Problem

State trees are an important element in a blockchain that supports fraud proofs or stateless clients, because they are needed to prove that a) a fraudulent transaction is attempting to spend an input that does not exist (in a UTXO-based blockchain), b) a fraudulent transaction is performing a transaction based on state that does not exist (in an account-based blockchain) or c) in the case of a stateless client, prove that some state exists.

However, a malicious miner could create a blockchain with a fraudulent state tree root, that does not match the state transitions made by the transactions in the block, and so no one would be able to find prove the inclusion of any state in the tree and find proofs of fraudulent state, because they don't have the data for it. You could prevent this by having data availability proofs for the entire state tree, but it would of course not be efficient to require nodes to broadcast the entire state of the system every block.

Solution abstract

A solution to this is to have many "intermediate" state tree roots, and corresponding merkle proofs for all the intermediate state tree roots that show modifying the state tree in a certain way should lead to the next intermediate state tree root. If any of these intermediate roots are wrong, then fraud proof is limited to that specific intermediate root.

Solution design

Each block header contains a standard sparse merkle tree with 2^{256}

leafs (every possible SHA256 hash), such that a leaf at index k

represents the state of the key k

(if you don't understand how having 2^{256}

leafs is even possible without burning the universe, see the linked paper). Sparse merkle trees are used because they do not have to be rebalanced, so if you have tree A and tree B, where tree B is tree A but with one leaf that is different, then proving that the root of tree A becomes the root of tree B when you change a specific leaf is always O(log(n))

. I will call this a "merkle change proof": which is a merkle proof that the root of tree A, combined with some change in leaf index k

in tree A, makes the root of tree B.

Now, instead of publishing just one sparse merkle tree root in each block header, a miner should publish multiple intermediate tree roots in the same header, such that, for example, if you apply the results of transactions 1-10 to the previous block's sparse merkle tree root, you get intermediate root R_1

, then if you apply transactions 11-20 to R_1

, you get intermediate root R_2

, until you apply all the transactions to each previous root and get final root R_n

.

Because there is an implicit ordering to the transactions in a block, a full node can now easily generate efficient fraud proofs for invalid state roots, because they only have to prove that one intermediate root is invalid for the whole block to be invalid. If a full node applies transactions 1-10 to the previous block's sparse merkle tree root, but does not get the same intermediate root R_1

, then they can publish a fraud proof, which will consist of a merkle change proof that applying transactions 1-10 to the previous block's sparse merkle tree root does not result in the published root R_1

. The same can be repeated for R_1

and R_2

, R_2

and R_3

, and so on.

The maximum size of the fraud proof is therefore equivalent to the number of transactions added by each intermediate state root (in this case, 10). However, light clients also need to be sure that the intermediate roots are actually available for full nodes to generate fraud proofs for, so these intermediate roots should be included in the same block of data that is covered by any data availability scheme such as erasure coding. This means there is a tradeoff between the size of the fraud proof, and the size of the data (the intermediate roots) that light clients need to ensure availability for.