One of the challenges of a Plasma implementation that wishes to use an RSA accumulator is maintaining the proof update procedure for each UTXO or coin ID per block commit. Here we consider the application of RSA accumulators to Plasma Cash with indivisible coin IDs.

Note: the goal of this construct is to introduce a stateless deletion and addition protocol to Plasma Prime. This prevents the client from doing any work aside from verification, and that the operator complexity of generating the accumulator output per block is limited to that of the product of all added primes per block (no need to recalculate old block outputs, effectively forgetting any previous block's state). The time to compute the accumulator output is verifiably slow (with the help of forthcoming ASICs) and the time to compute the Wesolowksi or Pietrzak NI-POKE varies depending on implementation choice

.

For every coin ID we will accumulate unique prime numbers by performing aVDF as outlined in RSA accumulators and Plasma Cash History Reduction. The complexity of generating the new root of the accumulator is equal to that of the product of the number of added primes per block group operations. Let u = p_1 * p_2 ... p_n

. Where the amount of work in the Wesolowski scheme is equal to T

squarings, our work is equal to u

group multiplications of g \in G

. Two types of block commits are presented in this protocol, one for adding an element and one for deleting. We maintain a database that stores the inclusion proof for every ID on the operator or proof service node with an O(N)*

~3000 bits of space for N

coin IDs and show that an operator can perform the update procedure for all coins in time equal to that of processing the VDF output of the new transactions added primes given N

parallel processors.

1. Adding

To add an element to the accumulator we raise the previous accumulator root proof \pi_r

and each individual coin proof \pi

to the new element or product of elements. This will be done for every coin ID and can be done in parallel.

1. Deleting

Deleting an element requires using Shamir's trick to calculate Bezout coefficients and aggregate inclusion proofs. Notice that when deleting an element, the new inclusion proof for every other coin ID is the aggregation of the removed element's inclusion proof and the other coin's inclusion proof. After applying Shamir's trick to every coin proof we are done and this can be done in parallel.

Example:

Define function \pi_{1,2} \leftarrow ShamirTrick(\pi_1, \pi_2)

to be a function that aggregates two inclusions proofs into one.

Given an RSA accumulator, let block 0 equal A_0 = g^{3}

containing 1 coin ID represented by prime number 3. Our operator's account database will look like…

d3: \pi = g

Next we will add 5 to the accumulator so that block 1 equals A_1 = g^{3*5}

. An operator calculates the new accumulator root by taking A_1 = {A_0}^5

. The database will now be…

d3': \pi = d3^5

d5: \pi = A_0

Block 2 will add elements 7, 11, and 13

let u=7*11*13=1001

(amount of work our vdf needs to compute)

A_2 = {A_1}^u

with database entries…

d3'': \pi = d3'^u

d5': \pi = d5^u

d7: \pi = A_1

Now, we remove element 3, the operator will set A_3 = d3''

and adjust the database with…

d5'': \pi = ShamirTrick(d3'', d5')

d7': \pi = ShamirTrick(d3'', d7)

Potential Issues:

with 2^{40}

coin IDs the operator or service node will need approximately 2^{40}*3000

bits or about 400 terabytes to store all inclusion proofs vdf outputs alone.

Given that adding elements to the accumulator is a VDF, there is a bottleneck on the number of transactions that can be processed per block. In an account model (if it is found possible in Plasma), where the accumulator is a vector and position binds accounts to vector components, it may be possible to process a larger amount of value per block commit. With others seeking to implement ASICs for this VDF, it may also be found feasible to process a large number of prime numbers.

Update

@keyvank noticed that subsets of proofs can be generated from a superset of aggregated coin proofs. Noticing that an aggregated coin proof is the exclusion of all of owned coins, to obtain a subset proof, you can reintroduce the remaining excluded primes not in the subset. This will allow the operator to store a constant sized aggregate proof for every account. When a user of the plasma chain needs to send a subset of their coins they can request that the operator adjust their proof and create one for the subset. This decreases the number of parallel executions for each block to one for each account instead of one for each ID, but it does require the operator to do computation when a user needs a subset proof for a later transaction. The client can also generate this assuming their coin ID ownership isn't too large.To illustrate…

let

A = g^{3*5*7*11}

let

User_0 = {11} ... \pi_0 = g^{3*5*7}

let

User_1 = {3,5,7} ... \pi_1 = g^{11}

get User_1

single coin proofs from aggregate \pi

\pi_3 = \pi_1^{5*7}

\pi_5 = \pi_1^{3*7}

\pi_7 = \pi_1^{3*5}