TLDR:

You can checkout a video demonstration of smart contract compilation here:

Compiling JS to ZK Smart Contracts

One of the biggest challenges for blockchain adoption is how few developers are able to write smart contracts.

Generally speaking, there does not currently exist a smart contract language that is easy for Web2 developers to adopt in mass, and I wanted to share a toy architecture that works with JavaScript smart contracts compiled with our IDE, in the hope that more developers can be onboarded to the ecosystem at large.

Steps

1. Compile each function in a JavaScript class to an arithmetic circuit for the proving system of your choice with public inputs that include a delta merkle proof for any state modified by each function.

2. Generate a Contract Circuit Whitelist

merkle tree whose leaves are the hashes of the verifier data corresponding to each contract function's circuit in the contract.

1. Create an ethereum contract that builds an append-only merkle tree whose leaves are the roots of Contract Circuit Whitelist trees. Any eth user can register a contract in the tree by submitting a contract circuit whitelist root. (Merkle Leaf Index = Contract Address)

2. Create a recursive block circuit which takes the following inputs + constraints to tie all the inputs together:

3. Previous block proof (or a no-op proof of same size/degree/verification topology for the first block)

4. A proof generated by proving a contract class circuit + data needed to verify the proof

5. A merkle proof of the Contract Circuit Whitelist root's inclusion in the L1 merkle tree built by the smart contract

6. A merkle proof of the function circuit proof's verifier data's inclusion within the Contract Circuit Whitelist tree

7. A delta merkle proof(s) of the contract's state tree (for any leaves of the contract's state that get modified)

8. A delta merkle proof of the global state tree's state leaf for this contract (global state tree's leaves contain roots of contract state trees)

Note: The circuit whitelist root, start global state root and end global state root should be public inputs

1. Generate a proof using the recursive block circuit, and use recursive verification to generate an equivalent groth16 proof

2. Submit the proof to an ethereum smart contract which verifies the proof, ensures the circuit whitelist public merkle tree root is the same as the one in our contract registration L1 contract, ensures the start state corresponds to the last end state stored in the contract, and updates the new global state root to the end root public input hash from the proof.

[

Screenshot 2023-05-17 at 9.57.00 PM

1858×764 187 KB

](https://ethresear.ch/uploads/default/original/2X/c/cddb0887740b4501016b1bed15da85bcabe6cedc.jpeg)

In production, our protocol uses a much more complicated architecture for our layer 2 that unlocks significant scalability and UX improvements, but this architecture is able to at least match or exceed existing ZK layer2's scalability and finality speed. We will publish more details on how to achieve these improvements in a future post.

Closing thoughts:

Would love to know if anyone else is interested in developing an EIP for these kinds of trustless state transitions that take advantage of Ethereum consensus.