In another post I looked into Merkle Mountain Ranges (MMRs) to offload history objects away from the state trie in the context of the stateless client paradigm. To summarise some of the really neat features MMRs have for maintaining witnesses of history objects:

1. Low frequency updates

—witnesses are updated log(#{updates after insertion}), as opposed to once per insertion for tries

1. Extend-only updates

—the witnesses (Merkle paths) only get extended, as opposed to having internal nodes be modified from unrelated object updates in the trie (this is great for parallelism)

1. Marginal memory overhead

—witness maintenance requires only log(#{objects}) overhead, as opposed to #{objects} overhead for tries

1. Shorter average-case witnesses

—size log(#{updates after insertion}), as opposed to size log(#{objects}) for tries

1. Shorter worst-case witnesses

—size log(#{updates after insertion}), as opposed to size #{objects} for tries

Despite the witness-friendly properties of MMRs for history objects, Vitalik pointed out what seems to be their main limitation (emphasis mine):

History, state, and asynchronous accumulators in the stateless model

The idea of having explicit data structures in the system to make low-witness-update-frequency history objects easily usable is definitely an interesting one. That said, there are limits: in general, any application that allows you to reference objects from the history is very often also going to require some kind of stateful mechanism

for efficiently proving whether or not those objects have already been consumed.

In the language of accumulators MMRs are not dynamic, hence the need for a dynamic accumulator (in our case, the Patricia-Merkle trie) to handle state in the general case. It then becomes natural to ask ourselves to what extent can contracts be state-minimised, so as to maximally limit the use of the trie and maximise the value of the MMR. There are two ways to minimise use of the trie:

1. Minimise amount of data stored in the trie
2. Minimise number of updates to the trie

One generic approach I suggested in the original post is to use SNARKs/STARKs to reduce the amount of data that needs to be put trie to just 32 bytes (the size of a hash). The problem with this approach is that we still have one trie update per transaction, so the number of trie updates is linear in the number of transactions. It turns out there is also a generic way to achieve a sublinear number of trie updates in this hybrid MMR-trie setup!

A cryptoeconomic accumulator

Below I describe a “cryptoeconomic accumulator” that is both witness-friendly and dynamic. We start with a witness-friendly MMR and, using a collateral scheme powered by the trie, bootstrap it into a dynamic accumulator.

Given an object o

we model accumulator additions and deletions by wrapping an add

or del

bit of metadata to o

(i.e. [add, o]

and [del, o]

). Such wrapped objects can then be appended to the MMR, with the rule that only users who have previously posted collateral can do so. The trie keeps track of who posted the collateral to enforce that rule.

A dynamic accumulator emerges if every [del, o]

event is paired with a previous [add, o] event, and we never have two [del, o]

events for the same

o

. (Two such repeated deletion events form a “violation”.) Violations can be prevented cryptoeconomically with the rule that if a user attempts to make a violation then any whistleblower can efficiently prove the violation within a period of time, thereby burning half the collateral and sending the other half to the whitleblower.

Notice additions and deletions to the cryptoeconomic accumulator take time to “confirm”, and there needs to be a scheme (e.g. checkpoints) to allow for rollback in case a violation attempt actually occurs. In exchange for a one-time collateral setup, a user can make an arbitrary number of deletions to the cryptoeconomic accumulator without touching the trie, thereby achieving the desired sublinearity.

Conclusion

It seems that MMRs just got even more awesome in the context of stateless clients. Their witness-friendly properties that are readily available for history objects can be extended to also handle state, modulo a sublinear number of trie updates.