Plasma Primeflow

Nathan Ginnever, Keyvan M. Kambakhsh

December 12, 2018

Introduction

This work presents an application of cryptographic universal accumulators [5] to Plasma cash history size reduction. Plasma Primeflow is an extension to the Cashflow specification [2] that maintains an O(N) merkle paths for inclusion proofs where N is a total number of *ranges* but, reduces the burden of exclusion proofs from O(N) per block commit to O(1) for all owned ranges over any interval of blocks. An RSA accumulator is utilized to achieve this, requiring a unique prime number for each range ID. It should be noted that this implementation requires a trusted setup ceremony to ensure that the order of the RSA field is unknown. There is a possibility of using lattice based accumulators [1] to add arbitrary data to the accumulator set [todo explore this further]. Arbitrary data in the accumulator is desired as to reduce the inclusion proofs from O(N) merkle paths to O(1) batched inclusion accumulator value.

As with previous Plasma designs, this paper will detail the *deposit*, *send*, and *exit* mechanisms of the NFT transaction system. This expands on the cashflow document by adding the necessary details for the RSA accumulation.

Transaction Format

Transactions will be associated with ranges and require the following format.

$[[send, [prime, sequence, start, offset, recipient], signature]$

Transasctions must be placed within their range $[start, offset]$. This placement belongs to a merkle sum tree which allows for the implementation to maintain transactions that involve multiple token IDs within the same range to be placed at the same leaf in the tree.

Ranges

We define ranges of ownership the same way as the cashflow spec [2]. This uses a merkle sum tree to reduce the inclusion proof size from $O(log(n))$ where $n = 2^{40}$ (a reasonable set of primes) to $n = |fragments|$ (using cashflow spec definition of fragments). For a set F of fragmented ranges and n coins, an owner must keep $log(F) * n$ hashes. As fragmentation increases the number of inclusion hashes increases to $log(2^{40}) = 40$ in the worst case.

TODO: Further definition on how ranges split and what happens to prime assignment when they do. Same for when two ranges become contiguous, can we combine primes?... When a range is assigned, it will show an offset that counts the number of NFT tokens in that range. Tokens can never be split or merged, just reassigned to new range ownership.

Hash-To-Prime (Ranges)

We can use a similar approach to what Bankex [3] proposes. The primes accumulated to mark range identifiers do not need to be large for any security reasons. Given this we should choose primes for this function to be as small as possible. We should still be able to reserve the first 2^{40} primes for coin IDs by selecting primes of size less than 10^{14}.

Accumulator Exclusion Proofs

The RSA accumulator will be used to mark exclusions rather than inclusions. In doing this, we can use the inclusion mechanics of RSA while still proving exclusion proof. This is necessary as there is no know way to batch exclusion proofs with RSA. Recall, for some tx p_i that spends a token, to prove exclusion we must find a value r s.t. $0 < r < p_i$ where $A * g^r$ is a known power of g^{p_i}.

Accumulator Mechanics

Given a group $G = Z/Z_p^*$ where accumulator $A = g^N mod(n)$ and $N = \prod p_i^{e_i}$. Given that the factorization of n is unknown, it is believed to be infeasible to fake proofs by the prover.

To achieve a list of exclusion, we note that multiple inclusions of the same prime number is possible. We will use this fact to construct a sequence number for each range. When a prime is accumulated more than once, we can use the number of times e_i that p_i is accumulated in N. If we want to know that the token range associated with p_i has never been spent since its deposit, we would ask for an inclusion proof of $p_i^{e_i}$. Then the prover will compute $w = N/p_i^{e_i}$. If a spend of p_i occurs and you want to know that p_i is at sequence s, you only need to see if p_i^{s+1} is in N.

An inclusion proof π is of the form $\pi = g^w$ for some witness w. The witness is the cofactor $N/p_i^{e_k}$. A verifier computes $g^{w^{p_i^{e_i}}} = A$. i.e To know that $p = 5$ has been spent once, set $e = 2$ and $w = N/5^2$. The verifier would compute $\pi^{5^2} = g^N$.

Wesoloski Proof of Exponentiation

The cofact witness $N/p_i^{e_k}$ can get very large as all transactions are accumulated in N. Using the Wesoloski scheme, the prover (operator) can supply a compact w as the proof to the verifier. This ensures that the operator uses the correct generator. [todo: perhaps clarify this]

Hash to Prime (Wesoloski)

We need large primes for the Wesoloski scheme as Alex [4] research suggests in his criteria for Hash to Prime. As we reserved prime numbers smaller than 10^14 in size for the accumulator, we can not reuse them to compute the proof of exponentiation. Selecting primes larger than 10^14 will be a requirement for the $H_{p-wesoloski}$ TODO expand on this.

Deposit

This is done in the same way as cashflow with an onchain token deposit to the plasma parent contract. The only difference here will be the assignment of a unique prime number that will identify a new range of tokens.

Send

The send format is described above and allows for an owner of a range of coins to be able to generate a transaction that assigns all or part of their range to a new owner.

Exit

Since we have no merge protocol, we must exit all fragmented ranges separately. Atomic swaps may help to defragment ranges. [todo expand]

Smart Contract

[TODO]

References

1 https://eprint.iacr.org/2014/1015.pdf
2 https://hackmd.io/DgzmJIRjSzCYvl4lUjZXNQTransaction-Tree-Structure
3 https://ethresear.ch/t/plasma-prime-design-proposal/4222
4 https://hackmd.io/s/SyynHeGCQ
5 https://www.cs.purdue.edu/homes/ninghui/papers/accumulator$_{a}cns07.pdf$