From: Perlner, Ray A. (Fed)
To: (b) (6)
Subject: RE: hash
Date: Wednesday, September 22, 2021 12:58:00 PM

I think it makes the most sense to use the average cost to access a bit of memory. Each bit of memory should be accessed almost exactly the same number of times.

A somewhat unrelated thought: It occurs to me that using Block Wiedemann to get rid of the log factor in memory access times is not actually optimal, assuming the planar memory model. Using block Wiedemann means you have to store more memory by a factor of the block size. So, e.g. if we use a block size of 25, it should increase memory access costs (for data sent including addresses) by a factor of 5 assuming planar memory. I think this means that in Rainbow Response's implicit computation model, it's better to use the complicated hashing strategy we were originally considering to group memory accesses from different equations into blocks. This also probably means that the use of Block Wiedemann can't fully account for the discrepancy between the Rainbow Response prediction for the cost of the equation solving in https://eprint.iacr.org/2016/412.pdf and the experimental data.

From: Daniel Smith (b) (6)
Sent: Wednesday, September 22, 2021 12:20 PM
To: Perlner, Ray A. (Fed) <ray.perlner@nist.gov>
Subject: Re: hash

Yeah, that make more sense. It might be worth mentioning that even with the worst case of the hash, (1,1,...,1), that the chunk of memory is a factor of something like 6600 smaller. So even if we measure by the worst case bit access, there is still a free savings of 6 bits on the complexity. That already derails the claims of the Rainbow team.

On Wed, Sep 22, 2021 at 10:39 AM Perlner, Ray A. (Fed) <ray.perlner@nist.gov> wrote:

> Hi Daniel,
>
> I agree that the average hash value is associated with 1 part in 2^{25} of the total memory, but this represents a mix of larger and smaller chunks of memory. The average memory location (which I think is what we actually care about) will be in one of the larger chunks.
>
> I think a good way for accounting for this is to consider the average cost (i.e. the square root of the chunk size) for a given memory location.
>
> Relative to the cost for true RAM access, this should be
>
> \sum_{i=3}^{25}Binomial(25,i)(Binomial(62,65-i))^{3/2} / (Binomial(87,65))^{3/2} = 0.00132 . See wolfram alpha link for the calculation here: https://www.wolframalpha.com/input/?i=Sum+from+i%3D3+to+25+of+%28%2825+choose+i%29*%2862+choose+%2865-i%29%29%5E%281.5%29%29%2F%2887+choose+65%29%5E%283%2F2%29

This is much closer to what you'd expect for memory chunks of size 2^20 as opposed to memory chunks of size 2^25.

Cheers,
Ray

From: Daniel Smith (b) (6)
Sent: Wednesday, September 22, 2021 7:26 AM
To: Perlner, Ray A. (Fed) <ray.perlner@nist.gov>
Subject: hash

Hi, Ray,

I'm going over the memory size associated to each hash to determine its average size. I wanted to get a bit more detail on this so that it is precise. I'm getting something different than your note, though, so I want to run it by you so you can see if I'm overlooking something.

Recall the numbers. We have m'=87, and o2=64. So we have Binomial(87,65) different minors variable patterns in the equations. So the calculation should be to determine the average number of minors variables patterns represented by each 25-bit hash. There are a few impossible hashes (in the sense that they correspond to 0 minors variable patterns). These are any hash with 0, 1 or 2 1s.

So from i=3 to i=25 counting the number of 1s in the hash, we add up the number of minors patterns including i of the first 25 columns and 65-i or the last 62 columns. This is \sum_{i=3}^{25}Binomial(25,i)Binomial(62,65-i).
We can then find the average number of patterns associated with a hash by dividing by 2^{25}-326.

Now, I think that it is clear that the sum above is exactly Binomial(87,65), and we're dividing by something that is very close to 2^25, so the answer should be Binomial(87,65)-2^25, so that each hash has on average a 2^{-25} share of the total memory. So I'm not getting a 2^{-20} fraction of the total memory.

What do you think? Is something wrong with this?

Cheers,
Daniel