Special thanks to Justin Drake for discussion

One of the challenges for all Plasma flavors, Plasma Cash too, is the amount of history data that users need to keep around, and send to recipients if they want to send coins. For example, suppose a Plasma chain has one block committed to chain per minute, with \approx 2^{16}

transactions per minute (~= 1000 transactions per second). We assume each user has their coins split across \approx 10

fragments. To have the full information needed to win an exit game, each fragment requires a Merkle branch of length \approx 16

per block, so that's 16 hashes * 32 bytes * 500000 minutes * 10 fragments \approx 2.5

GB for one year. In a transfer or atomic swap or similar operation, this is data that must be transferred.

We can increase the efficiency here with RSA accumulators. An RSA accumulator is a data structure providing similar functionality to a Merkle tree: a large amount of data can be compressed into a single fixed-size "root", such that given the data and this root, one can construct a "witness" to prove that any specific value from the data is part of the data represented by that root. This is done as follows. We choose a large value N

whose factorization is not fully known, and a generator g

(eg. g = 3

) which represents the empty accumulator. To add a value v

to an accumulator A

, we set the new accumulator A' = A^v

. To prove that a value v

is part of an accumulator A

, we can use a proof of knowledge of exponent scheme (see below), proving that we know a cofactor x

such that (g^v)^x = A

(note that x

may be very large; it is linear in the number of elements, hence why we need a proof scheme rather than providing x

directly).

For example, suppose that we want to add the values 3,5,11

into the accumulator. Then, A = g^{165}

. To prove that 3

is part of A

(we'll use the notation "part of [g...A]

" from now on for reasons that should become clear soon), we use a proof of knowledge scheme to prove that (g^3)^x = A

for some known cofactor x

. In this case, x = 55

, but in cases involving many thousands of values x

could get very large, which is why the proof of knowledge schemes are necessary.

One example of a proof of knowledge scheme,described by Wesolowski (see also this talk by Benedikt Bünz) is as follows. Let h = g^v

and z = h^x

. Select B = hash(h, z) \mod N

. Compute b = h^{floor(\frac{x}{B})}

, and r = x \mod B

. A proof is simply (b, z)

, which we can verify by checking b^B * h^r = z

(proof of completeness: b^B * h^r = h^{B * floor(\frac{x}{B}) + x \mod B} = h^x

). Note that this proof is constant-sized.

To prove that a value is v

not

part of [g...A]

, we need only prove that we know r

such that 0 < r < v

where A * g^r

is

a known power of g^v

using the algorithm above. For example, suppose we want to prove that 7

is not part of [g...A]

in the example above. 7

is not a factor of 165

, so there is no integer x

such that (g^7)^x = g^{165} = A

. But g^7

is

a known root of A * g^3 = g^{168}

, so we can provide the values r=3

(satisfying the desired 0 < r < 7

) and cofactor x = 24

, and we see (g^7)^{24} = A * g^3 = g^{168}

.

Now, here is how we can use this technique. We require Plasma Cash roots to come with both a Merkle tree of transactions and

an RSA accumulator of coin indices that are modified in that block. Using a transaction in an exit game requires both the Merkle proof and the RSA accumulator proof of membership. This means that a RSA proof of non-membership is sufficient to satisfy a user that there is no data in that block that could be used as a transaction in an exit game. However, the RSA accumulator that we use is cumulative

; that is, in the first block, the generator that we use can be 3, but in every subsequent block, the generator used is the accumulator output of the previous block. This allows us to batch proofs of non-membership: to prove that a given coin index was not touched in blocks n .... n+k

, we make a proof of non-membership based on the post-state of the accumulator after block n+k

using the pre-state of block n

as the generator. If a user receives this proof of non-membership, they can be convinced that no proof of membership can be generated for any

of the blocks in the range n ... n+k

.

This means that the history proof size for a coin goes down from one Merkle branch per Plasma block to two RSA accumulator proofs per transaction of that coin

: one proof of membership for when the transaction takes place, and one proof of non-membership for the range where it does not. If each coin gets transacted on average once per day, and an RSA proof of non-membership is ~1 kB, then this means \approx 1

Kb * 365 days * 10 fragments \approx 3.6

MB for one year.

If there are many coins, then we can optimize further. Proofs of non-membership can be batched: for example, if you prove that A * g^{50}

is a power of g^{143}

, then you know that log_g(A) = -50 \mod 143

, which implies it is not a multiple of 11 or

1. This works well up to a few hundred indices, but if we want to have very fine denominations it may be possible to batch much more, see Log(coins)-sized proofs of inclusion and exclusion for RSA accumulators