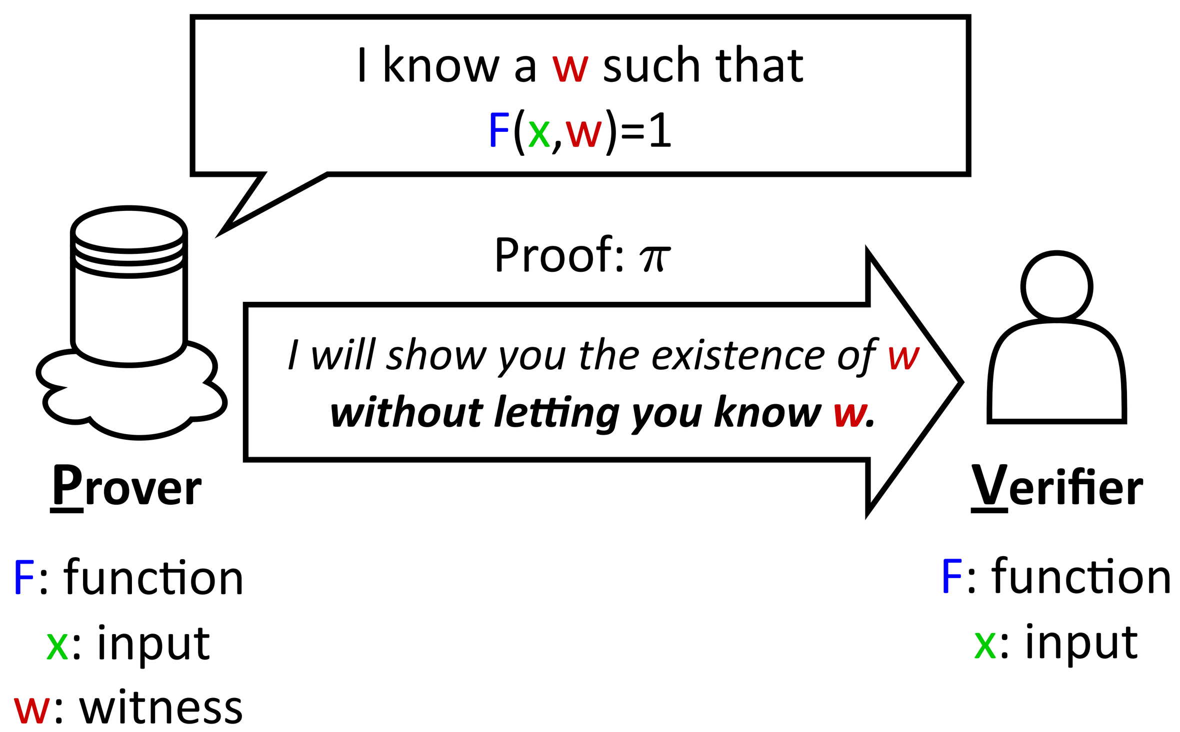
'An honest prover's proof can always pass the verifier's check.'

$$\forall x \in \alpha_F, \forall \omega, F(x,\omega) = 1 \implies$$
$$\Pr[V(x,\pi) = 1 \mid \pi \leftarrow P(x,\omega)] = 1$$

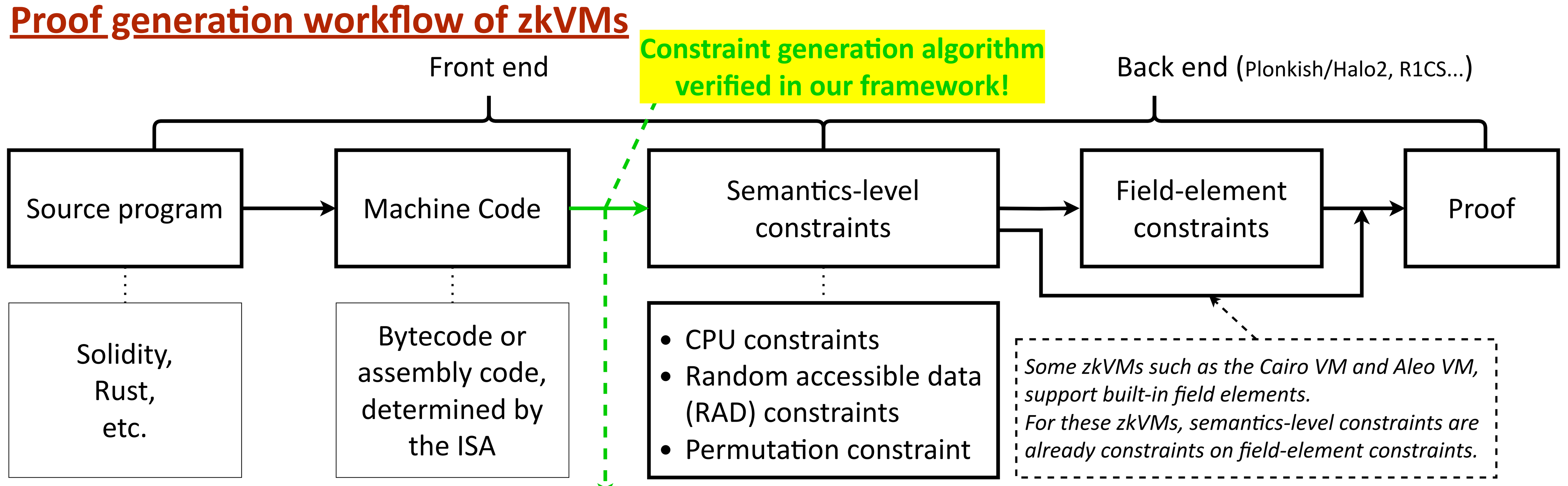
Soundness:

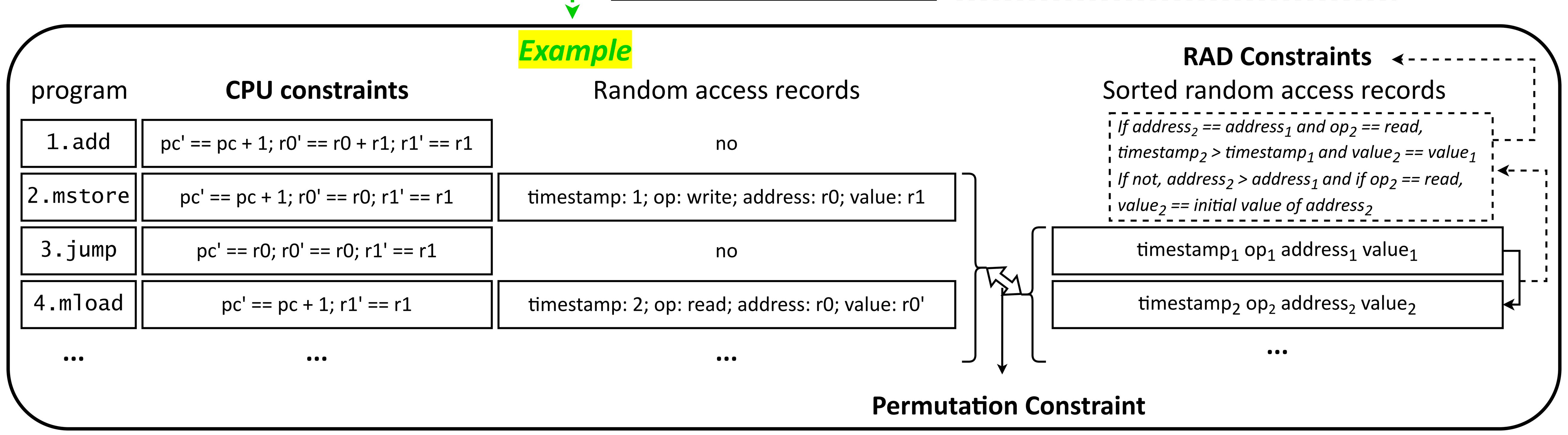
'A malicious prover's proof should be declined with high probability.'

$$\forall P^*, \Pr[x \notin \alpha_F \wedge V(x,\pi) = 1 \mid (x,\pi) \leftarrow P^*] = \text{negl}(|x|)$$



Proof generation workflow of zkVMs



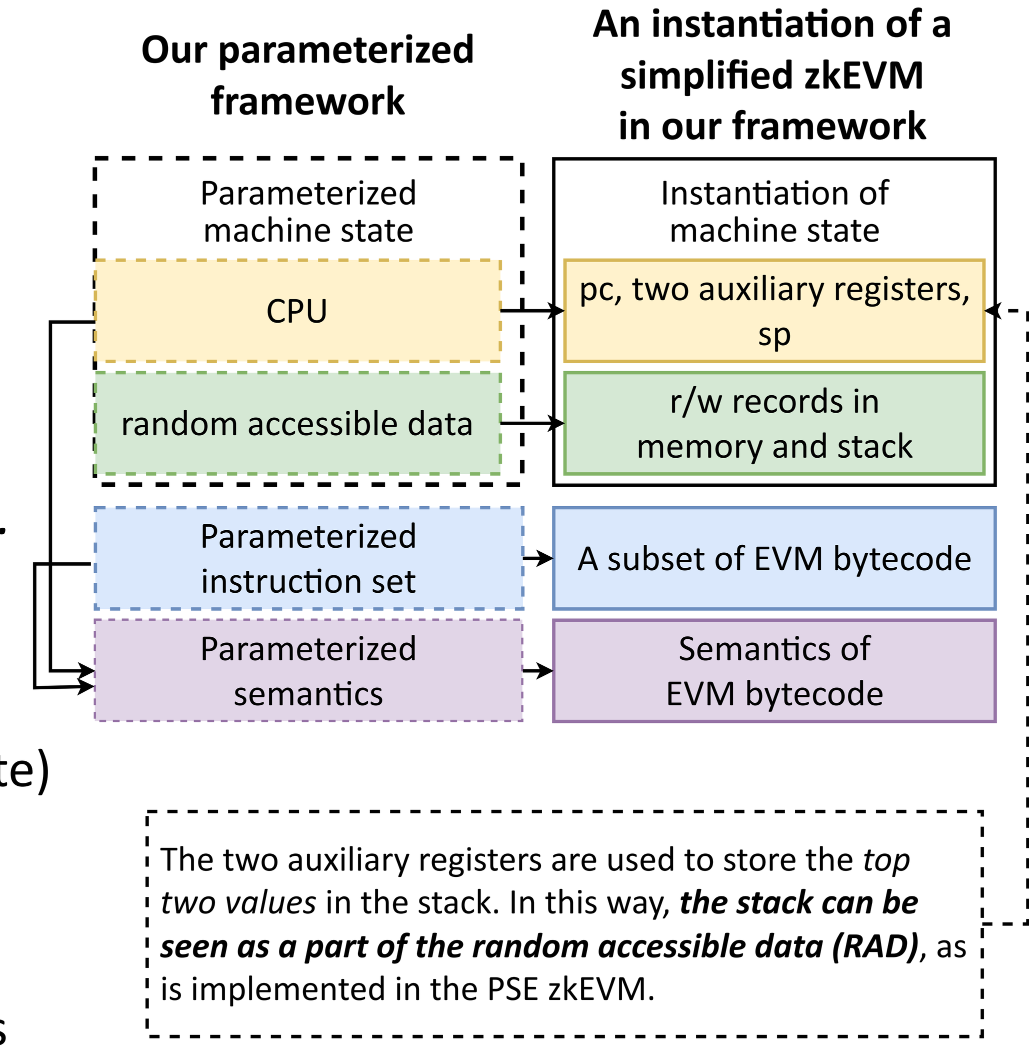


Motivation

- Current zkVMs are susceptible to **bugs** and **vulnerabilities.**
  e.g. A severe bug in Aztec VM's verifier breaks soundness, resulting in **millions of dollars worth of cryptocurrency getting stolen**.
- Existing verification works on zkVMs are **hard-coded** to their memory settings and machine types, which have two demerits:
  1. It is **hard to port** them to other zkVMs. 2. The correctness proof is **not reusable** during development.

Approach

- We *parameterize the ISA (Instruction Set Architecture)*, and define *semantics-level constraints* based on these parameterized definitions.
- Then, we verify the *parameterized constraint generation algorithm.*
- *Two instantiation examples: Cairo VM and a simplified zkEVM.*



From correctness of the constraint generation algorithm to the maintenance of soundness and completeness

x: program, public part of the initial state, initial pc & output(last state)

X: set of all valid x

ω: private input

Ω_1: private part of the initial state & a valid execution trace

Ω_2: private part of the initial state & valid semantics-level constraints

Φ: proof space

$$F : X \times \Omega \to \{0,1\}$$
$$P : X \times \Omega \to \Phi$$
$$V : X \times \Phi \to \{0,1\}$$
$$i_{12} : X \times \Omega_1 \to \Omega_2$$
$$i_{21} : X \times \Omega_2 \to \Omega_1$$

Correctness of the constraint generation algorithm:

$$\forall x \in X, \forall \omega_2 \in \Omega_2, F_2(x,\omega_2) = 1 \implies F_1(x, i_{21}(x,\omega_2))$$
$$\forall x \in X, \forall \omega_1 \in \Omega_1, F_1(x,\omega_1) = 1 \implies F_2(x, i_{12}(x,\omega_1))$$

The maintenance of soundness and completeness:

If there is a pair of algorithms (P_2, V_2), that satisfy soundness and completeness for the set α_{F_2}, there exists a corresponding pair of algorithms (P_1, V_1), that satisfy soundness and completeness for the set α_{F_1}.

Contribution

- **First** to put forward a **parameterized framework** for the formal verification of zkVMs.
  Features:
  1. Verification of the front end and the back end are decoupled.
  2. Different zkVMs can share the same proof.
  3. Proofs can be reused, reducing repetitive code.

  Insights:
  1. Why we verify one phase?
     - Verification of different phases can be combined, which supports *modular design* of zkVMs.
  2. Why zkVMs share this phase?
     - Different zkVMs share the *same constraint generation algorithm*.
- **First** to **formalize the cryptographic security properties of zkVMs**, including **soundness** and **completeness**.
  Only two previous zkVMs verified: *Cairo VM* and *Aleo VM* (not open source).
  They do not realize the difference between maintenance of soundness and completeness and the correctness of the constraint generation algorithm.

Evaluation

The parameterized proof of **soundness** and **completeness** contains about **3800** and **2980** lines of code respectively.

Formal verification of Cairo VM	Instantiation	Soundness	Completeness
Using our parameterized framework	**1092 lines of Coq code**	**No extra efforts!**	**No extra efforts!**
Not using our parameterized framework	/	3266 lines of Lean code	Not proved