Introduction to cryptology (GBIN8U16)

✧

Collisions brief

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

2019–04

Collision finding: how?

A *collision* in a function $\mathcal{F} : \mathcal{I} \to \mathcal{O}$ is a pair of two distinct inputs that evaluate to the same image, i.e. a, $b \neq a$ s.t. $\mathcal{F}(a) = \mathcal{F}(b)$

How to find one 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. $\mathcal{F}^k(x) = \mathcal{F}^{2k}(x)$
   - Happens for first $k := a + \lambda > \lambda$ s.t. $k \equiv 0 \mod \mu$
   - $k - \lambda = a \equiv -\lambda \mod \mu$, $2k - \lambda = 2a + \lambda = k + a \equiv a \mod \mu$
   - Most likely, $\mathcal{F}^{k-1}(x) = \mathcal{F}^{2k-1}(x)$, so the collision is “trivial”
3. Find k' s.t. $\mathcal{F}^{k'}(x) = \mathcal{F}^k(x)$; set $\mu = k' - k$
4. Find k'' s.t. $\mathcal{F}^{\mu+k''}(x) = \mathcal{F}^{k''}(x)$; set $\lambda = k''$
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 $k = 171 \times 2 = 342 = 193 + 149$, $k - 193 = 149 \equiv 149 \mod 171$
- And $2k - 193 = 193 + 2 \times 149 \equiv -149 + 2 \times 149 \mod 171 \equiv 149 \mod 171$

- $\mathcal{F}^{342}(0) = \mathcal{F}^{684}(0) = \mathcal{F}^{513}(0)$
- $\mu = 513 - 342 = 171$
- $\mathcal{F}^{193}(0) = \mathcal{F}^{364}(0) \Rightarrow \lambda = 193$
- $\mathcal{F}^{192}(0)$ 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 $\alpha_i = \mathcal{F}^i(s_t)$ for a random s_t until α_i is distinguished, then store (s_t, α_i, i) e.g. in a hash table, then start again
- If (s_t, α_i, i), $(s_{t'}, \alpha_j, j)$ are s.t. $\alpha_i = \alpha_j$, $i < j$, 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_i, i), $(s_{t'}, \alpha_j, j)$ 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, $\approx$ optimal speed-up: T threads decrease running-time by a factor T

More collision-based attacks: TMTO

- Consider a key-recovery attack on a block cipher: one wants to find a secret key k used with $\mathcal{E}$
- In a chosen-plaintext scenario $\rightsquigarrow$ e.g. inverting $x \mapsto \mathcal{E}(x, 0)$: a "random" function
- Can be done with time $= 2^\kappa$, negligible memory
- Assume that one can afford a *huge offline* precomputation *once*
  - Can be done with memory $= 2^\kappa$, negligible (?) *online* time (after a precomputation of time 2^κ)
- Something in between?

$\Rightarrow$ Can use a *time-memory tradeoff* to speed-up the key search (Hellman, 1980)

- (May be used to invert other functions as well)

TMTO: the idea

Offline (precomputation) phase:

- Form many iteration chains for $x \mapsto \mathcal{E}(x, 0)$, for random starting points s, storing the starting and ending points α in e.g. a hash table
  - That is, compute $s \to s^0 \to s^1 \to \ldots$, with $s^0 = \mathcal{E}(s, 0)$, $s^1 = \mathcal{E}(s^0, 0)$, etc.
- Use $\approx M$ chains of length $\approx T$
  - The precomputation takes time MT

TMTO: the idea (cont.)

Online phase:

- Ask for $c_0 = \mathcal{E}(k, 0)$
- Compute the chain $c_0 \to c_0^0 \to \ldots$ starting at c_0
- Search a collision of this chain with one of the M stored ending points α_i
- Restart computing the chain ending in α_i from s_i, find t s.t. $\mathcal{E}(s_i^t, 0) = c_0 \Rightarrow k = s_i^t$

This online phase is successful if c_0 is part of a chain

TMTO: comments

- The memory complexity is M
- The online phase (if successful) takes time T (ignoring the cost of searching for collisions among stored ending points)
- The success probability is $\approx MT/2^\kappa$ (assuming tha all chains are distinct)
  - Take $MT \approx 2^\kappa$?
  - Does not work: when $MT^2 \approx 2^\kappa$, new chains collide with exisiting ones w.h.p. $\rightsquigarrow$ does not cover more keyspace
    - For instance, one chain of length $2^{\kappa/2}$ forms a ρ w.h.p.
  - Take $M = T = 2^{\kappa/3} \Rightarrow$ success probability of $2^{-\kappa/3}$

TMTO: more comments

- One may increase the success probability of Hellman's TMTO by considering N "independent" mappings $x \mapsto \varphi(\mathcal{E}(x, 0))$
  - E.g., take φ to be a bit permutation
- If $N = M = T \approx 2^{\kappa/3}$, the success probability ≈ 1, the total time and memory complexities are $MN = TN = 2^{2\kappa/3}$
- In practice, one would (probably) want the memory complexity to be $\ll$ the time complexity
- In practice, checking if $c_0^i = \alpha$ for some α is slow (memory accesses are slow compared to computations) $\Rightarrow$ only use α with a distinguished property $\Rightarrow$ only check when c_0^i is distinguished too

TMTO: even more comments

- If one wants to invert a permutation, Hellman's TMTO $\rightsquigarrow$ Baby-step/Giant-step
  - No chain collisions $\Rightarrow$ better complexity
- This TMTO is somehow similar to PCS, but only *one* collision is useful!

More collision-based attacks: MiTM

Suppose one has a good block cipher
$\mathcal{E} : \{0,1\}^\kappa \times \{0,1\}^n \to \{0,1\}^n$, with a *small* κ (e.g. 64)
How can one define $\mathcal{E}'$ from $\mathcal{E}$ with a *larger* key?

- One idea: "double-encryption": Take $\mathcal{E}'(k_0||k_1, \cdot) = \mathcal{E}(k_1(\mathcal{E}(k_0, \cdot))$
- This is quite simple
- But doesn't really work...

Meet-in-the-Middle: how?

Assume $n \geq 2\kappa$ and one knows that $\mathcal{E}'(k_0||k_1, 0) = c_0$

1. Compute $L_0[i] = \mathcal{E}(i, 0)$, $i \in \{0,1\}^\kappa$
2. Compute $L_1[i] = \mathcal{E}^{-1}(i, c_0)$, $i \in \{0,1\}^\kappa$
3. Search for a match between L_0 and L_1
    - All collisions $L_0[x] = L_1[y]$ give a candidate $x||y$ for $k_0||k_1$

- The time complexity is $\approx 2^\kappa \Rightarrow$ not much more than for $\mathcal{E}$ alone
- (But memory complexity increases to 2^κ)
- (And an attack interrupted after t tries has success prob. $\approx t^2/2^{2\kappa}$ instead of $t/2^\kappa$)

Alternatives to double encryption

As double-encryption does not increase security so much, one may instead:

- Use "triple-encryption" (this time not so bad, but quite slow) $\rightsquigarrow$ Triple-DES :S
- Use an "FX" construction: $\mathcal{E}'(k_0 || k_1, x) = \mathcal{E}(k_0, x \oplus k_1) \oplus k_1$ (fast; not so bad, but not ideal)
- Use combinations of the two