polygon zkEVM

Architecture - L2 State Tree Concept

v.1.0

December 13, 2023

1 L2 State Tree Concept

1.1 Introduction

In this section, will be discussed the State Tree of Layer 2 within the zkEVM architecture. The main components of the zkEVM architecture in this section are: the **Prover**, the Prover's **HashDB** and the **Executor**.

In Ethereum, ensuring the accuracy of state changes resulting from the execution of transactions is crucial. Within the zkEVM context, the **zkEVM ROM** is a **zkASM program** designed to prove computations for stating the correct Layer 2 State transitions from a given a batch of L2 transactions. We might use multiple zkASM instructions to implement a single zkEVM opcode (in fact, this happens most of the times). Recall that the L2 State is organized as a Merkle Tree, being its root a cryptographic summary of the current state data referred to as the **State Root**. Hence, the **ROM** must have the capability to **correctly** execute CRUD (Create, Read, Update, and Delete) operations on the Merkle Tree representing the current state.

1.2 Introduction to the Storage State Machine

In the zkEVM, a secondary State Machine known as the **Storage State Machine** has been implemented. This machine is specifically designed to generate an execution trace, providing evidence for the creation, reading, updating, and deletion of L2 state data. For example, when we need to retrieve a value from the tree, it's essential to obtain a set of nodes from the tree, referred to as the Merkle proof, ensuring the accuracy of the read state value. Similarly, any operation that modifies the tree must provide proof that the tree modification is executed correctly. After processing the last L2 transaction in a batch, the resulting root will be the new state root.

Let's provide specific illustrative examples (Figure 1) showcasing CRUD operations on the L2 State, being the Storage State Machine responsible for providing proof of their validity:

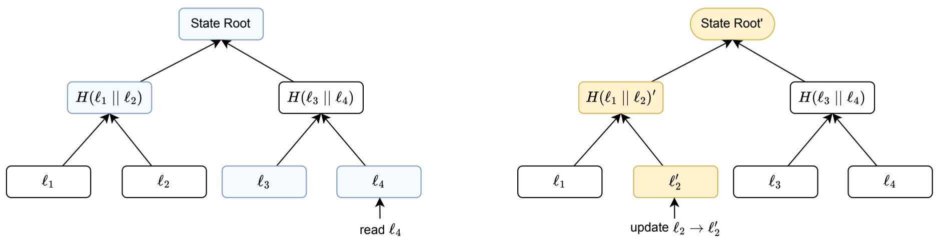
State Root
$H(\ell_1 \mid\mid \ell_2)$
$H(\ell_3 \mid\mid \ell_4)$
ℓ_1
ℓ_2
ℓ_3
ℓ_4
read ℓ_4
State Root'
$H(\ell_1 \mid\mid \ell_2)'$
$H(\ell_3 \mid\mid \ell_4)$
ℓ_1
ℓ_2'
ℓ_3
ℓ_4
update $\ell_2 \to \ell_2'$

Figure 1: Read proof for the leaf ℓ_4 (left) and update of the leaf ℓ_2 to a new value ℓ_4' (right).

In the left figure, the objective is to read the information of the fourth leaf (ℓ_4). Suppose we are conducting a transaction, and ℓ_4 represents the balance of the account initiating the transaction. In the process of reading this leaf, it is crucial to verify its inclusion in the current state at the specified position (which will be deterministically computed from some unique properties of the data being stored, such as the account address or the type of data). The Storage State Machine comes into play by generating an execution trace, and the boxes highlighted in blue must be supplied to it. These blue boxes enable the Storage State Machine to validate that the root it computes aligns with the State Root that we have.

In the right figure, we examine a scenario where a leaf is updated, resulting in a change in its value and, consequently, a modification of the State Root. Specifically, all the yellow boxes in the figure are affected as they incorporate information from the updated leaf ℓ_2'. The Storage State Machine is once again employed to generate an execution trace, ensuring the accuracy of all tree modifications. By following the same procedure, the consistency of State Roots is verified to confirm that the changes have been accurately executed.

All the essential hashes required for computing execution traces are stored in the **HashDB**. During the execution of the L2 State's related zkEVM opcodes (like for example `:SSTORE` or `:SLOAD`), the Storage State Machine's Executor consistently retrieves values from the HashDB and use them in order to construct the corresponding execution traces. The HashDB contains all the hashes (i.e., all the nodes) of the Merkle tree of the L2 state. In Figure 2 we can observe how the Executor process works.

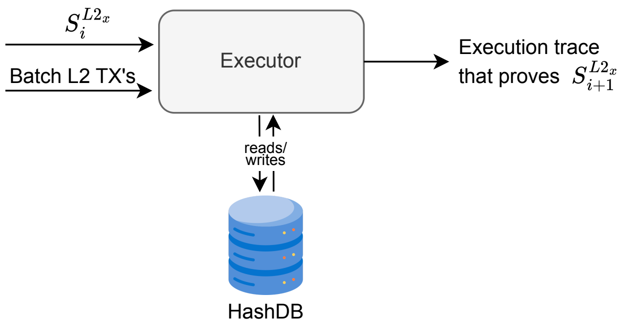

Figure 2: Executor Proving the current State by Interacting with the HashDB to retrieve the corresponding Merkle Proofs to fill the execution trace.

It's important noting that the Merkle tree utilized in the Polygon zkEVM is a binary Sparse Merkle Tree (SMT), distinguishing itself from the Merkle tree in the L1 EVM, which is a completely different hexary structure.