Prepared by : Garvit Goel, Rahul Ghangas, Jinank Jain

In this article, we will discuss how you can verify Ed25519 signatures on Ethereum today in a very gas-efficient way, without the use of any pre-compiles such as the proposed EIP665. We will use the same principles as used by many zk-rollups. We have already shipped the code for this, albeit not yet audited. Let's get to it.

For dApps that want to verify Ed25519 signatures on Ethereum, rather than verifying the signature(s) directly on Ethereum, (and performing the curve operations inside a solidity smart contract), one can construct a zk-proof of signature validity and verify the proof on-chain instead.

Gas cost for verification of a single Ed25519 signature is about ~500K gas (when the Ed25519 curve is implemented directly in Solidity). On the other hand, the gas cost for verifying a zk-snark on-chain is about ~300k gas. These gas savings become significant when you want to verify a large number of signatures in one batch, then you can just create a single ZK-proof for the entire batch.

At Electron Labs, we have built a circom-based library that allows you to generate a zk-snark proof for a batch of Ed25519 signatures.

You can check out the details of our mathematical approach here. You can even test proof generation today using the APIs given on the previous link.

Check out the complete code base here

The Performance of a Single Ed25519 is as below:

All metrics were measured on a 16-core 3.0GHz, 32G RAM machine (AWS c5a.4xlarge instance).

Single ED25519 Signature

Constraints

2,564,061

Circuit compilation

72s

Witness generation

6s

Trusted setup phase 2 key generation

841s

Trusted setup phase 2 contribution

1040s

Proving key size

1.6G

Proving time (rapidsnark)

6s

Proof verification Cost

~300K gas

Furthermore, for batching we support:

1. Max batch size supported = 99 signatures
2. Proof generation time for a batch of 99 signatures = ~16 minutes.

While these metrics are good for a PoC, we need to do a lot better. Hence, as next steps, we are planning to integrate recursive snarks. This will increase the max batch size and reduce proof generation time by multiple orders of magnitude.

Use Cases:

We believe one of the best use cases for this tech is extending light client bridges to Ethereum. This includes Polygon Avail, IBC and Rainbow Bridge. One could also build zk-rollups that use Ed25519 for user accounts.

Ending Note:

We would love to work with teams that want to use this tech. We are also looking for research groups who are working on recursive snark technology to help us scale our tech.