(Applied) Cryptography

Week #4: Hash Functions and Keyed Hashing

Manuel Barbosa, mbb@fc.up.pt

MSI/MCC/MERSI – 2022/2023

DCC-FCUP

Part #1: Hash Functions

What is a Hash Function?

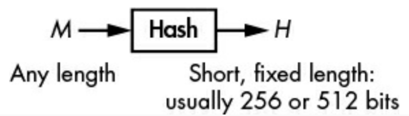
Hash functions are everywhere:

- key derivation
- digest for authentication
- randomness extraction
- password protection
- proofs of work

Not only in crypto:

- indexing in version management repositories
- deduplication in cloud storage systems
- file integrity in intrusion detection

What are they?



What is a Hash Function? (2)

The hash output is short: hash, fingerprint, digest.

Cryptographic hash functions give strong security guarantees.

Most intuitive property is "**use hash as identifier**":

- cryptographic hash functions cannot be injective (why?)
- yet they should be somehow *well distributed* or *unpredictable*
- we assume hash value identifies arbitrarily large input

For example: signing a $H(M)$ is as secure as signing M.

Hash functions need to be deterministic and public:

- everyone should be able to recompute hash/identifier
- so what does security mean?

What is a secure Hash?

We look for efficient algorithms that seem to have nice properties:

- unpredictable outputs
- hard to find pre-images
- hard to find collisions

Hash functions are validated heuristically:

- similarly to process for AES
- international competition to select designs
- competitors are scrutinized wrt security and performance
- several rounds so more eyes on small number of proposals
- most recent one was for SHA-3

(First) Preimage resistance

Preimage resistance:

- Let $\mathcal{S}$ the set of preimages (domain)
- Let $\mathcal{R}$ the set of images (range)
- Attacker given a value $Y \in \mathcal{R}$
- Adversary guesses $X \in \mathcal{S}$ and wins if $H(X) = Y$

How should Y be chosen?

Some ways make the problem trivial, e.g.:

- $Y = H(X')$ where $X' \in S$ and $S \subset \mathcal{S}$ is small
- Why?

One-wayness requires X' chosen at random from large S. (large?)

Practical hash functions are candidate one-way functions.

Collision Resistance (CR)

Hard to find any $X \neq X'$ such that $\mathrm{H}(X') = \mathrm{H}(X)$.

Suppose we have the best possible hash function:

- What is the probability that two hash values collide?
- Outputs are random so: $1/2^n$ where n is the output length
- We will find a collision if we check roughly 2^n pairs

Is CR easier or harder to achieve than preimage resistance?

How can we break hash functions generically?

Attack that always finds a preimage?

- Search through all possible preimages (aka brute-force)
- Expected cost if hash function is perfect and outputs n bits?
- 2^n operations. Why?

Attack that always finds a second-preimage?

- Nothing better than previous attack

Attack that always finds a collision?

- Exponentially easier!

Finding collisions

Any compressing/non-injective function has collisions.

They can be found with work $2^{\frac{n}{2}}$ using birthday attack:

- Compute values like brute-force attack but ...
- Store them in a fast data structure *indexed by image value*
- Each new image value is searched in data structure
- Repeat until collision is found

How many operations?

- After n values we have checked $n * (n-1)/2$ pairs (why?)
- To check 2^n pairs we need roughly $\sqrt{2^n} = 2^{\frac{n}{2}}$ values
- Overall complexity essentially that of finding a preimage for hash with n/2 bits!

Somewhat counterintuitive: **birthday paradox**.

The implication of birthday attacks

When CR is required hash outputs are 2x security parameter:

- 128-bit security => 256-bit hashes
- 256-bit security => 512-bit hashes

We can use security-parameter-sized hash outputs when:

- We don't require security against arbitrary collisions
- For example, when we only require pre-image resistance
- For example, when we are deriving a key from a secret input

Part #2: Building Hash Functions

Building hash functions

There are two approaches that both use iterative processes:

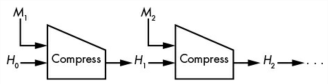
- Merkle-Damgård construction: Used for MD4, MD5, SHA-1, SHA-256, SHA-512 relies on a $m+n$-to-n bits compression function to construct a hash function of output length n for arbitrary input lengths.
- Sponge construction: Used for SHA-3, uses a l-bit permutation to construct a hash function for arbitrary input and output lengths.

Merkle-Damgård construction

All prominent hash functions from the 80s to the 2000s.

H_0 is the initial value or IV: constant and public.

Message M is broken into blocks of size m, $M_1, M_2 \ldots$



In SHA-256 block size is 512 bits and the output size is 256 bits.

In SHA-512 block size is 1024 bits and the output size is 512 bits.

Padding is **always** added to the message before breaking into blocks:

- append a 1 bit
- fill with zeros up to 64/128 bits away from next block end
- last 64/128 bits encode the message length in bits

Security of Merkle-Damgård construction

Magical result:

- Compression function is CR (for small inputs)
- Implies whole construction is CR (for arbitrary inputs)

I.e., to break the hash function you must break the compression function.

This means $2n$-to-n CR hashing solves all our problems!

Is that true?

Problems with MD: Length Extension

Suppose you compute $h = H(K \| M)$ where K is secret.

- For the ideal hash nothing is revealed about the key
- Preimage resistance seems to indicate this is the case
- A trivial attack shows this is **not** the case:
  - We can find the keyed hash of related messages
  - Just add padding and append extra data:

$$h' = H(K \| M \| \text{pad} \| M')$$

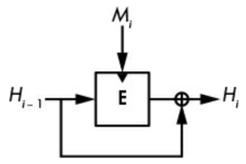
  - We can start computing h' from h
  - So h reveals something useful about K!

This is problematic for message authentication.

It is also problematic for proofs of storage and similar applications.

Compression Functions: Davis -Meyer

All popular MD hash functions use the Davis-Meyer construction:



Many variants convert block ciphers into compression functions.

Counter-intuitively:

- Key input takes a message!
- This construction creates a fixed point $h_i = \mathbf{D}(0, M)$ in MD!
- This can be checked in the SHA-256 compression function
- Not a problem?

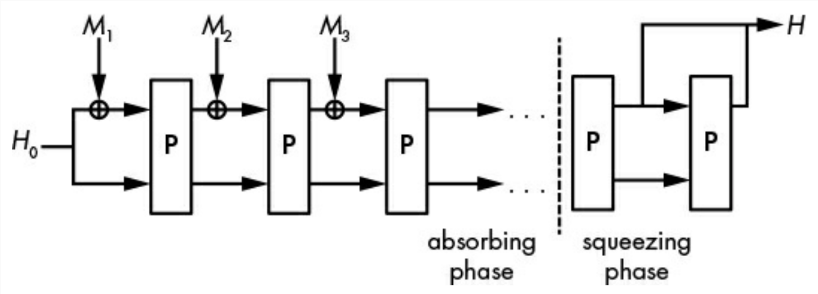
Sponge Construction

A more recent alternative to MD is the sponge construction.

It relies on a fixed (non-keyed) permutation.

It is very versatile:

- Varying input/output lengths
- PRGs and stream ciphers
- PRFs and keyed hashes



Sponge Construction (2)

Sponge operates in two phases: absorb and squeeze.

The state is the same size w as the permutation input.

Absorb Starting from fixed initial value h_0 gradually accumulate message into state:

- Message is broken in blocks of size r (rate)
- Block is smaller than state size
- Block XOR'ed into state
- Permutation recomputed

Squeeze The dual process iteratively constructs output:

- Output is constructed block by block
- Permutation is computed over entire state
- Block-sized part of state is accumulated in output

Part #3: Concrete Hash Functions

MD5

Broken! 128-bit output.

Most popular hash function until broken in 2005.

These days it takes seconds to find collisions.

The SHA function family (next) uses essentially same design.

Secure Hash Algorithm (SHA) Family

Standardized by NIST in the US but international de-facto standard.

SHA-0 was published in 1993 but replaced with SHA-1 in 1995:

- Both with 160-bit outputs
- Vulnerability not public at the time
- Later discovered collision attack in $2^{60} \ll 2^{80}$ operations
- Later attacks brought effort to 2^{33}

SHA-1 was unbroken until very recently.

Currently most applications use SHA-2 (256 or 512 bits):

- Still same design principles, but larger parameters

Future applications adopting SHA-3 will evolve to the Sponge.

SHA-1 internals

Merkle-Damgård with Davis-Mayer compression function.

The block cipher used in the compression function called SHACAL.

Message blocks are 512-bits and hashes are 160-bits long.

Davis-Meyer addition not XOR: five 32-bit additions.

```
SHA1-blockcipher(a, b, c, d, e, M) {
    W = expand(M)
    for i = 0 to 79 { // K are constants
        new = (a <<< 5) + f(i, b, c, d) + e + K[i] + W[i]
        (a, b, c, d, e) = (new, a, b >>> 2, c, d) }
    return (a, b, c, d, e)
}
```

Insecure! Estimated collisions 2^{63} in 2005; actual collisions in 2017.

SHA-2 Family

Family of 4 hash functions SHA-[224, 256, 384, 512].

Three digit identifier gives the output length.

Increasing parameters and improved internal block ciphers.

SHA-224 and 256 still use 512 bit blocks (64 rounds).

SHA-224 is identical to SHA256 with different IV/truncated output.

SHA-384 and 512 are related in the same way.

SHA-512 compression function very similar but 80 rounds.

No non-generic attacks on any of these hash functions:

- Still SHA-3 was (prudently) developed with different design

SHA-3

Keccak selected in 2009 after 3 year NIST SHA-3 competition.

Competition called for new designs in case of attacks on SHA-2.

Keccack is very different and very flexible:

- sponge based with 1600-bits permutation (in SHA-3)
- blocks can be 1152, 1088, 832, 76 bits
- corresponding to 224, 256, 384 or 512 bits outputs
- as a bonus we get the SHAKE functions:
    - SHAKE128 and SHAKE256
    - eXtendable Output Functions (XOFs)
    - you can specify the output length!
    - Why would these be useful?

Part #4: Keyed Hashing

MACs as Keyed Hashes

Short summaries of potentially large messages:

- Called a hash if everything is public
- *Keyed hashing* is the intuitive view of a MACs

A Message Authentication Code (MAC):

- symmetric authentication: $T \leftarrow \mathrm{MAC}(M, K)$
- T guarantees message M creator knew a secret key
- implies message M not changed since creation

Message Authentication Codes (MACs)

Typical use of MAC, e.g., SSH, IPSec, TLS:

- two parties want message authentication and integrity
- some form of set-up/agreement to establish common key K
- sender computes $T \leftarrow \mathrm{MAC}(M, K)$ and sends (M, T)
- receiver gets (M, T) recomputes $T' \leftarrow \mathrm{MAC}(M, K)$
- receiver accepts if $T = T'$

Acceptance means: M was produced by someone knowing K.

Note that in this process the message is public!

MACs do not give confidentiality: sending only T makes no sense.

Encryption does not give authenticity: ciphertexts can be mauled.

Real world: need to combine encryption schemes and MACs.

MAC security

Standard notion is UF-CMA:

- Unforgeability
- Chosen message attacks

Security experiment as follows:

- Experiment chooses K
- Attacker (adaptively) outputs M to get $T \leftarrow \mathrm{MAC}(M, K)$
- Eventually attacker outputs (M^*, T^*)

Attacker wins if tag is valid and M^* not authenticated by experiment.

Obviously implies attacker cannot recover K. (Why?)

MAC security (2)

Crucial insight:

- MAC on its own does not protect against replay attacks
- Suppose network scenario:
  - Attacker sees authenticated message (M, T)
  - Delivers (M, T) multiple times
  - MAC will verify every time!
- Simple technique, impose message never repeats in network:
  - Prepend counter and keep counter as state on both sides
  - Prepend timestamp (local clock reading)
  - How should receiver operate in both cases?

Part #5: Constructing MACs

Some history

MACs constructed from hash functions and block ciphers.

Simplest construction: prefix key.

$$\mathrm{MAC}(K, M) = \mathrm{H}(K \| M) \quad \text{or} \quad \mathrm{PRF}(K, M) = \mathrm{H}(K \| M)$$

Merkle-Damgård hashing yields insecure MAC and PRF!

- Given (M, T) attacker outputs $\mathrm{H}(K \| M \| \mathrm{pad} \| M')$
- This can be computed just from T and M'
- It's called a **length extension attack**

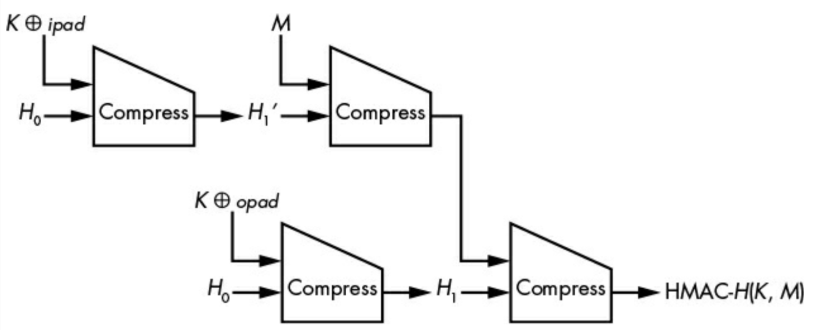
Resistance to such attacks was requirement in SHA-3 competition:

- Abandon MD construction
- Include explicit keyed hash

HMAC Construction

When instantiated with MD construction:

- Compression function is PRF $\Rightarrow$ secure MAC
- HMAC is simply $\mathrm{H}((K \oplus \mathrm{opad}) \| \mathrm{H}((K \oplus \mathrm{ipad}) \| M))$
- ipad and opad are constants: align to block size



On collision-based forgeries

Hash function collisions $\Rightarrow$ hash-based MAC forgeries.

However, attacker cannot easily search for them: key is unknown.

Obviously collisions at MAC output also yield forgeries:

- **This is true for any MAC.**
- **Collisions occur whp after** $2^{\frac{n}{2}}$ **MACs issued.**
- Could happen earlier if size of chained value is smaller than n bits

Building MACs directly from block ciphers

We have seen: block ciphers $\Rightarrow$ hash functions $\Rightarrow$ MACs.

There are also direct constructions: CMAC is used in IPSec.

CMAC is an improvement over CBC-MAC:

- Take CBC mode of operation
- Fix IV to all zero block
- Take last ciphertext block as tag

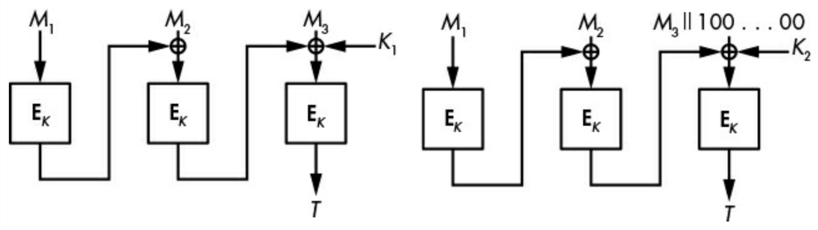
CBC-MAC turns out to be insecure:

- Can forge MACs after just two chosen authenticated messages.

CMAC internals

CMAC fixes CBC-MAC by processing last block differently:

- All blocks but last are processed like in CBC-MAC
- Two keys K_1 and K_2 are derived from K:
  - $L \leftarrow \mathbf{E}(K, 0)$
  - $K_1 = (L \ll 1) \oplus (0\text{x}00..0087 * \mathrm{LSB}(L))$
  - $K_2 = (K_1 \ll 1) \oplus (0\text{x}00..0087 * \mathrm{LSB}(K_1))$



Custom MAC constructions

More efficient MAC constructions are designed from scratch.

Poly1305 is one such construction by D.J.Bernstein.

It is based on

- Universal hash functions
- Wegman-Carter construction

Universal hash functions

Universal hash functions are a weak form of hashing

- Don't need to be collision resistant
- They are parametrised by a key: $\mathrm{UH}(K, M)$
- They guarantee that for any two fixed messages $M_0 \neq M_1$:

$$\Pr[\mathrm{UH}(K, M_0) = \mathrm{UH}(K, M_1)] \leq \epsilon$$

When K is random and for ϵ very small.

There is no other security requirement $\Rightarrow$ easy to construct.

We can use a universal hash function as a MAC:

- **But only one message can be authenticated**

Wegman-Carter Construction

The Wegman-Carter construction:

- Converts universal hash function
- Into a fully secure MAC
- Using a PRF or block cipher

Intuition: encrypt universal hash value

$$\mathsf{UH}(K_1, M) \oplus \mathsf{PRF}(K_2, N)$$

- The full MAC key is (K_1, K_2)
- N is a public value that must never repeat
- This can be kept as a counter or generated at random

Poly1305-AES: Wegman-Carter in Practice

Initial proposal used AES as the Wegman-Carter PRF.

The universal hash function uses prime $p = 2^{130} - 5$.

$$\text{Poly1305}((K_1, K_2), M) = (M_1 K + \ldots + M_n K^n \pmod{p}) + \text{AES}(K_2, N)$$

Blocks are 128 bits and last block is padded with 100 . . .

All blocks set bit 129 so MSB is 1.

The final addition is performed modulo 2^{128} (why?).

TLS recommends Poly1305 with ChaCha20 rather than AES.

Thank you!

mbb@fc.up.pt

http://www.dcc.fc.up.pt/~mbb