hash functions and you

or, why breaking SHA-1 is a thing

Google figured out how to generate two PDF files with the same SHA-1 hash

why does this matter?

hashes rule everything around me

What's a hash function?

Any function that can be used to map data of arbitrary size to data of fixed size.

-- the wikipedia entry on hash functions

data → function → number

data → function → number

message → hash → digest/hash/code

the hash value is (usually) a lot smaller than the data!

hash tables!

Suppose you have a lot of data that you want to stick into memory for fast access by a *key*. You use a hash function to map the keys into *buckets*, one for every output number, and put the matching data for a key into the key's bucket. There might be more than one thing in the bucket, but that's fine because you can look inside the bucket to find your item in a small collection, instead of having to search the whole thing.

This is the data structure underlying associative arrays, caches, and a million other things in software.

Data integrity!

Alice publishes a package on npm. Bob wants to know if the package he's downloading is the data Alice published, because he's worried about data corruption or tampering.

Alice creates a *checksum* of the data:

```
> md5 <  package.tgz`
0030be42121988078dca0ec982d04f72
```

She gives the output to Bob, and he can run it on his download to see if he has the data she meant to give him.

- output is *deterministic*
- output is *uniformly distributed*, not clustered
- usually: fast
- often: output value has a fixed size
- sometimes: similar inputs produce nearby outputs
- sometimes: similar inputs produce distant outputs (avalanche effect)

avalanche effect

a small change in input

large change in output

locality-sensitive hashes
or similarity hashes
do exactly the opposite

copyright violation detection on youtube will have similarity hashing behind it

mostly we want the avalanche effect
sometimes called the butterfly effect

often we want cryptographic hashes

a good cryptographic hash

- deterministic
- fast to compute
- has the avalanche effect
- reconstructing the original message from the hash is infeasible
- finding collisions is infeasible

collisions mean we can't tell that

two inputs are different

deliberate collisions would be an attack

Back to the classic example!

Carol wishes to trick Bob into running her npm package instead of the one Alice wrote. Because Alice used the weak MD5 algorithm to sign her data, Carol is able to craft a tarball that has the same MD5 digest but different data inside. She man-in-the-middle attacks Bob and serves him her cleverly-crafted package.tgz instead of Alice's.

Bob is now pwned.

collision-resistance is crucial for verifying data integrity

This is why "breaking" SHA-1 matters.

Collision-resistance might not always matter

non-cryptographic hashes

- usually a lot faster than cryptographic hashes
- finding collisions might be feasible
- ditto reconstructing the original

non-cryptographic hashes

- use when speed matters
- use when defense against malicious input doesn't matter

uses for non-cryptographic hashes

- bloom filters 🌻
- lookup tables
- sharding data uniformly

non-cryptographic hashes

- MurmurHash
- CityHash
- HighwayHash
- xxHash
- seahash

you can design your own

non-crypto hash with a little math

crytographic hashes are harder

cryptographic hashes

are the workhorses of security

use a cryptographic hash

- to verify message integrity
- to verify passwords without knowing them
- to identify data

choose a cryptographic hash

to defend against

malicious input or attack

hashes suitable for passwords have some unusual properties

hashing passwords

- salt + password ➜ hashing function ➜ output
- store the output *only*
- repeat the transformation when checking the password to see if you get the same result

An attacker who can run that transformation frequently & quickly is one who can "brute-force attack" your users' passwords.

password hashes are tunably expensive

slow to run, use a lot of memory

to slow down attackers

password hash algorithms

- bcrypt
- scrypt
- argon2
- PBKDF2

some cryptographic hashes you should know!

- md5
- the SHA family
- blake2
- siphash

MD5

- designed in 1991
- blown apart by 1996
- really fast
- **do not use** as a cryptographic hash
- can use for data bucketing if you trust the input

the SHA family

Secure Hashing Algorithm

NIST standards

standardization means SHA algos are widely available & widely used

SHA-1

- 160-bit result (40 hex digits)
- very widely used (git shasums!)
- designed by NSA in 1995, 1st attack in 2005
- better attack in 2015
- collision by Google in 2017
- **do not use** as a cryptographic hash

npm uses SHA-1

for data integrity checks for tarballs

Back to that classic example!

Alice is using SHA-1 to sign her package tarballs. Carol is an employee of Google or maybe of a nation-state's spy agency, has a lot of computing power available, and really wants to pwn Bob. She cleverly crafts a tarball that has the same shasum as Alice's, and serves it to Bob.

In ten years, Carol will be able to do this with a Raspberry Pi 7 the size of her thumbnail instead of a fleet of cloud computers.

SHA-2

- comes in 224, 256, 384 or 512 bit variants
- SHA-256 & SHA-512 are the most-used
- designed by NSA in 2001
- no feasible attacks known
- **do use** freely
- replace SHA-1 with this, generally

SHA-3

- comes in 224, 256, 384 or 512 bit variants
- won a competition to choose next SHA standard
- adopted as standard in 2015
- no feasible attacks known
- **do use** freely

blake2

- a SHA-3 finalist
- 32 bit word variant (produces 256-bit results)
- 64 bit word variant (produces 512-bit results)
- faster than SHA-3 selection
- **do use** freely

siphash

- optimized for speed with small input
- used in hash table implementations in many languages
- hash table implementations can have security implications!

which one should npm adopt?

- SHA-2, SHA-3, and BLAKE2 are all fine choices
- SHA-2 is safer because of implementation availability
- we are not size-sensitive about the output
- we care more about picking an algo that will last
- probably end up with SHA-512

summary

- stop using MD5
- stop using SHA-1
- do use SHA-2 and newer hashes
- use bcrypt to store passwords
- don't invent unless you're an expert
- in which case you're full of quibbles about this talk

Questions?

More-of-a-comment-reallys?

all the links

Hash function wiki page ▪ bcrypt scrypt ▪ PBKDF2 ▪ argon2 ▪ MD5 ▪ SHA-1 ▪ SHA-2 ▪ SHA-3 ▪ BLAKE2 ▪ seahash ▪ xxhash ▪ siphash ▪ designing your own non-cryptographic hash function