Cryptology complementary

✧

Hash functions, collisions

Pierre Karpman
pierre.karpman@univ-grenoble-alpes.fr
https://www-ljk.imag.fr/membres/Pierre.Karpman/tea.html

2018–04–05

Cryptographic hash functions

Hash function

A hash function is a mapping $\mathcal{H} : \mathcal{M} \to \mathcal{D}$

So it really is just a function...

Usually:

- $\mathcal{M} = \bigcup_{\ell < N} \{0,1\}^\ell$, $\mathcal{D} = \{0,1\}^n$, $N \gg n$
- N is typically $\geq 2^{64}$, $n \in \{\cancel{128}, \cancel{160}, \cancel{224}, 256, 384, 512\}$

Also popular now: extendable-output functions (XOFs): $\mathcal{D} = \bigcup_{\ell < N'} \{0,1\}^\ell$

- Hash functions are *keyless*
- So, how do you tell if one's **good**?

Three classical security properties

1. **First preimage**: given t, find m s.t. $\mathcal{H}(m) = t$
2. **Second preimage**: given m, find $m' \neq m$ s.t. $\mathcal{H}(m) = \mathcal{H}(m')$
3. **Collision**: find $(m, m' \neq m)$ s.t. $\mathcal{H}(m) = \mathcal{H}(m')$

Generic complexity:
1), 2): $\Theta(2^n)$;
3): $\Theta(2^{n/2})$ $\leftsquigarrow$ "Birthday paradox"
(There's actually more...)

Birthday paradox

If all outputs of $\mathcal{H}$ are independent and uniformly random, one may expect to find one collisions among $\sqrt{2^n}$ inputs

- N elements define $\approx N^2$ pairs, which have independent probability 2^{-n} of forming a collision

Why do we care? Applications!

Hash functions are useful for:

- Hash-and-sign (RSA signatures, (EC)DSA, ...)
- building MACs (HMAC, ...)
- Password hashing (with a grain of salt)
- Hash-based signatures (inefficient but PQ)
- In padding schemes (OAEP, ...)
- Etc.

$\Rightarrow$ A versatile building block, but only a building block

So, how do you build hash functions?

- Objective #1: be secure
- Objective #2: be efficient
  - Even more than block ciphers!
  - $\Rightarrow$ work with limited amount of memory

So...

- (#2) Build $\mathcal{H}$ from a *small component*
- (#1) Prove that this is okay

What kind of small component?

Compression function

A compression function is a mapping $\mathfrak{f} : \{0,1\}^n \times \{0,1\}^b \to \{0,1\}^n$

- A family of functions from n to n bits
- Not unlike a block cipher, only not invertible

Permutation

A permutation is an invertible mapping $\mathfrak{p} : \{0,1\}^n \to \{0,1\}^n$

Yes, very simple

- Like a block cipher with a fixed key, e.g. $\mathfrak{p} = \mathcal{E}(0, \cdot)$

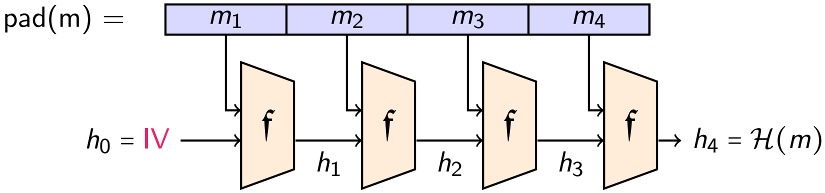
From small to big (compression function case)

Assume a good $\mathfrak{f}$

- Main problem: fixed-size domain $\{0,1\}^n \times \{0,1\}^b$
- Objective: *domain extension* to $\bigcup_{\ell<N}\{0,1\}^\ell$

The classical answer: the Merkle-Damgård construction (1989)

MD: with a picture



That is: $\mathcal{H}(m_1 \| m_2 \| m_3 \| \ldots) = \mathfrak{f}(\ldots \mathfrak{f}(\mathfrak{f}(\mathfrak{f}(\mathrm{IV}, m_1), m_2), m_3), \ldots)$

$\mathrm{pad}(m) \approx m \| 1000 \ldots 00 \langle \text{length of } m \rangle$

MD: does it work?

Efficiency?

- Only sequential calls to $\mathfrak{f}$
- $\Rightarrow$ fine

Security?

- Still to be shown
- Objective: *reduce* security of $\mathcal{H}$ to that of $\mathfrak{f}$
  - "If $\mathfrak{f}$ is good, then $\mathcal{H}$ is good"
- True for collision and first preimage, **false** for second preimage
- Won't see the details, though (in the end, everything is quite fine)

So how to do $\mathfrak{f}$?

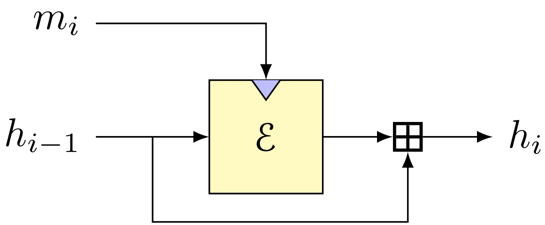
1. Start like a block cipher
2. Add *feedforward* to prevent invertibility

Examples:
"Davies-Meyer": $\mathfrak{f}(h, m) = \mathcal{E}_m(h) \boxplus h$
"Matyas-Meyer-Oseas": $\mathfrak{f}(h, m) = \mathcal{E}_h(m) \boxplus m$

- Systematic analysis by Preneel, Govaerts and Vandewalle (1993). "PGV" constructions
- Then rigorous proofs (in the ideal cipher model) (Black et al., 2002), (Black et al., 2010)

Re: Davies-Meyer

Picture:



Used in MD4/5 SHA-0/1/2, etc.

Re: Re: Davies-Meyer

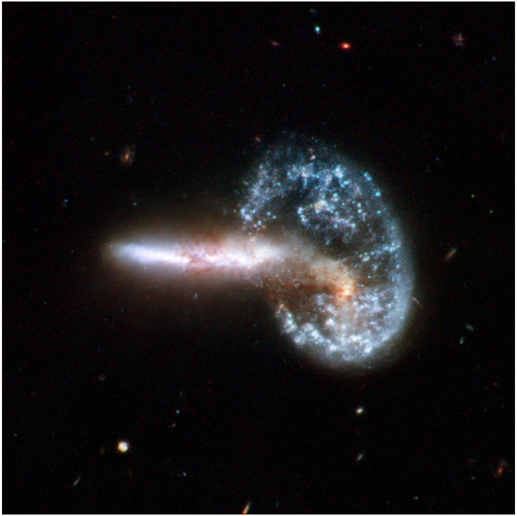
Why is the "message" the "key"?

- Disconnect chaining value and message length!
- $\mathcal{E}$'s block length: fixed by security level
- $\mathcal{E}$'s key length: fixed by "message" length
- Large "key" $\Rightarrow$ more efficient
- Example: MD5's "block cipher": 128-bit blocks, 512-bit keys

DM incentive: use very simple *message expansion* ("key schedules")

- To be efficient!
- Warning: can be a source of weakness

Let's collide now!



Let's collide now!

Computing collisions for a (generic) function $\mathcal{F} : \mathcal{I} \to \mathcal{O}$ has many applications in crypto, e.g.:

- Generic attacks on hash functions
- Generic discrete logartihm computations
- Factorization
- Generic attacks on mode of operations
- Intermediate step in some dedictated attacks

Collision finding: how?

Finding a collision in $\{\mathcal{F}(i),\ i \in [0, M]\}$ for some M (e.g. $\approx \sqrt{\#\mathcal{O}}$)
The easy way:

1. Incrementally store the $\mathcal{F}(i)$ in a data structure w/ efficient insertion & comparison
   - Sorted list, hash table, etc.
2. Look for a duplicate at every insertion

Quite simple; easily parallelizable; huge memory complexity

Collision finding: memoryless, sequential

Objective: decreasing the memory complexity of collision search

- One idea: if $\mathcal{O} \subseteq \mathcal{I}$, look at iterates of $\mathcal{F}$: compute $\mathcal{F}(x)$, $\mathcal{F}(\mathcal{F}(x))$, etc. for some x
- If $\mathcal{F}^i(x) = \mathcal{F}^j(x)$, then $\mathcal{F}^{i-1}(x)$ and $\mathcal{F}^{j-1}(x)$ form a collision for $\mathcal{F}$
- Question 1: how soon does such an event happen?
- Question 2: how is this useful?

Collision finding: Pollard ρ (A. 1)

Rho (ρ) structure of $\mathcal{F}^r(x)$, $r \in \mathbb{N}$:

- If $\mathcal{F}^i(x) = \mathcal{F}^j(x)$, $i < j$ the smallest values where this happens, then $\mathcal{F}^i(x) = \mathcal{F}^{i+k(j-i)}(x)$
- $\Rightarrow \mathcal{F}^r(x)$ has a *cycle* of length $j - i$
- $\Rightarrow \mathcal{F}^r(x)$ has a *tail* of length i

Proposition

For a random function $\mathcal{F}$, for a random starting point x, the expected cycle and tail length of $\mathcal{F}^r(x)$ are both $\approx \sqrt{\#\mathcal{O}}$

$\Rightarrow$ One can look for collisions in $\mathcal{F}^r(x)$ instead of $\mathcal{F}(\cdot)$ directly

Collision finding: Pollard ρ (A. 2)

To find a collision in $\mathcal{F}$, find the tail (λ) and cycle (μ) length of $\mathcal{F}^r(x)$ for some x

- Can be done with constant (in $\mathcal{F}$'s parameter sizes) memory, using Floyd's cycle-finding algorithm:

1. Compute $\mathcal{F}^i(x)$, $\mathcal{F}^{2i}(x)$ in parallel, $i = 1, \ldots$
2. Find k s.t. $\mathcal{F}^k(x) = \mathcal{F}^{2k}(x)$
    - Most likely, $\mathcal{F}^{k-1}(x) = \mathcal{F}^{2k-1}(x)$, so the collision is "trivial"
    - (But one has $k - \lambda \equiv 2k - \lambda \equiv \lambda + 2(k - \lambda) \mod \mu$, so $k \equiv 0 \mod \mu$)
3. Find k' s.t. $\mathcal{F}^{k'}(x) = \mathcal{F}^k(x)$; set $\mu = k' - k$
4. Compute $\alpha = \mathcal{F}^\mu(x)$; find k'' s.t. $\mathcal{F}^{\mu+k''}(x) = \alpha$; set $\lambda = \alpha - \mu$
5. $\mathcal{F}^{\lambda-1}(x)$ and $\mathcal{F}^{\lambda+\mu-1}(x)$ form a non-trivial collision

$\Rightarrow$ Constant memory complexity, time complexity = $\Theta(\sqrt{\#\mathcal{O}})$, with small constant

Collision finding: Pollard ρ example

Let $\mathcal{F}^r(0)$ be such that $\lambda = 193$, $\mu = 171$

- $-193 \equiv 149 \mod 171$
  - At $i = 342 = 193 + 149$, $i - 193 = 149 \equiv 149 \mod 171$
  - And $2i - 193 = 193 + 2 \times 149 \equiv -149 + 2 \times 149 \mod 171 \equiv 149 \mod 171$
- $\mathcal{F}^{342}(0) = \mathcal{F}^{684}(0) = \mathcal{F}^{513}(0)$
- $\mu = 513 - 342 = 171$
- $\mathcal{F}^{193}(0) = \mathcal{F}^{364}(0) \Rightarrow \lambda = 193$
- $\mathcal{F}^{192}(0)$ and $\mathcal{F}^{363}(0)$ form a collision

Parallel collision search

- Limitation of the ρ approach: it is sequential
- In the real world, one wants parallel approaches to hard problems (if possible)
- Still with memory $\ll$ time

$\Rightarrow$ Parallel collision search (van Oorschot & Wiener, 1999)

- Define a *distinguished property* for the outputs of $\mathcal{F}$ (e.g. $\mathcal{F}(x)$ starts with z zeroes for some z)
- For as many threads t, compute "chains" of $\alpha_t^i = \mathcal{F}^i(s_t)$ for a random s_t until α_t^i is distinguished, then store (s_t, α_t^i, i) e.g. in a hash table, then start again
- If (s_t, α_t^i, i), $(s_{t'}, \alpha_{t'}^j, j)$ are s.t. $\alpha_t^i = \alpha_{t'}^j$, $i < j$, compute $s'_{t'} = \mathcal{F}^{j-i}(s_{t'})$; find k s.t. $\mathcal{F}^k(s_t) = \mathcal{F}^k(s'_{t'})$

PCS comments

- One must choose the distinguished property s.t.
  - Not so many points are distinguished (to limit memory complexity)
  - Recomputing a chain from the start is not too long (to limit time complexity)
- If (s_t, α_t^i, i), $(s_{t'}, \alpha_{t'}^j, j)$ are s.t. $\mathcal{F}^k(s_{t'}) = s_t$ for some k, the collision is trivial
- If a chain enters a cycle w/o distinguished points, it never terminates
- For a "well-chosen" distinguishing property, ≈ optimal speed-up: T threads decrease running-time by a factor T