You probably *do* need a cryptographic hash function

Ruxmon Melbourne, August 2015
Michael Samuel

http://www.ruxmon.com/melbourne/

Background: The basics

Variable length input → fixed length output

Definitions:

- **Collision**: when multiple inputs map one output
- **Preimage**: finding an input to match an output

Uses:

- **Probabilistic mapping:** hash tables, load sharing
- **Integrity:** checksums, MACs, signatures

Background: Hash vs MAC

A *Message Authentication Code* uses a secret key to provide security.

MACs still require *collision* and *preimage* security, but they're only required to provide it

- *for* those who possess the key
- *against* those who don't possess the key

In this presentation I use Hash and MAC somewhat interchangeably.

In most cases it's clear that you need one or the other, but occasionally it's borderline.
In general if both will work a MAC is safer.

Problem Statement

Insecure hash functions can be crippled or disabled by data.

“A good hash function should satisfy two requirements:

1) Its computation should be very fast.
2) **It should minimize collisions**

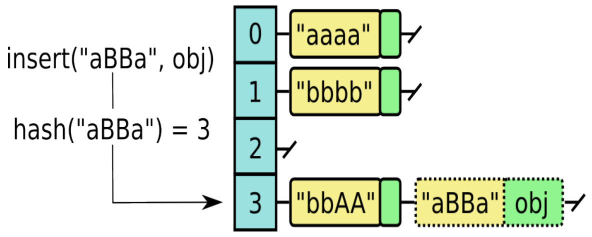
Property (a) is machine-dependent, and **property (b) is data-dependent**.”

-- Knuth, The Art Of Computer Programming Vol 3 (1997, so I forgive him)

Hash Tables

If an attacker can force all items into a single bucket, you now just have a linked list - O(n) lookups instead of O(1).

Inserts are lookup-then-insert - so you could force n*(n-1)/2 compares.



Language	Hash	Since
Python	SipHash	3.4
Ruby	SipHash	1.9.3p327
libstdc++	FNV/Murmur	
v8 (js)	Jenkins	
Go	It Depends	

Collisions force worst case behaviour - a DoS attack!

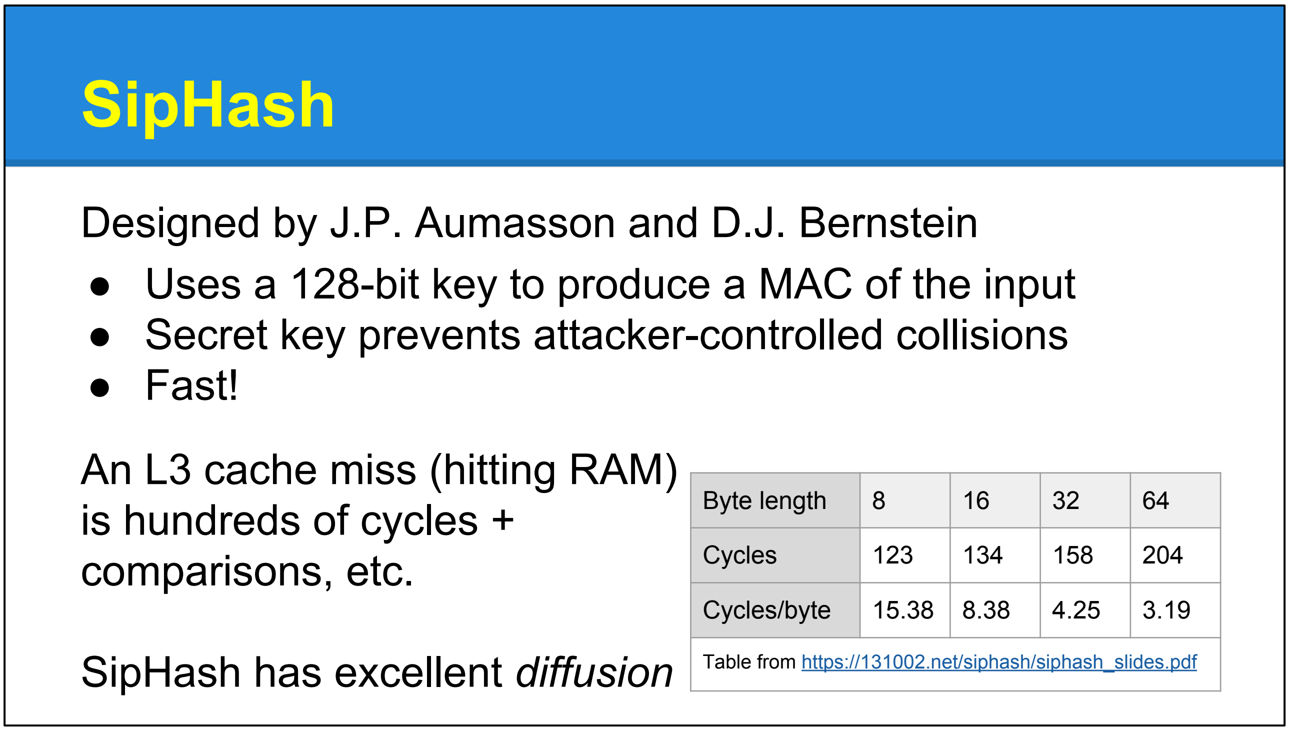
Until recently (see next slide), it was rare that anyone even considered that maybe the hash function for a hash table needed to be secure. After all, if somebody were to DoS a hash table they'd only be DoSing themselves.

But this isn't a safe assumption - worker threads and non-blocking I/O loops shared by many users is common now - if you can force the CPU to be busy you can stop the process from serving other users.

Random notes:

- libstdc++ uses FNV if sizeof(size_t) == 4, Murmur if sizeof(size_t) == 8
- Go uses either one of a couple of weak hashes, or a very-reduced-round AES based hash - depending on machine type
- Go's hashes are seeded with CPU ticks, which is dubiously secret

Image from http://commons.wikimedia.org/wiki/File:Hashtable_linkedlist_collision.png



SipHash

Designed by J.P. Aumasson and D.J. Bernstein

- Uses a 128-bit key to produce a MAC of the input
- Secret key prevents attacker-controlled collisions
- Fast!

An L3 cache miss (hitting RAM) is hundreds of cycles + comparisons, etc.

SipHash has excellent *diffusion*

Byte length	8	16	32	64
Cycles	123	134	158	204
Cycles/byte	15.38	8.38	4.25	3.19

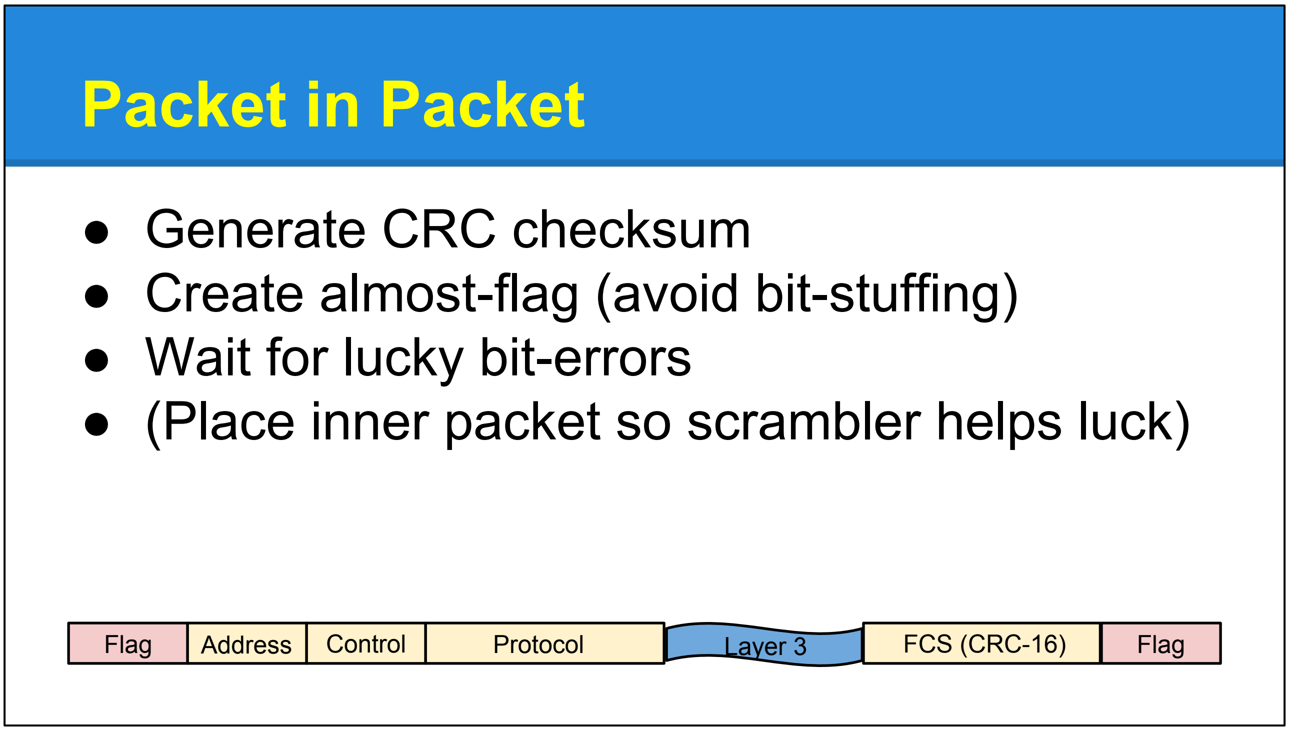
Table from https://131002.net/siphash/siphash_slides.pdf

See https://131002.net/siphash for papers, code, etc.

There are two variants of SipHash - SipHash-2-4 is 6-round and SipHash-4-8 is 12-round.

For situations where there's no compatibility issues (such as in-memory hash tables) you can choose SipHash-2-4 which is appears secure and it'd be trivial to change with a software update.

You could use SipHash-4-8 for when you're committed to compatibility and CPU time of the hash is irrelevant anyway. The only gain is security margin (not level) - insurance against cryptanalysis.



Packet in Packet

- Generate CRC checksum
- Create almost-flag (avoid bit-stuffing)
- Wait for lucky bit-errors
- (Place inner packet so scrambler helps luck)

Flag
Address
Control
Protocol
Layer 3
FCS (CRC-16)
Flag

Travis Goodspeed first introduced Packet in Packet to me at Ruxcon in 2012 - see https://www.youtube.com/watch?v=iQk0GHXs8NY or http://travisgoodspeed.blogspot.com.au/2011/09/remotely-exploiting-phy-layer.html

You'd have to be pretty lucky to come across a high-BER POS link behind a firewall these days, but in general both radio and wire protocols should be protected by a MAC or AEAD.

Rsync algorithm

Signature phase (on receiver):

- Divide destination file into (700 byte) blocks
- Calculate rolling sum and strong sum on each block

Delta phase (on sender):

- Create hash table of rolling sum → {strong sum,file offset}
- Go through file byte-by-byte looking for {rolling sum,strong sum} matches, generating either COPY or DATA commands

Delta phase (on receiver):

- COPY blocks - copy data from original file into new file
- DATA blocks - put raw data into new file

The rolling sum used by the rsync utility and librsync is addler32, but modulo 65536.

This function was chosen because it has the property that you can add and remove single characters from the checksum without a full recalculation, which is used to roll-in and out bytes in the delta phase.

The “safe” alternative would be to calculate the strong sum at each byte offset - quite expensive CPU-wise (but would be more bandwidth-efficient).

Rsync - collisions

https://github.com/therealmik/rsync-collision

Generated from two md5 collisions.
First collision is just any collision, second reverses the effect on the rolling sum.
Can be generated in seconds.

Chosen-prefix collisions possible too*Conditions Apply - escalate privs?

Still not fixed. Be careful what you rsync.

Originally rsync used md4, but switch to md5 later due to security concerns. The rsync maintainers haven't updated to a newer hash function yet, and don't seem to be keen on doing so.

There's a checksum seed argument, which for the md4 protocols prepends a seed, turning md4 into somewhat of a MAC. But for the md5 protocol it appends the seed, which is not effective due to the way MD5 works internally. (the same applies for many other hash functions, including SHA-1 and the SHA-2 family)

In any case, the seed is only 32-bit and either chosen using transient but not secure methods, or passed on the command line.

There's a whole-file MD5 too, so chosen-prefix collisions may need to be in first block of the file unless you can find a multi-IV collision somehow - I'd love to hear about it if you know how!

Librsync

https://github.com/therealmik/librsync-collision

Same as rsync, but used **md4** truncated to **64 bits**.

Rather than attacking md4, I decided write a generic *birthday attack* (as a warning to others).

1.0.0 replaced md4 with **blake2b**, 256 bits output

(Good response from maintainer).

Still vulnerable to Hash DoS (limited by rolling sum)

librsync is not the same codebase or maintainer as the rsync utility. It is used by Dropbox, as well as the duplicity backup program.

Sorry folks, you can't get Dropbox t-shirts anymore - they only pay cash for bugs now

Try this at home: MS-RDC

http://research.microsoft.com/pubs/64692/tr-2006-157.pdf

- Similar to rsync
- Used by DFS
- Microsoft acknowledges security issues:
    - https://msdn.microsoft.com/en-us/library/dd304647.aspx
    - http://msdn.microsoft.com/en-us/library/dd340879.aspx
    - Suggests using secure hash on whole file (and then what??)

So the difference between a Ruxmon and Ruxcon talk is that for Ruxcon I'd have finished the research a week before presenting :)

It'd be interesting to see how DFS handles collisions - would it refuse to sync or would it transfer without delta-compression?

Try this at home: memcacheDoS

Current memcached uses either Jenkins hash or Murmur hash (runtime config).

This could possibly be combined with logic bugs in some webapps that don’t expect items to disappear as soon as you add them.

The type of logic bug I expect to find is where a programmer assumes that an newer item wouldn’t expire before an older item, or simply that an item that was just inserted will still be there.

This requires the potential for attackers to control the value of *keys* stored in memcached, which pub-discussion suggests is rare.

Try this at home: unfair-queueing

Certain ISP routers implement fair-queuing as a bunch of FIFO queues, and service each queue round-robin.

A hash of the source and destination addresses is used to select which queue a packet goes in.

Adjust Bittorrent peer selection - all in one queue so it doesn't slow interactive traffic. Why pay extra for QoS?

On Cisco routers this is just:

```
interface x
  fair-queue
```

From what I've heard the hash used is Cisco-proprietary, so it may be EULA-secure ;)

Try this at home: disk deduplication

It’s unlikely that anyone set out using a non-cryptographic hash function for disk deduplication.

There are *unconfirmed* rumours that major vendors are using MD4 and MD5 still. That would be interesting.

Try this at home: Rowhammer ECC

While the original paper authors never claimed that ECC RAM protects against Rowhammer, some vendors do.

ECC's single bit-error correction property implies that the same value can be expressed more than one way.

Understanding this property may be the key to rowhammer exploitation on systems with ECC RAM.

Be careful what hardware you do this on - some blade chassis remember the RAM chip is bad and insist on getting an engineer out to replace it (perhaps calling them out automatically!).

Conclusion

Replace your insecure hash function with

$\mathbf{H}(\mathbf{x}) \rightarrow \mathbf{x} \cdot 0$

Don’t like the result? Then switch to a cryptographically secure hash.

Thanks!

Michael Samuel

https://www.miknet.net/
Twitter: @mik235
GitHub: therealmik
IRL: The Oxford Scholar