Consider a Merkle tree of depth n

where we want to proof m

leaf values. The naive solution requires m \times n

hashes and m \times n

values. We can do better:

Motivating example: Consider two sibbling leafs. In the first step, we hash them together. In subsequent steps we hash this value back to the root. We need n

hashes and n-1

values instead of 2n

.

In general, all leafs will eventually merge. Where they merge we have left and right branch already available and don't need a value. From that point on, we only need a single Merkle path to the root saving values and hashes corresponding to the depth.

The actual number of hashes required depends on where the queried indices are. In general the closer together they are, the better.

Reference implementation of optimal Merkle de-commitments in Python and Solidity is here: https://gist.github.com/Recmo/0dbbaa26c051bea517cd3a8f1de3560a

Credits for the idea and algorithm go to Starkware!