We're looking for a commitment scheme to commit to a list of N values (think N ~= 2**28

) which has the following properties:

1. The commitment should be small (fixed size or polylog)
2. The commitment should be (computationally) binding, ie. a commitment c

constructed from one vector V = [v1 ... vN]

should not match against any other feasibly-discoverable vectors. (We don't care about hiding properties)

1. For any given set of positions x1 ... xk

where 1 <= xi <= N

, there should be an efficient (ie. quasilinear time to calculate, sublinear proof size

) way to prove that the values V[x1]

, V[x2]

… V[xn]

are part of the vector committed to by c

1. There should be an efficient (ie. ideally O(k)

but O(k * log^c(n)

is okay too) way to compute such a proof for any x1 ... xk

. Requiring O(n)

or even slightly larger precomputation before you receive the coordinates is okay.

1. Given a set of updates (x1, y1) ... (xk, yk)

to a vector there should be: * (i) an efficient (ie. ideally O(k)

, but O(k * log^c(n))

is okay too) way to update c

- (ii) an efficient (ie. ideally O(k)

, but O(k * log^c(n))

is okay too) way to update any precomputed tables required to generate proofs (that's updating the entire precomputed table needed to generate all witnesses

, not updating a single witness)

1. (i) an efficient (ie. ideally O(k)

, but O(k * log^c(n))

is okay too) way to update c

1. (ii) an efficient (ie. ideally O(k)

, but O(k * log^c(n))

is okay too) way to update any precomputed tables required to generate proofs (that's updating the entire precomputed table needed to generate all witnesses

, not updating a single witness)

Note that we have constructions that almost

satisfy these goals:

- Merkle trees

: satisfy everything but

the crucial requirement (3) for a k-element proof to be sublinear in k

- Kate commitments

: satisfy everything but

(5. ii) efficient witness updating (witness updating is O(n * min(k, log(n)))

because the value of each witness depends on every element)

- SNARK proofs over Merkle trees using MIMC/Pedersen

: satisfy everything but

generating a SNARK to compress many Merkle branches is ~1-2 orders of magnitude too expensive

- RSA accumulators

: no efficient witness updating (5. ii)

The goal is to have a ready construction that can be used for state storage constructions, eg.Multi-layer hashmaps for state storage

Construction based on not-yet-existent moon-math cryptography

As a proof-of-concept to show that a construction could conceivably exist, consider the case where we had high-degree graded encodings, ie. a primitive even stronger than multilinear maps, where (i) given x

you can compute encode(x)

, and (ii) given encode(x)

and encode(y)

you can compute encode(x*y)

and (iii) you can check encodings against each other for equivalence.

Let h

be a hash function that outputs fairly long values (sufficiently long that given n

outputs with very high probability no output will be a factor of the product of all n-1

other outputs). To commit to V = [v1, v2 ... vn]

, compute commitment = encode(h(2**256 + v1)) * encode(h(2**256 * 2 + v2)) * ... * encode(h(2**256 * n + vn))

.

To prove that set of key/value pairs S = {(i1, v[i1]) ... (ik, v[ik])}

is inside the commitment, use the product of all encode(h(2**256 * i + v[i]))

values not

in S

as a witness; the verifier would recompute the encodings of the key/value pairs in S

, multiply them by the witness, and check that they get the same value as the original commitment.

Note that if the prover precomputes and stores a tree, containing the encodings for the subsets {v1}, {v2} ... {vn}, {v1,v2}, {v3,v4} ... {v[n-1], vn}, {v1..v4} ... {v[n-3] ... vn} ... {v1...vn}

, then any proof for k

elements can be constructed in k*log(n)

time by multiplying together the appropriate sister nodes in the tree, and any single update to the vector would only require updating log(n) elements in this tree.

One possible path to finding a solution is taking this tree-structure-based approach, but to get to a construction feasible today one would replace multiplication with some other operation, where multiple “sister nodes” in the tree can somehow be aggregated.