TL;DR:

- =nil;'s zkLLVM compiler and Proof Market enable arbitrary, trustless computation; =nil; is using them both to bring zkSharding to Ethereum.
- zkSharding enables parallel EVM shards to execute and settle transactions to a unified execution environment; powering cheaper transactions with greater throughput than other approaches.
- The zkLLVM compiler can build a fully compatible zkEVM, which eliminates manual overhead in circuit adjustments thus future-proofing our zkEVM. Importantly, this method also shifts the crux of circuit security vulnerabilities from human error to an auditable and automatic process via the LLVM.
- While automatic circuit compilation is less performant today, we have direct sight of circuit compilation improvements that will meaningfully improve performance.

zkEVMs (aka a zero-knowledge EVM) scale Ethereum by offloading the execution of transactions to a specialized execution layer (aka a rollup). Rollups execute, order, bundle and compress transactions and post the bare minimum amount of information to the Ethereum layer (L1). Core to a rollup is the security mechanisms whereby the rollup proves it correctly executed transactions to Ethereum L1, which in the case of a zk-Rollup, is done via a zero-knowledge proof. Rollups are critical to achieving higher throughput, bandwidth, and expressiveness of Ethereum.

Since rollups were first introduced nearly a decade ago, a variety of zkEVM types have been proposed and are now in various phases of development. Below, we review the landscape of current scaling solutions pushing Ethereum forward and then introduce our solution, a zkEVM powered by the =nil; zkLLVM and Proof Market.

The zkEVMs

Vitalik Buterin offers a useful classification of zkEVMs indexed on compatibility with Ethereum's base execution environment, a C++ version named evmone. This sort of classification helps us understand metrics around technical alignment (e.g., composability), security, upgradeability, and performance relative to Ethereum, but does not specify or prescribe principles to follow.

Vitalik classifies the zkEVMs across a spectrum of most compatible but slowest to generate proofs (Type 1s) to less compatible but more performant proof times (Type 2 and Type 3) to the least compatible but the most performant (Type 4s).

Because a Type 1 is identical to base layer bytecode, it's broadly regarded as the "holy grail" of the zkEVMs. Type 1s can verify Ethereum blocks and works out of the box with existing apps/tools. It requires no adjustments to existing code and minimal auditing.

All types of zkEVMs are assessed based on their circuits, which define computations and compression quality. So far, most types are defined with either a Halo 2 proof system and PLONK arithmetization or a STARK proof system and AIRs arithmetization. All of these systems or types, rather than using independent development languages rely on libraries of zk-specific logic expression. This gives all types of zkEVMs common drawbacks.

1. Defining circuits is costly:

defining a circuit takes years and lots of resources. Oftentimes it requires a newzkDSL and suffers from poor upgradability.

1. Circuits are complex:

as rollups age they will add features, requiring changes to existing circuits. New complexity compounds risks and costs associated with the rollup.

1. Manually defined circuits are insecure:

Manual circuits rely on humans and use young, untested libraries (many have warned againstmanually defined circuits).

1. Auditing circuits is hard:

The complexity and challenge of auditing circuits means auditing will never be very accessible or inexpensive. This will only compound with time.

zkEVM on =nil;'s zkLLVM

At =nil; we recognize that fostering trust with our community requires time, commitment and transparency, which is why we're opting for a methodical and public release schedule.

=nil;'s zkEVM is a Type 1 that compiles to our zkLLVM. It leverages evmone (the only production tested implementation of Ethereum) to ensure security and complete consistency with the base layer. The zkLLVM (C++ or Rust) compiles down to the circuit, removing human error from the circuit definition process.

Besides security and compatibility, compiling to a circuit significantly enhances flexibility, as it means new EIPs (such as 4844) can be added at the VM level and recompiled down to the zkLLVM circuit instead of requiring re-writing circuits manually, which is costly, slow, imprecise, and harder to audit.

Most importantly, the zkLLVM affords us the flexibility to streamline significant modifications to the EVM, such as parallelization or additions like transient storage. Whereas manual adjustments require laborious review to identify the affected parts of a circuit and manually alter them without compromising composability or the general functionality of the EVM.

A note on performance

Improvements such as enhanced security and flexibility always come with tradeoffs. Today our zkEVM – while performant – lags manually defined circuits in terms of the number of constraints by about 10-15%. This is because when manually writing a circuit a developer can densely pack a circuit with gates. Fortunately, it's possible to include circuit density as a parameter for our compiler optimization, which our research team is diligently working on. Additionally, this performance gap will close further thanks to proof system optimizations and hardware acceleration efforts made by a growing list of professional teams.

Benchmark performance is slightly lower.

Final thoughts

Scaling Ethereum (and blockchains generally) involves batching data, producing greater throughput at a lower cost per transaction. This improvement cannot come at the expense of security, auditability, accessibility, or the future proofness of the system. We're excited to bring the =nil; Foundation's zkEVM to market, which we believe makes the appropriate security and performance tradeoffs coupled with a sharded design under the hood to scale Ethereum.

For further specification details, see the published spec, =nil;'s zkEVM1 A Secure Updatable Type-1 zkEVM. Additionally, we're excited to share the Github repository that includes an initial implementation. Keep an eye out for future updates regarding our zkEVM with extensions in the new year.