Innovation in the domain of zero knowledge cryptography and its applications have enabled impressive advancements for scaling and privacy in Eth 1.0. Existing applications such as ZK-rollup and coin mixers have already taken hold in Eth1 and will be deployed and innovated upon in Eth 2.0. Ensuring that wasm as an execution layer is sufficiently performant to handle the rigorous computational demands of the underlying cryptographic building blocks behind these applications is an important task in assessing the viability of wasm as the execution layer for Eth2.0.

In this article, wasm implementations of two algorithms which form key components in existing ZK applications on Eth1.0 are presented and benchmarked. Three benchmark types are presented: benchmarks which use wasm interpreters only, benchmarks which measure execution time of wasm compiled to native and benchmarks which measure interpreted wasm code calling into a native API for arithmetic operations. All benchmarks were performed on an Azure D2s v3 instance and measure wasm execution time (and not instantiation or compilation).

MiMC

MiMC is a relatively-new hash function which can be represented with minimal complexity inside ZKSNARK circuits (compared to many conventional hash functions). It is used as a key component in the design of existing Dapps such as Tornado cash and ZK rollup implementations.

An implementation of MiMC has been developed in Assemblyscript for benchmarking purpose. Because assemblyscript is low-level, the wasm that it compiles to is relatively close to the source code.

Benchmarks

As a baseline for comparison against the EVM, a deposit into Tornado cash costs ~1,000,000 gas, the vast majority of which is spent computing 20 MiMC hashes to compute a merkle root.

A gas limit of 10,000,000 gas and a target execution time of 300 milliseconds per block results in 1.5 milliseconds per hash as a goal to meet if the wasm implementation of MiMC will be approximately as performant as the EVM one.

Results:

- V8 interpreter: 2.7 seconds
- compiler (v8-liftoff): 0.25 milliseconds
- wabt interpreter with native arithmetic API: 0.174 milliseconds

Groth16 Proof Verification

Groth16 is a performant ZKSNARK proof scheme utilized in existing Eth1 dapps such as Tornado Cash and implementations of ZKRollup.

Websnark is a library written in wasm and designed for fast ZKSNARK proof generation and verification. The library provides fast Webassembly routines for computing elliptic curve pairings and a Javascript implementation of Groth16 verification which calls into these. This code was ported almost directly to Assemblyscript.

Benchmarks

Verification of a groth16 proof in EVM costs approximately 600,000 gas. This yields 18 milliseconds as a target execution time for meeting the current performance of EVM.

Results:

- v8-liftoff: 26 milliseconds
- v8-interpreter: 3.2 seconds
- wabt interpreter with native arithmetic API: 18 milliseconds

Next steps

As the results of benchmarking indicate, the use of a native arithmetic API is shown to have very good performance with presented wasm algorithms able to meet and surpass their EVM counterparts. This is because the bottleneck in most cryptography is in modular arithmetic while other parts are often negligible.

To fully assess the viability of this approach for Eth2.0, a wider variety of cryptographic algorithms must be implemented and benchmarked in wasm. If a native arithmetic API can be proposed which provides sufficient performance for a wide variety of applications while maintaining a minimal trusted computing base, it may be a good option for the execution layer of Eth2.0.

Special thanks to Casey Detrio and Paul Dworzanski of the Ewasm team for their contribution of fast modular multiplication routines, performance optimizations and bug fixes.

Benchmark Source and Instructions for Running Benchmarks

- https://github.com/jwasinger/groth16-verify-wasm
- https://github.com/jwasinger/mimc-wasm
- Instructions for executing benchmarks with Wabt