Towards a zkVM architecture whitepaper

EF Cryptography Team

29 January 2026

This document outlines requirements for a "*zkVM architecture whitepaper*" covering the zkVM circuit-bus architecture, recursion and final proof derivation.

The goals of such a document are:

- to ensure zkVM designs are documented with enough precision for independent review
- to provide a security argument on why the overall architecture is secure

The requirements below apply to **STARK-based zkVMs using the RISCV ISA**. Teams with other zkVM designs should reach out to the Ethereum Foundation Cryptography team to discuss any needed adjustments.

1. Background

1.1. The zkVM Paradigm

This document is based on the assumption that the zkVM candidates operate in the following paradigm:

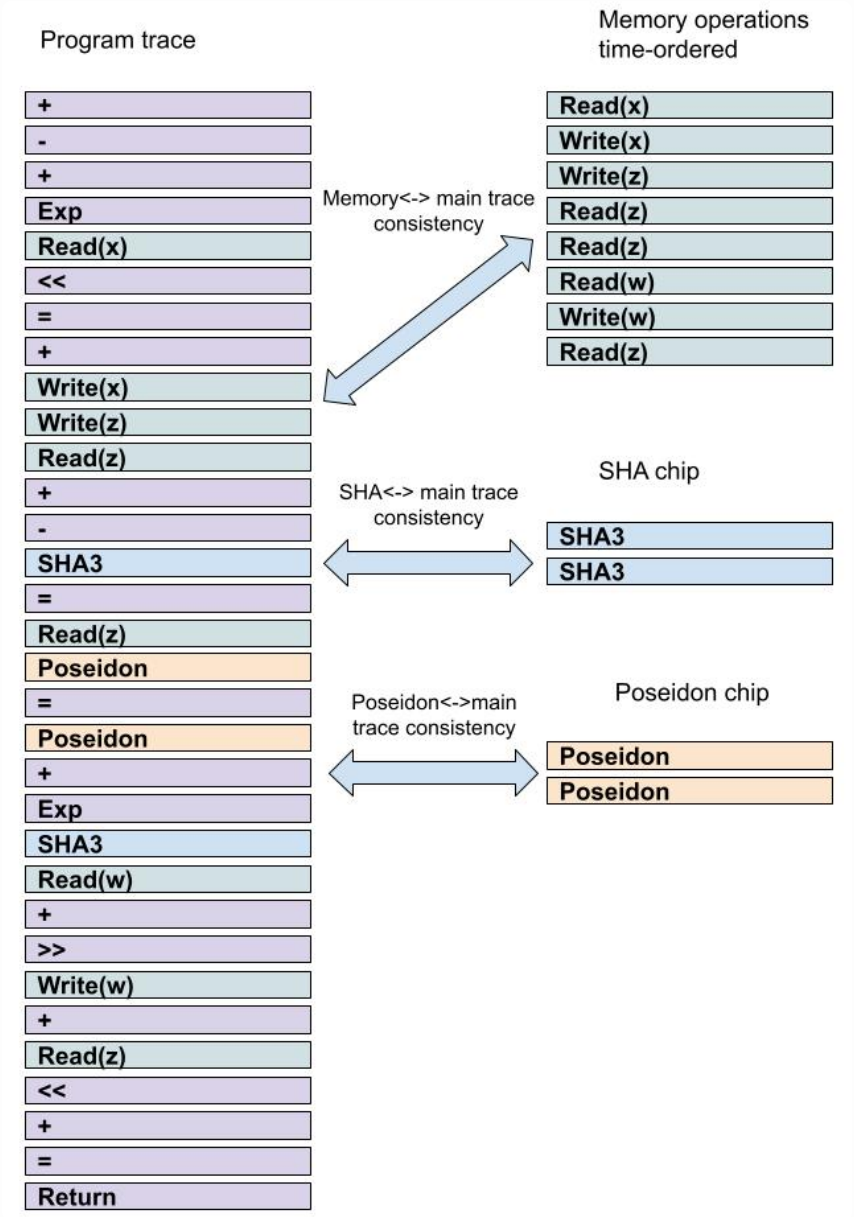
1. They prove the correctness of a single RISCV *guest program*. The program *trace* totals up to a billion instructions in size.
2. As proving the program in its entirety is too slow and not scalable, the trace is split into *segments* (also known as chunks or shards in some zkVMs). These *segments* are proven those separately, and aggregate results.
3. As the operations from the supported instruction set vary significantly, similar operations are clustered into specialized *chips*. A *proof* for a single part/segment may encompass several chips, but it may also happen that a single chip requires a dedicated, *external* circuit – and a separate proof.
4. In order to produce a proof, a single segment (or a group of chips) is transformed into a *circuit*: a sequence of algebraic operations over its internal variables. Then the validity of the circuit boils down to the satisfiability of algebraic relations, which is proven with some variant of a STARK protocol.
5. The RISCV program operates with random-access memory (RAM). As RAM can not be implemented in a circuit as is, the memory read/write operations are transformed so that their correctness must be enforced by another proof.

6. The algebraic relations mentioned above are not sufficient to enforce the correctness of all instructions, and additional arguments such as range checks are needed. Those arguments and memory correctness arguments are often represented as algebraic arguments (e.g. lookups) and then merged with algebraic relation arguments in a single proof.
7. In order to shorten the verification time and to save the bandwidth, the zkVM additionally makes a "proof of proofs", which encompasses the proofs for individual segment or chips.
8. This "final" proof is made in a recursive or tree-like manner, where individual proofs are leafs, internal tree nodes aggregate the siblings, and the root is the final proof.

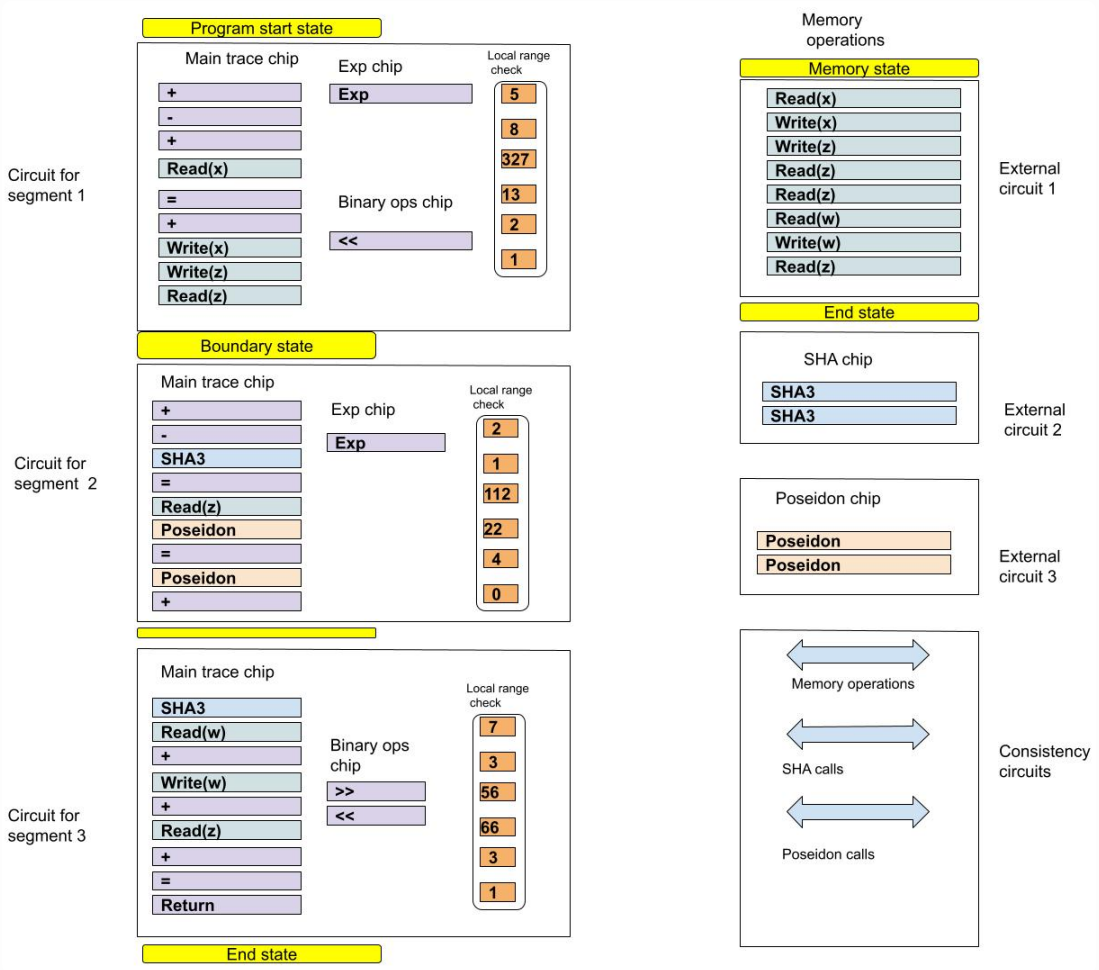
1.2. Vanilla zkVM structure

In order to make the paradigm more apparent, we illustrate our view of a simple zkVM adhering to it in the following diagrams. We ask every zkVM design that differs substantially from that view to reflect the differences in the whitepaper.

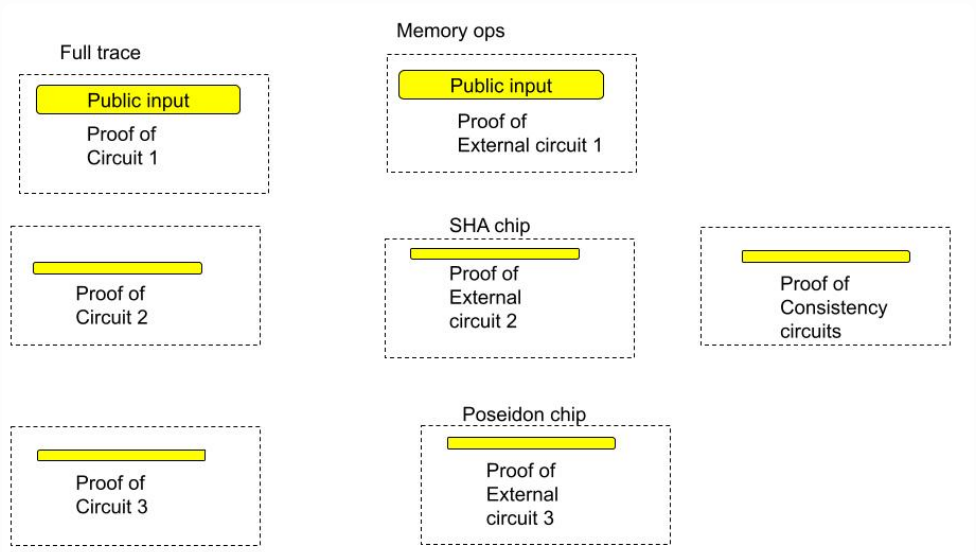
Grouping memory and expensive operations into separate chips:



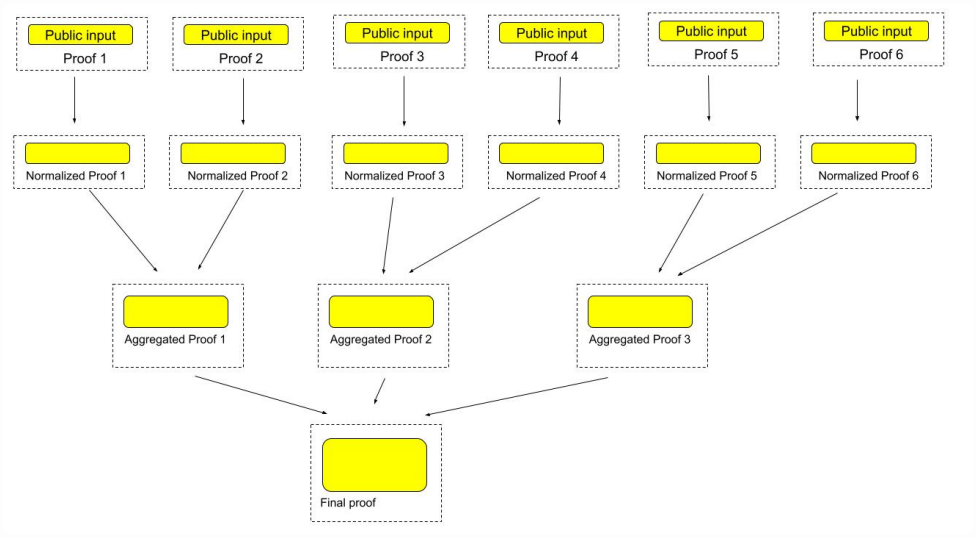
Segmentation and circuitization:



Generating individual proofs:



Recursive aggregation of proofs into a final proof. Here we assume that the aggregation graph may contain some auxiliary nodes, which transform or "normalize" proofs for more efficient aggregation:



2. Motivation

2.1. Goals

The *"zkVM architecture whitepaper"* serves two purposes:

- Enable cryptographers to reason about the completeness and soundness of the final proof. Whereas the whitepaper would not be sufficient to reason about the security *alone*, we should be able to argue the security *assuming* that the implementation of all the components described in the whitepaper is correct.
- Enable the security auditors of a zkVM codebase and/or architecture to verify the subject against the whitepaper rather than against the more general RISCV standard. Thus the whitepaper should be detailed and non-ambiguous enough to make the auditor's job significantly easier. The OpenVM whitepaper is a good example of the target level of detail and technical rigor expected.

We recognize that reaching this level of rigor requires real effort. The goal is shared confidence in zkVM security, benefiting both teams and the broader ecosystem. We are here to help you get there.

2.2. Scope

Assuming the paradigm and our goals described above, we presume that to be sufficient for a confident reasoning on the zkVM design security the document should cover the following topics:

- How the big program is split into segments.
- How the executed instructions are rearranged into chips, and the interaction between chips.

- How circuits are built around the chips and the segments, and how the logic of the segmentation and of partitioning into chips is reflected in the relations to be proven.
- What are additional mechanisms, if any, that do not fit into the simple logic described above directly, but are required to assert the correctness of the program –- and how these mechanisms are represented in circuits (e.g. memory checking arguments) and interact with each other.
- How individual proofs are aggregated into one final proof so that the statements of those proofs are correctly merged and altogether constitute the statement on the full program correctness.

2.3. Timeline

In order to provide timely feedback and to approach the complexity of the design in an iterative manner, we break the whitepaper preparation into three phases with the following milestones:

1. **W1: May 1, 2026** — Architecture overview (Section 3)
2. **W2: September 1, 2026** — Architecture details (Section 4)
3. **W3: December 1, 2026** — Security argument (Section 5)

In the rest of this document, we provide concrete requirements for **W1**.

Requirements for **W2** and **W3** will be published later, allowing us to tailor expectations based on communication with teams and M1 feedback.

3. W1: Architecture Overview

Due: May 1, 2026

This phase covers the high-level architecture: an overview of segmentation and recursion. It provides the foundation for deeper work in phases M2 and M3.

3.1. Execution model

Here we expect the preliminaries necessary to introduce further concepts:

- Notion of program, program execution, VM state;
- List of supported instructions;
- Statement about the program execution to be asserted in the final proof

3.2. Segmentation

Here we expect a description of how the zkVM transforms the program execution trace into individual segments for further proving as circuits constraining a finite execution segment:

- How execution is represented as one or more execution traces (*segments/chunks/shards*).
- Whether the segments are further partitioned into other blocks (*chips*) dealing with specific instructions.
- Whether some instructions are viewed as *precompiles* that their chips are proven separately, and how this is represented in the architecture.

3.3. Interactions

The document should describe **how memory is handled in zkVM** w.r.t. to segmentation:

- different types of memory handled (local-global, ROM-RAM, etc)
- Which argument type is used for consistency (permutation, lookup/logup, etc.) and where the randomness for the argument comes from (if used).

If, apart from the memory organizations, the zkVM is using the *bus model* to describe the interaction between chips and/or segments, then **the document should overview how the bus is organized** and how the consistency between the entities is enforced. Concretely:

- List the buses;
- What argument type is used for consistency.

3.4. Individual proofs

The document **should explain the individual proof creation** to the level where the parameters used by the particular zkVM in their soundcalc integration become apparent:

- Overview of how circuits are created out of segments and chip groups;
- If circuits are grouped by types due to different size, security level, proof size, etc. –- list those types and explain how many circuits of each type appear.
- All the commitment schemes and argument types (FRI-based, lookup-based, etc.) used.
- If any, the additional subprotocols used to prove correct execution of a segment, e.g. lookup arguments for the range checks.

3.5. Recursion structure

If recursion is used, the document should explain:

- The recursion topology (linear chain, tree, configurable fan-in).
- What each recursion step generally proves.
    - In particular, it should specify the NP relation of each node in the tree. The goal is to clarify how the relations get aggregated throughout the recursion and altogether yield the final relation of the full program. This item can be done in an informal fashion, as long as the result is informative.
- Mapping between nodes in the recursion topology and the circuit instances (i.e. names) in soundcalc.

4. W2: Architecture Details

Precise requirements to be published June 1, 2026. Deliverable due September 2026.

Building on the architecture overview, this phase will request deeper technical details on the bus organization, memory handling, instruction fetching, how segments are converted into circuits, as well as deeper questions on the recursive architecture. The goal is to reach a level of precision where each component's role in the overall proof system is unambiguous.

5. W3: Security Argument

Precise requirements to be published July 1, 2026. Deliverable due December 2026.

This final phase focuses on the security argument: a proof sketch demonstrating that the zkVM implements a sound argument of knowledge. This should show why, if the final proof verifies, there exists a valid execution of the original program respecting VM semantics.

Specific requirements will be informed by the architecture details from earlier phases. We will provide guidelines, as well as an example, on how such a security argument should look.