TL;DR:

An improved hashmap construction to store state that does not depend on a combination with Merkle trees to resolve collisions, and therefore gets rid of DOS attacks described in the previous construction. The construction is efficient as long as the number of stored elements is not much larger than the degree of the polynomials used for storage.

Background

Please refer to

- Hashmap-based polynomial commitments for state for an introduction on Kate commitments and multi-reveals
- Using polynomial commitments to replace state roots for more motivation, and also how to open several polynomials using just one opening

Construction

As in my previous construction (Using polynomial commitments to replace state roots), we will use polynomials in the Lagrange basis to store values (which will be hashes of key-value pairs) at point evaluations of the polynomial. In a polynomial $p(x) = \sum_{i=0}^{n-1} a_i x^i$

of degree n

, we can store n

different values, for example at the positions $0, 1, \ldots, n-1$

. In practice, this will be on a subgroup rather than integers for efficiency, but we don't need to worry about this for the description of this construction.

Here is how we will use polynomial commitments to store a key-value map: Let $p_i(X) \in \mathbb{F}[X], i=0, 1, 2, \ldots$

be an infinite sequence of polynomials of degree n

. We say that the key-value pair (k, v)

is stored at x

in polynomial p_i

if $p_i(x) = \mathrm{hash}(k,v)$

. We say nothing is stored at x

in p_i

if $p_i(x)=0$

. If all $p_i(x)=0$

for $x\in\{0, \ldots, n-1\}$

then p_i

is "empty" and its commitment value doesn't have to be stored.

We start with all p_i

empty. Iteratively, for each key-value pair (k,v)

to be stored, we will

- Check $x=\mathrm{hash}(k, 0)\% n$

in p_0

- Check $x=\mathrm{hash}(k, 1)\% n$

in p_1

- Check x=\mathrm{hash}(k, 2)\% n

in p_2

- …

until we either encounter k

has been stored or an empty position in p_i

. We then store \mathrm{hash}(k,v)

at p_i(x)

for x=\mathrm{hash}(k, i)\% n

.

Performance

- To prove membership/an existing key, only a single value has to be decommitted (as by construction, the same key cannot be added twice). The total witness size is only the data plus one group element for the proof, for all

data in aggregate (amortized it is just the data that has to be provided, as the group element is negligible)

- To prove non-membership (and adding a new key), a number of values will have to be provided. Below I will show that the number amount of data that has to be provided to prove non-membership is about m/n

, where m

is the total number of keys stored. * As a note, by separating keys and values into two separate polynomial sequences, only the keys have to be provided and not the values for proofs of non-membership.

- As a note, by separating keys and values into two separate polynomial sequences, only the keys have to be provided and not the values for proofs of non-membership.

Number of non-empty polynomials

To estimate how many polynomial commitments we have to store in this construction, we can compute the expected number of “overflows” if we want to store m

elements in a layer of size n

(i.e., in a polynomial of degree n

).

A position (“bucket”) in a polynomial overflows if more than one key-value pair would be stored at it. In other words, we can get the expected number of overflows per bucket by computing for each position x

, how many elements in excess of 1 (since it can hold 0 or 1 elements) it is expected to store, which is

- The expected number of elements stored at x

minus one

- Plus the probability that 0

elements will be stored at x

; i.e.

E_x(m) = \frac{m}{n} - 1 + \left(1-\frac{1}{n}\right) ^ m

The expected total number of overflows E

can be computed via linearity of expectation:

E = \sum_{x=0}^{n-1} E_x(m) = m + n \left(\left(1-\frac{1}{n}\right) ^ m - 1\right) \approx m + n (\mathrm{e}^{-\frac{m}{n}} - 1)

This also lets us easily compute the number of elements that are stored at this layer, i.e. do not overflow

$F = E - m \approx n (\mathrm{e}^{-\frac{m}{n}} - 1)$

Lets look at two possible regimes: If $m \gg n$

$E \approx m + n (\mathrm{e}^{-\frac{m}{n}} - 1) \approx m - n$

If $m \ll n$

:

$E \approx m + n (\mathrm{e}^{-\frac{m}{n}} - 1) \approx m + n\left(1 - \frac{m}{n} + \frac{1}{2} \left(\frac{m}{n}\right)^2 \mp \ldots - 1\right) \approx \frac{1}{2}\frac{m^2}{n}$

Estimating the last layer as the one at which an expected 0.5

elements are stored, the second to last layer has $\frac{1}{2}\frac{m_{-1}^2}{n} = \frac{1}{2} \Rightarrow m_{-1}= \sqrt{n}$

elements stored (Sanity check: This is exacty what we would expect from the birthday paradox, so we're on the right track). Layer last but two would have $\frac{1}{2}\frac{m_{-2}^2}{n} = m_{-1} = \sqrt{n} \rightarrow m_{-2} = \sqrt{2n\sqrt{n}} = \sqrt{2} n^{\frac{3}{4}}$

. We can iterate this until we get to a "relatively full" layer, at which point th $m \gg n$

regime takes over.

From this we see that what will happen is that

- We will have m/n

almost full layers

- Then we will have a number δ

of layers which will get gradually more empty

The total number of layers is thus $m/n+\delta$

. δ

depends weakly on n

, and my numerical estimate is $\delta(n=2^{24})\approx 4$

and $\delta(n=2^{32}) \approx 5$

.

Since the layers at the end get empty very quickly, the number of reveals necessary to prove non-membership does not contain δ

in the average case, and thus only ca. $m/n+1$

reveals are necessary to prove non-membership.

DOS attack

We will consider the following DOS attack:

- How many extra layers ϵ

can an attacker add to the expected $m/n+\delta$

layers (effectively forcing the storage of $m/n+\delta+\epsilon$

layers by grinding)

Note that this also puts a limit on any attack to an individual key: An attacker cannot add more than $\delta+\epsilon$

elements to the (expected) non-membership proof of length m/n

. The real limit is smaller than this.

Attack

Let’s assume that the store is currently empty. This is roughly equivalent to having m/n+\delta

layers (partially) filled and trying to fill up the layers after these layers, so we should arrive at the same result modulo a small constant (It is very easy to create a collision in almost full layers).

I assume that the best strategy for the attacker will probably be to fill every layer up to level \sqrt{n}

[^1]. Assuming layers 0, \ldots, {\ell-1}

are each filled with \sqrt{n}

elements, inserting a new element into p_\ell

means trying to create a key that collides in each of these layers, needing \sqrt n^\ell

computational power. Most of the computation will be required to insert the last element.

At n=2^{24} \Leftrightarrow \sqrt{n}=2^{12}

will become infeasible at around \ell=7

. This means a total number of \delta=8

layers could be added by an attacker at a computational cost of 2^{84}

and the economic cost of inserting 7\cdot 2^{12} \approx 28{,}000

keys.

At n=2^{30}

this becomes \delta=7

at a computation cost of 2^{90}

and the economic cost of inserting 6 \cdot 2^{15} \approx 200{,}000

keys.

[^1]: It would be interesting to try and improve this attack; the attacker can maybe do slightly better by filling up lower layers a bit more and higher layers a bit less. Attacks should be compared on the basis of inserting the same number of keys, as we assume that inserting order of n

keys would be prohibitively expensive.