LeanIMT: An optimized IMT

Privacy & Scaling Explorations

June 5, 2024

1 Abstract

Contents

2 Introduction

2.1 Motivation

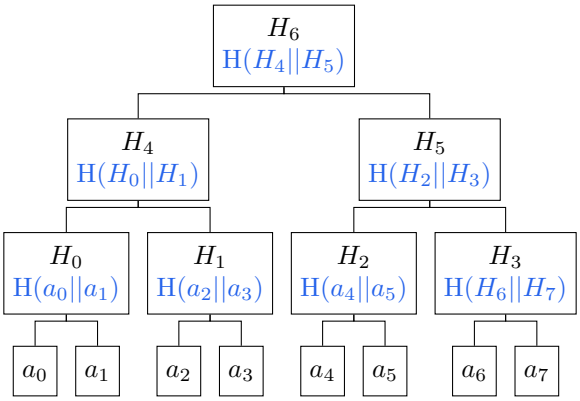
3 Merkle Tree

3.1 Incremental Merkle Tree

An Incrememental Merkle Tree (IMT) is a Merkle Tree (MT) designed to be updated efficiently.

3.2 Binary Tree

A Binary Tree is a tree data structure in which each node has at most two children, referred to as the left child and the right child.
TODO: Explain what is a Merkle tree and an Incremental Merkle Tree.



4 LeanIMT

4.1 Definition

The **LeanIMT** (Lean Incremental Merkle Tree) is a Binary IMT.

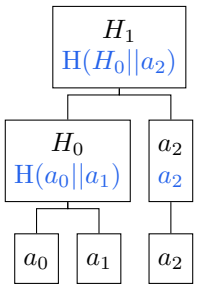
The LeanIMT has two properties:
1. Every node with two children is the hash of its two child nodes.
2. Every node with one child has the same value as its child node.

Example of a LeanIMT

$T = (V, E)$

$V = \{a_0, a_1, a_2, H_0, H_1, H_2\}$

$E = \{(a_0, H_0), (a_1, H_0), (a_2, a_2), (H_0, H_1), (a_2, a_2)\}$



4.2 Insertion

There are two cases:
1. When the new node is a left node.
2. When the new node is a right node.

We will always see one of these cases in each level when we are inserting a node. It is like, when you insert a node, if that node is left node, the parent node which is in the next level, will be the same node. If it is a right node the parent node, will be the hash of this node with the node in its left. This algorithm will be the same in each level, not only in level 0.

Case 1: The new node is a left node

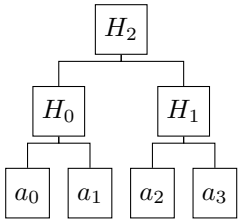
It will not be hashed, it's value will be sent to the next level.

If we add a_4.

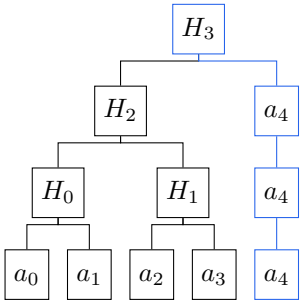
$T = (V, E)$

$V = \{a_0, a_1, a_2, a_3, H_0, H_1, H_2\}$

$E = \{(a_0, H_0), (a_1, H_0), (a_2, H_1), (a_3, H_1), (H_0, H_2), (H_1, H_2)\}$



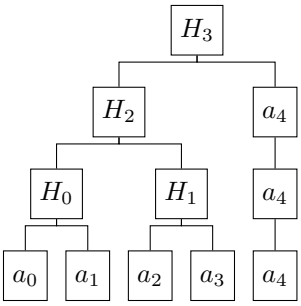
Before inserting a_4



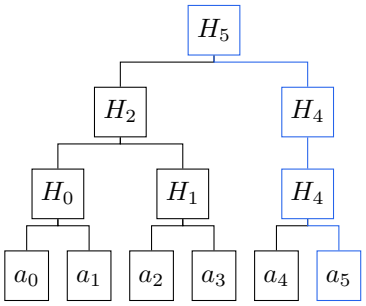
After inserting a_4

Case 2: The new node is a right node

If we add a_5.



Before inserting a_5



After inserting a_5

4.2.1 Pseudocode

Algorithm 1 LeanIMT Insert algorithm

```
1: procedure INSERT(leaf)
2:     require leaf is defined
3:     if depth < newdepth then  ▷ If the tree depth after inserting a node is greater, add a new tree level
4:         add a new empty array to nodes
5:     end if
6:     node ← leaf
7:     index ← size  ▷ The index of the new leaf equals the number of leaves in the tree.
8:     for level from 0 to depth - 1 do
9:         nodes[level][index] ← node
10:        if index is odd then  ▷ It's a right node
11:            sibling ← nodes[level][index - 1]
12:            node ← hash(sibling, node)
13:        end if
14:        index ← ⌊index/2⌋  ▷ Divides a number by 2 and discards the remainder.
15:    end for
16:    nodes[depth] ← [node]  ▷ Store the new root at the top level
17: end procedure
```

5 Benchmarks

6 Conslusions

This document is based on the work of [1].

References

[1] Barry Whitehat Kobi Gurkan Koh Wei Jie. "Semaphore: Zero-Knowledge Signaling on Ethereum". In: (2020). URL: https://semaphore.pse.dev/whitepaper-v1.pdf.