(WARNING: This design seems to be broken, seehere)

(Previous knowledge of Plasma Cashflow, Merkle Sum Trees, and RSA accumulators is assumed)

Current thoughts on RSA accumulators are all trying to assign prime numbers to non-fungible coins as in Plasma Cash, and use some creative approaches to batch a large number of coins in log(coins) sized proofs@benj0702 came up with the great idea of assigning primes to transactions, by adding a prime

property to our transactions, which is chosen by the clients. This post tries to further explain this idea. Read the original post here.

A regular Plasma Cash transaction is like this:

struct Transaction { uint256 coinId; address recipient; uint256 prevBlock; }

Plasma Cashflow transactions (With RSA accumulators) are probably like this:

struct Transaction { uint256 coinStart; uint256 coinEnd; address recipient; uint256[] prevBlocks; uint256 prime; }

Notes:

- Instead of a single coinId

, we have a range of coins, specified with coinStart

and coinEnd

. (Read Plasma Cashflow spec)

- We may want to merge contiguous ranges of coins from multiple transactions, so we should provide an array of prevBlock

s.

- We also assign a prime number to our transactions.

The rule is: If you want to spend some range of coins, you should say in which block(s) did you get those coins (prevBlocks

) and the accumulator of the current block should include the corresponding primes of the parent transactions.

E.g. if I assign prime 7

to my transaction (which is submitted in block 18) and I get a batched exclusion proof of prime 7

in blocks [A_{18}...A_{100}]

, I can be completely sure that there isn't any valid

transaction in blocks (19, 100)

that could spend part/all of my coins.

You might ask what happens if someone tried to spend a transaction that is also labeled with prime 7

in block 46? (In other words, what should we do if 7

is accumulated in [A_{45}...A_{46}]

?)

Nothing bad happens! It is very important to understand that the actual security comes from our Merkle-Sum-Trees

, so here is what the prover should do to convince me that my coins are not spent in blocks (19, 100)

, he gives me:

- RSA exclusion proof of prime 7

in blocks [A_{18}...A_{45}]

- Merkle-Sum-Tree exclusion proof of my coins in block 46
- RSA exclusion proof of prime 7

in blocks [A_{46}...A_{100}]

Now the question is, how can we synchronize the clients with each other, and prevent them from using each other's prime for their transactions? (So that their coin history size is reduced as much as possible) Through the operator! The operator may have some kind of prime pool

, that contains all of the available primes in it. Clients can ask the operator for free primes through some API call, and the operator doesn't accept transactions with primes that are not free. (Even if he does, nothing bad happens, as previously explained)

Whenever an UTXO is completely spent, its prime can get back into the prime pool and be used again by other transactions.

Example

Say Alice sent Bob coins (500,1500) through the following tx which is submitted in block 14:

coinStart: 500 coinEnd: 1500 recipient: Bob prevBlocks: [...] prime: 19(Submitted in block 14)

Now Bob owns coins (500,1500), he wants to send coins (500,1000) to Charlie and coins (1000,1500) to David. He simply creates two transactions:

coinStart: 500 coinEnd: 1000 recipient: Charlie prevBlocks: [14] prime: 23(Submitted in block 18)

(Accumulator of block 18 should include prime 19 as the parent transaction is labeled with prime 19, or the transaction is invalid)

coinStart: 1000 coinEnd: 1500 recipient: David prevBlocks: [14] prime: 29(Submitted in block 20)

(Accumulator of block 20 should include prime 19 as the parent transaction is labeled with prime 19, or the transaction is invalid)

Now let's say Charlie was the owner of coins (0, 500) through some transaction in block 3 with prime 7, and he wants to send coins (250,1000) to Edward

coinStart: 250 coinEnd: 1000 recipient: Edward prevBlocks: [18,3] prime: 31

(Accumulator of block 20 should include prime numbers 7 and 23 as the parent transactions are labeled with primes 7 and 23, or the transaction is invalid)