zk-SNARK proof of the validity of Plasma transactions.

Konstantin Panarin*

E-mail: kopanarin@yandex.ru,kp@bankexfoundation.org

We propose certain zk-Snark construction to assert the validity of transactions in plasma chains. As the crux of the matter, plasma state S is represented by a Merkle tree, where leaf nodes contain balances and paths correspond to addresses. All associated addresses are read top-down starting from the most significant bits. Note, that for now we assume that leaf nodes contain balances and do not contain anything else. This is done only for simplicity as presence of hashed valued inside leaves doesn't change the matter of things.

Assume that at some point in time plasma state S is considered to be valid by the majority of the nodes in the network, and we are going to show that a trusted transition from state S to state S' via transaction (addr_from, addr_to, amount) can be made by plasma operator with the use of zk-SNARKS. The main idea is that this zk-SNARK proof should contain as much information as the ordinary Merkle-based proof. We let addr_from, addr_to, amount and merkle_root_before_trx and merkle_root_after_trx to be the only public inputs for our zk-SNARK and all further mentioned inputs are considered private.

The gadget for the zk-SNARK takes the following steps:

1) Compute the common prefix for addr_from and addr_to. It is a bit mask which contains 1's in the positions where merkle tree paths addr_1 and addr_2 coincide and 0's in other positions: for example if tree height is 8 (so bit-length of addr_1 addr_2 is also 8) than common_prefix mask for addresses 0b01101010 and 0b01101100 will be 0b11111000.

2) Compute the "special number" sp_index. It is a number which has precisely one 1 in the position of the first 0 in the bit representation common_prefix. In the previous example special number is equal to 0b00000100. The meaning of this number will be made clear later.

3) Let balance_from be private input contained in leaf corresponding to addr_from address. Given (again as a private input) merkle-proof $[m_1, m_2, ...m_n]$ to balance_from construct a gadget checking that computed root hash (via given merkle-proof) is equal to merkle_root_before_trx.

4) The same gadget as in step 3 made for balance_to, addr_to and merkle-proof $[p_1, p_2, ..., p_n]$.

5) We should show that parts of merkle proofs $[m_1, ..., m_n]$ and $[p_1, p_2, ..., p_n]$ corresponding to common_prefix mask are in fact coincide. The aim of this additional check is to provide stronger resistance against hash collision attacks (although necessity of this check is arguable). This check can be accomplished with the following series of equations: common_prefix[i] $\cdot m_i =$ common_prefix[i] $\cdot n_i$, where common_prefix[i] means the extraction of i-th bit of common prefix and i runs from 0 to tree height.

6) Note, that in the construction of gadgets in step 2 and 3 we get vectors of intermediate hashes: hash1 = hash(leaf), hash2 = hash(hash1 || m1) or hash2 = hash (m1 || hash1) depending on the address bit. The fact is that when we construct such series of hashes for balance_from one of them (and exactly one of them) will be included in the merkle proof $[p_1, .., p_n]$ for balance_to. This is a place where "special number" comes into play, because the index i for which hash_i = p_i is equal to the position of the unique 1 in bit representation of sp_num. We come to another series of checks: sp_num[i] $\cdot$ hash_i = sp_num[i] $\cdot p_i$.

7) The same thing is in 5 constructed for intermediate hashes for balance_to and vector $[m_1, ..., m_n]$.

8) Check possibility of spendability, that is check that balance_from $\geq$ amount;

9) Update merkle tree changing two leaf nodes: balance_from = balance_from - amount, balance_to = balance_to + amount. Construct new merkle proofs $[m'_1, m'_2, ..., m'_n]$ and $[p'_1, p'_2, ..., p'_n]$. Notice that both new proofs differ correspondingly from $[m_1, ..., m_n]$ and

$[p_1, ..., p_n]$ exactly in one position, and this position is again equal to the position of the unique 1 in bit representation of sp_num.

10) repeat steps (3)-(7) for updated balances, proofs and merkle root hash.

That's it. Using this scheme as a starting point we have a possibility to prove that the transition from state S to state S' via applying all transactions in a block is correct. This seems to be a nice problem to apply recursive zk-SNARKS: we construct separate zk-SNARK for each transaction in a block and than we prove that the proofs for every pair of adjacent transactions is correct and proceeding this way we get a "Merkle" tree of proofs.

We are now going to approximately estimate the number of R1C (rank 1 constraints) in the preprossessing step of constructed zk-SNARK. The setup is the following: let tree height be H, the number of R1C constraints generated by one run of inner hash-function is R and bit-size of all balances and hash digests is bounded by p. Then common prefix can be calculated in H bitwise operations, special number can be derived directly from common prefix in another set of H bitwise operations. Every merkle proof results in $H * R$ and we have four of them. The comparisons in steps (5), (6), (7) and (9) require H constraints each. The check of spendability requires p constraints. It is now obvious that overall difficulty of zk-SNARK is dominated by the difficulty of used hash function: there will be $4 \cdot H \cdot R$ constraints, and one of the solutions to the problem of minimizing the size of R1CS system is to use SNARK-friendly hash-function. Our choice is novel hash MiMC and we use MimC 2n/n mode. In the case of Plasma we use MiMC with 256-bit output, so we take n = 1025. The number of inner MimC rounds will be equal to 1292 and each round results in 2 R1C constraints.