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

Youwei Zhong

Shanghai Jiao Tong University

October 30, 2024

Table of Contents

What are zkVMs?

Zero-Knowledge Virtual Machine (zkVM): a kind of virtual machine based on Zero-knowledge Proof (ZKP) that allows for verifiable computation.

What are zkVMs?

Zero-Knowledge Virtual Machine (zkVM): a kind of virtual machine based on Zero-knowledge Proof (ZKP) that allows for verifiable computation.

What is a 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.

Example: What is a ZKP?

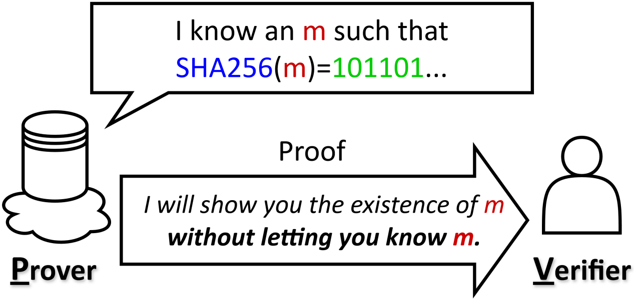

Figure: Example of convincing the Verifier the hash value of some files

Example: What are zkVMs?

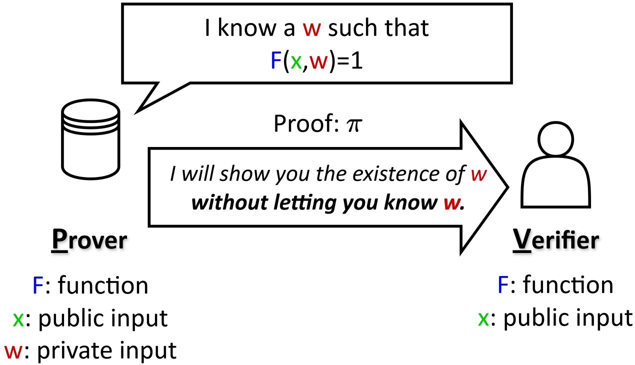

Figure: Example of a zkVM

What are zkVMs?

A zkVM is a ‘virtual machine’ that can generate a proof for the correct execution of arbitrary programs without revealing anything beyond the mere fact of the program and the public input.

What is the workflow of zkVMs?

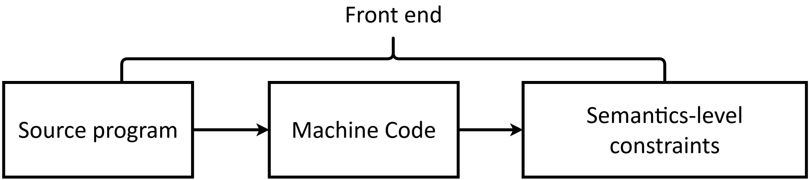

Figure: Front end of a zkVM

What is the workflow of zkVMs?

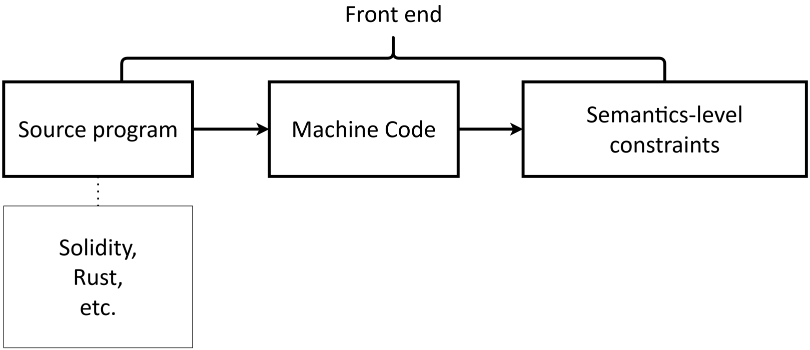

Figure: Front end of a zkVM

What is the workflow of zkVMs?

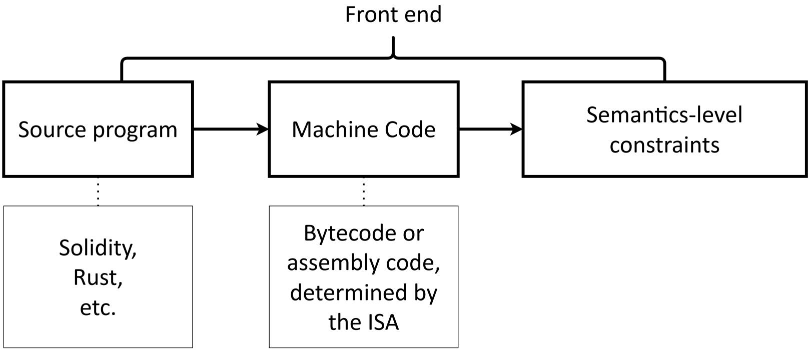

Figure: Front end of a zkVM

What is the workflow of zkVMs?

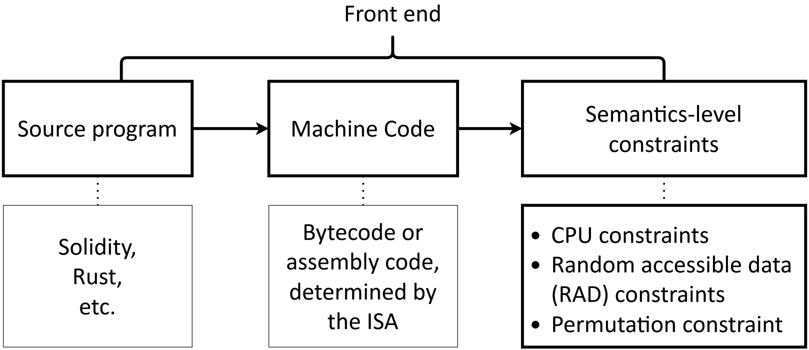

Figure: Front end of a zkVM

What is the workflow of zkVMs?

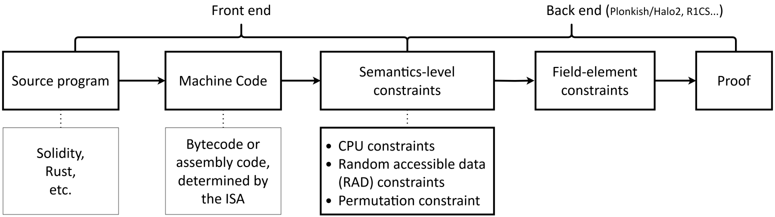

Figure: Workflow of a zkVM

What is the workflow of zkVMs?

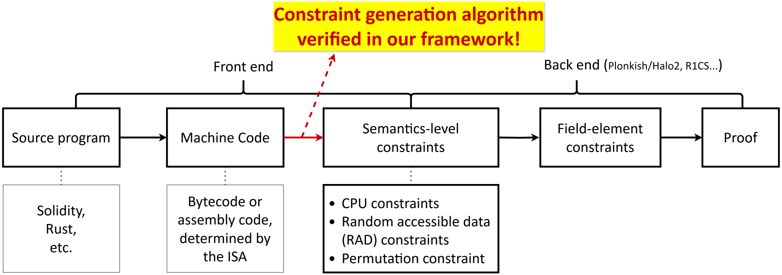

Figure: Workflow of a zkVM

What is the workflow of zkVMs?

Constraint generation algorithm verified in our framework!

Machine Code → Semantics-level constraints

Bytecode or assembly code, determined by the ISA

- CPU constraints
- Random accessible data (RAD) constraints
- Permutation constraint

Why zkVMs use this contraint generation algorithm?

- To make ZKP succinct!

Why zkVMs use this contraint generation algorithm?

- To make ZKP succinct!
- How? The algorithm separates CPU states from the random accessible data (like memory, or a very large stack).

Example: contraint generation algorithm

program

```
1.add
2.mstore
3.jump
4.mload
...
```

Figure: Machine code program

Example: contraint generation algorithm

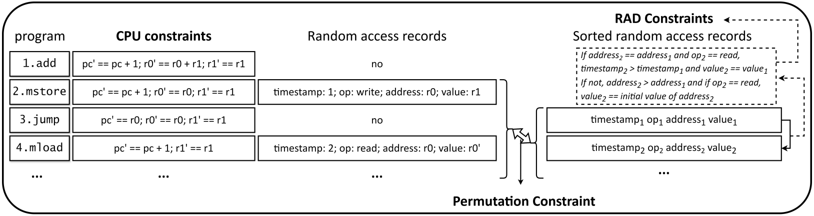

Figure: Example: How does the contraint generation algorithm work?

Why succinctness matters?

- Zero-knowledge is not all that matters in the application of zkVMs.

Why succinctness matters?

- Zero-knowledge is not all that matters in the application of zkVMs.
- One of the most famous zkVM: zkEVM (Zero-Knowledge Ethereum Virtual Machine)

Why succinctness matters?

- Zero-knowledge is not all that matters in the application of zkVMs.
- One of the most famous zkVM: zkEVM (Zero-Knowledge Ethereum Virtual Machine)
- To avoid extra energy cost of rerunning previous smart contracts on the blockchain, zkEVM put ZKPs of smart contract programs on the Ethereum blockchain instead.

Motivation

- Why do we verify zkVMs?

Motivation

- Why do we verify zkVMs?
- Which properties of zkVMs do we want to verify?

Motivation

- Why do we verify zkVMs?
- Which properties of zkVMs do we want to verify?
- Why do we want to verify these properties?

Why zkVMs? Which properties?

- The main purpose of zkVMs is verifiable computation.

Why zkVMs? Which properties?

- The main purpose of zkVMs is verifiable computation.
- Privacy is actually the extra (not necessary) feature of zkEVM.

Recap: example of zkVMs

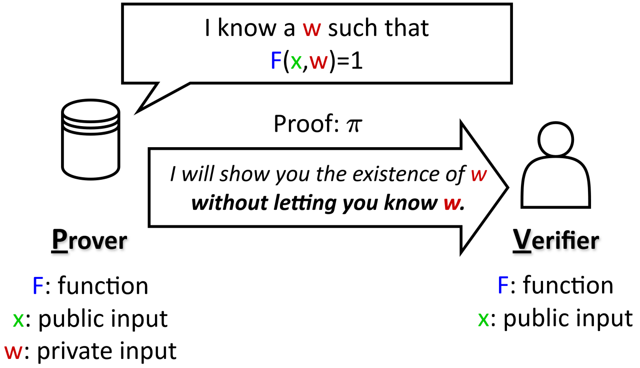

Figure: Example of a zkVM

When we say verifiable computation...

- 'An honest Prover's proof can always pass the Verifier's check.'

When we say verifiable computation...

- 'An honest Prover's proof can always pass the Verifier's check.'
- 'A malicious prover's proof should be declined with high probability.'

Which properties of zkVMs do we want to verify?

- Completeness: 'An honest Prover's proof can always pass the Verifier's check.'

Which properties of zkVMs do we want to verify?

- Completeness: 'An honest Prover's proof can always pass the Verifier's check.'
- Soundness: 'A malicious Prover's proof should be declined with high probability.'

Why do we verify zkVMs and these properties?

- Current zkVMs are susceptible to bugs and vulnerabilities.

[1] Michael Connor, Jonathan Wu, and Ariel. *Disclosure of recent vulnerabilities - HackMD*. [Online; accessed 23. Apr. 2024]. Apr. 2024. URL: https://hackmd.io/@aztec-network/disclosure-of-recent-vulnerabilities.

Why do we verify zkVMs and these properties?

- Current zkVMs are susceptible to bugs and vulnerabilities.
- Example: A severe bug in Aztec VM's verifier breaks soundness, resulting in millions of dollars worth of cryptocurrency getting stolen![1].

[1] Connor, Wu, and Ariel, *Disclosure of recent vulnerabilities - HackMD*.

Our ultimate goal

- End-to-end formal verification of zkVMs!

Our ultimate goal

- End-to-end formal verification of zkVMs!
- We start with verifying one common phase of all zkVMs, the constraint generation algorithm.

Typical zkVMs

The table below displays the differences and similarities among four typical zkVMs[2], they all follow the proof generation workflow defined before, and share the same constraint generation algorithm.

Aspect	PSE zkEVM	Cairo VM	ZKWASM	Miden VM
Machine Type	Stack Machine	Register Machine	Stack Machine	Stack Machine
Instruction Set	EVM Bytecode	Cairo Assembly (CASM)	WebAssembly (WASM)	Miden Assembly
Memory Model	Random Access Memory	Read-only Memory	Random Access Memory	Random Access Memory
Built-in field elements	No	Yes	No	Yes

[2] Pérez Carlos et al. *zkEVM Community Edition - Privacy & Scaling Explorations*. [Online; accessed 21. Mar. 2024]. Mar. 2024. URL: https://pse.dev/en/projects/zkevm-community; Lior Goldberg, Shahar Papini, and Michael Riabzev. *Cairo – a Turing-complete STARK-friendly CPU architecture*. Cryptology ePrint Archive, Paper 2021/1063. https://eprint.iacr.org/2021/1063. 2021. URL: https://eprint.iacr.org/2021/1063; Sinka Gao et al. "ZAWA: A ZKSNARK WASM Emulator". working paper or preprint. Mar. 2023. URL: https://hal.science/hal-03995514; Bobbin Threadbare et al. *miden-vm*. [Online; accessed 23. Apr. 2024]. Apr. 2024. URL: https://github.com/0xPolygonMiden/miden-vm.

Previous work: Aleo[3] (Not Open Source)

- Need to verify the proof produced by the compiler each time it runs a program.

[3] Alessandro Coglio et al. *Compositional Formal Verification of Zero-Knowledge Circuits*. Cryptology ePrint Archive, Paper 2023/1278. https://eprint.iacr.org/2023/1278. 2023. URL: https://eprint.iacr.org/2023/1278.

Previous work: Aleo[3] (Not Open Source)

- Need to verify the proof produced by the compiler each time it runs a program.
- Aleo's front-end language, Leo, is Turing-incomplete.

[3] Coglio et al., *Compositional Formal Verification of Zero-Knowledge Circuits.*

Previous work: Aleo[3] (Not Open Source)

- Need to verify the proof produced by the compiler each time it runs a program.
- Aleo's front-end language, Leo, is Turing-incomplete.
- Hard-coded: targeted towards specific high-level languages and instruction set.

[3] Coglio et al., *Compositional Formal Verification of Zero-Knowledge Circuits.*

Previous work: Cairo[4] (Open Source)

- Only have soundness proof.

[4] Jeremy Avigad et al. "A Proof-Producing Compiler for Blockchain Applications". In: *14th International Conference on Interactive Theorem Proving (ITP 2023)*. Ed. by Adam Naumowicz and René Thiemann. Vol. 268. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2023, 7:1–7:19. ISBN: 978-3-95977-284-6. DOI: 10.4230/LIPIcs.ITP.2023.7. URL: https://drops-dev.dagstuhl.de/entities/document/10.4230/LIPIcs.ITP.2023.7; Jeremy Avigad et al. "A verified algebraic representation of cairo program execution". In: *Proceedings of the 11th ACM SIGPLAN International Conference on Certified Programs and Proofs*. CPP 2022. Philadelphia, PA, USA: Association for Computing Machinery, 2022, pp. 153–165. ISBN: 9781450391825. DOI: 10.1145/3497775.3503675. URL: https://doi.org/10.1145/3497775.3503675.

Previous work: Cairo[4] (Open Source)

- Only have soundness proof.
- Also hard-coded.

[4] Avigad et al., "A Proof-Producing Compiler for Blockchain Applications"; Avigad et al., "A verified algebraic representation of cairo program execution".

Previous work: Cairo[4] (Open Source)

- Only have soundness proof.
- Also hard-coded.
- Read-only memory.

[4] Avigad et al., "A Proof-Producing Compiler for Blockchain Applications"; Avigad et al., "A verified algebraic representation of cairo program execution".

Summary for previous works

- Need to prove soundness and completeness for every change of the zkVM.

Summary for previous works

- Need to prove soundness and completeness for every change of the zkVM.
- Proof not directly portable to other zkVMs.

What did we do?

- We parameterize the ISA (Instruction Set Architecture), and define semantics-level constraints based on these parameterized definitions.

What did we do?

- 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.

What did we do?

- 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.

Instantiation example

The figure below shows the instantiation of a simplified zkEVM:

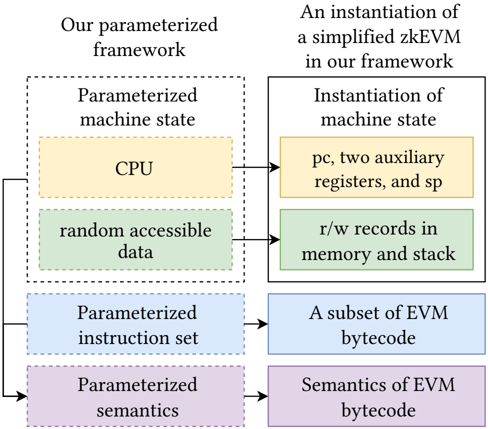

Figure: An instantiation in our parameterized framework

Features

- Verification of the front end and the back end are decoupled.

Features

- Verification of the front end and the back end are decoupled.
- Different zkVMs can share the same proof.

Features

- Verification of the front end and the back end are decoupled.
- Different zkVMs can share the same proof.
- Proofs can be reused, reducing repetitive code.

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

Figure: Comparison of verifying Cairo VM

Recap: soundness and completeness

- Completeness: 'An honest Prover's proof can always pass the Verifier's check.'
- Soundness: 'A malicious Prover's proof should be declined with high probability.'

Recap: soundness and completeness

- Completeness: 'An honest Prover's proof can always pass the Verifier's check.'
- Soundness: 'A malicious Prover's proof should be declined with high probability.'
- Does the correctness of the constraint generation algorithm induce the maintenance of completeness and soundness?

Maintenance of completeness and soundness

Suppose we have the completeness and soundness of the back end:

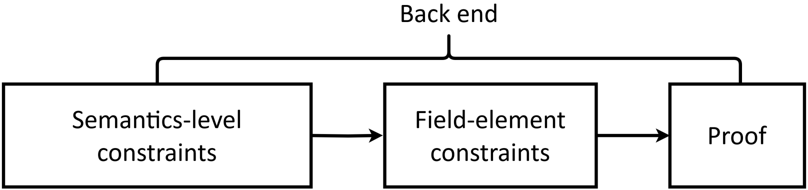

Figure: Back end of zkVMs

Maintenance of completeness and soundness

Suppose we have the completeness and soundness of the back end, which means:

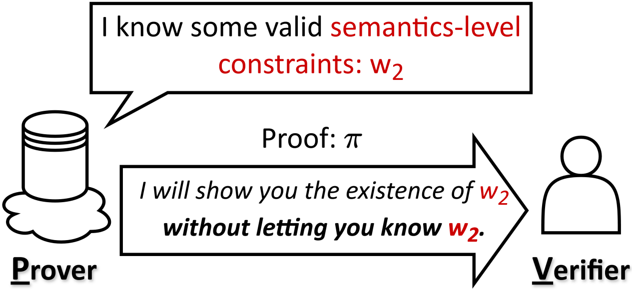

Figure: Existence of a ZKP system for semantics-level constraints

Maintenance of completeness and soundness

Correctness of the constraint generation algorithm should induce the existence of the following ZKP system:

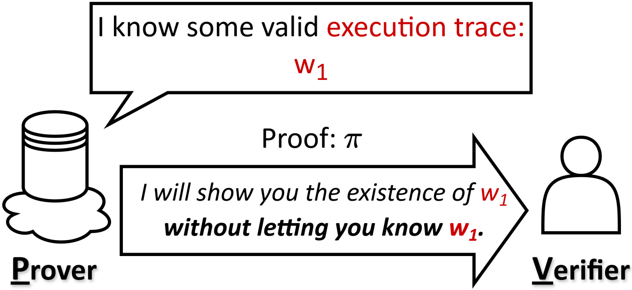

Figure: Existence of a ZKP system for the execution trace

Maintenance of completeness and soundness

- We formalize the soundness and completeness of ZKP systems.
- We prove that the correctness of the constraint generation algorithm induces the maintenance of soundness and completeness.

Contributions

- We are the first to put forward a parameterized framework for the formal verification of zkVMs.

Contributions

- We are the first to put forward a parameterized framework for the formal verification of zkVMs.
- We are the first to to formalize the cryptographic security properties of zkVMs, including soundness and completeness.

Future Work

- Verify maintenance of zero-knowledge and knowledge soundness during the transformation using the constraint generation algorithm.

Future Work

- Verify maintenance of zero-knowledge and knowledge soundness during the transformation using the constraint generation algorithm.
- Verify the back end of zkVMs.

Acknowledgement

- Supervised by Qinxiang Cao (caoqinxiang@sjtu.edu.cn) and Yuncong Hu (huyuncong@sjtu.edu.cn).
- Funded by Ethereum Foundation FY24-1541.