What was the reason behind Ruby choosing SipHash for Hash?

Hello,

I am digging into the history behind Ruby using SipHash for its Hash. I found that in 2012 there were CVE-2012-5371 showing up; the Ruby maintainers went with the decision to switch algorithms, probably, because we wanted something quick to implement at the time.

The change went live in late 2012.

Fast forward with the Ruby 3x3 initiative, we now seem to care about the performance again. And hash DoSing does not seem to be an urgent threat now; we have time to be deliberate about Hash again.

I can't find the old discussion related to Ruby's SipHash decision. I just found that SipHash is not the only solution to prevent hashtable DoSing. There is an interesting discussion on golang side in late 2015: [https://github.com/golang/go/issues/9365](https://github.com/golang/go/issues/9365)

Just to recap, Go's authors argue that:

- Cryptographic hash is not needed to construct a DoS-resistant hashtable.
- If the random seed is per-hashable bases, the attack vector exploitable from a remote adversary seems unlikely.
- If we want to be extra careful about it, and since the collision is unlikely, when collision actually does occur despite the per-hashable seed, we can handle that as a special case where we can rerandom the seed and rehash the key.
- The way random seeds are folded into the hash does matter, for example, CityHash does f(g(msg), seed); in such case, collision in g will cause a collision in f because the output of g is independent of the seed.
- Slowing down hashtable for everyone to prevent hard-to-exploit DoS doesn't seem to be a good trade-off.

On the actual implementation, they use AES-NI to achieve good pseudo-random functions' properties. And use some fallback non-cryptographic hashing function on the platform without AES-NI.

Now, I read the rationale on golang side, I want to understand the rationale on the Ruby side too.

I am not there 10-years-ago, and I can't find records or discussions at the time. There might be some Ruby limitations that the approach described by go's authors does not apply.

So, I asked in the hope of someone still remembering what was happening, the situation we are in 10 years ago, or the limitation of Ruby that prevents per-Hash seeds.

---

**History**

**#1 - 02/16/2022 04:12 PM - midnight (Sarun R)**

I misread a point related to extra carefulness. They don't suggest a rehash on collision, but on hash map grow; since on hash map grow we need to rehash the key anyway, they suggested that in addition to the rehashing we add an extra step for reseeding, the runtime cost is almost zero. This way, the DoSer can't grow the hash with lots of collisions passing a certain threshold where the hashmap grows.

**#2 - 02/16/2022 05:16 PM - Eregon (Benoit Daloze)**

One concern which might have been considered is what if someone writes their own hash table in pure-Ruby using #hash? Then the only way to protect those is a better #hash method by default, which seems to be what Ruby did. OTOH, I think almost nobody defines their own hash table in pure-Ruby (would be a lot of work and much slower).

I agree a cheaper hash function for Ruby would be welcome for performance. Using SipHash/murmur/etc for e.g. Integers feels needlessly expensive.

There is also the approach to simply use a process-wide global seed for randomizing hashes, and that's still used in Ruby IIRC. But apparently that's not enough without also a better hash function?

So then is the per-Hash-instance seed good enough, which is what you suggest?
FWIW Java took yet another route, where they did not change hash functions, but when seeing many collisions for HashMap/ConcurrentHashMap they switch internally to a tree-like data structure and compare using compareTo() instead of just hashing.

since on hash map grow we need to rehash the key anyway

BTW this is not true in Ruby, where the hash code of the key is saved in the internal hash entry, and should not be recomputed needlessly for semantics compatibility (and efficiency).
In other words, the hash of a key is only computed once per Hash, when added to a Hash, in Ruby, unless Hash#rehash is used.

#3 - 02/17/2022 12:02 AM - shyouhei (Shyouhei Urabe)
JFYI this thread can be something related: https://bugs.ruby-lang.org/issues/13017