Organization

Slides, TP / TD sheets and code (only for this part of the course):
`andreschrottenloher.github.io/pages/teaching.html`

Questions at:
andre(dot)schrottenloher(at)inria(dot)fr

Cryptanalysis
Part I: Collisions and random functions

André Schrottenloher

Inria Rennes
Team CAPSULE

Introduction

What is cryptanalysis?

- "Breaking" cryptosystems?
- More generally: **evaluating the security**

- Looking for an **unpredicted** behavior of the scheme;
- Looking for a better algorithm to attack it.

The situation differs between:

- asymmetric and symmetric crypto;
- the provable setting (modes of operation) & the unprovable setting (primitives).

Remark

- Most often, our "attacks" are **infeasible** (and we know that)
- They are infeasible because of the resources (time / memory) or the attacker scenario is (looks?) impractical (related-key, etc.)
- We're at the lowest level of cybersecurity, so we **cannot afford the smallest weakness**
- Besides, weaknesses have a tendency to become worse over time.

Important principles:

$$\text{Security} = \int_0^{+\infty} \text{Cryptanalysis effort } dt$$

"We can only gain confidence through a continuous (public!) cryptanalysis effort"

$$\frac{d(\text{attack complexity})}{dt} < 0$$

"An attack will only improve over time"

Security levels

Security level

- A security level is expressed in "bits of security".
- 120 bits of security $\simeq$ the attack requires 2^{120} operations to execute.

What is feasible "in practice"?

- $1000 \simeq 2^{10}$
- $4GHz \simeq 2^{32}$ operations per second on a CPU
- multi-core CPUs

With massively parallelized GPUs: 2^{60} is accessible.

The Bitcoin network computes 2^{90} SHA-256 per year using a massive amount of ASICs.

However computing 2^{128} hashes would require more energy than vaporizing all the Earth's oceans $\implies$ 128 bits of security is good.

Hash Function Security

Hash functions

A **hash function** is a public function that takes a variable-length message and outputs a fixed-length digest: $H : \{0,1\}^* \to \{0,1\}^n$.

The "ideal" behavior of a hash function is to look like a completely random function $\{0,1\}^* \to \{0,1\}^n$.

This lecture

- Focus on **compression functions** and / or **small-range hashing**: the input has size $n + m$.
- Typically used with the Merkle-Dåmgard domain extender to produce large-scale hash functions.

The hash function output should not give **any information** on the input.

Preimage resistance

Fix $H : \{0,1\}^* \to \{0,1\}^n$.

Preimage resistance

For $t \leftarrow \{0,1\}^n$, it should be difficult to find m such that $t = H(m)$.

- By brute force, this takes time $\mathcal{O}(2^n)$ (to succeed with constant probability)
- So it should take time $\mathcal{O}(2^n)$

Preimage resistance

Fix $H : \{0,1\}^* \to \{0,1\}^n$.

Preimage resistance

For $t \leftarrow \{0,1\}^n$, it should be difficult to find m such that $t = H(m)$.

- By brute force, this takes time $\mathcal{O}(2^n)$ (to succeed with constant probability)
- So it should take time $\mathcal{O}(2^n)$

Example: password authentication.

- One stores only $H(\text{password})$.
- An attacker having access to the database cannot find the passwords.

Second preimage resistance

Fix $H : \{0,1\}^* \to \{0,1\}^n$.

For $x \leftarrow \{0,1\}^m$, it should be difficult to find $y \neq x$ such that $H(y) = H(x)$.

- By brute force, this takes time $\mathcal{O}(2^n)$ (to succeed with constant probability)

Second preimage resistance

Fix $H : \{0,1\}^* \to \{0,1\}^n$.

For $x \leftarrow \{0,1\}^m$, it should be difficult to find $y \neq x$ such that $H(y) = H(x)$.

- By brute force, this takes time $\mathcal{O}(2^n)$ (to succeed with constant probability)

Example: hash-and-sign signatures

- Sign H(message)
- Integrity of files
- One cannot **forge**: find another file with a valid signature

Collision resistance

Collision resistance

- Producing a collision (pair $x \neq y$ such that $H(x) = H(y)$) should take time $\mathcal{O}(2^{n/2})$ (why? next slides)

This is the same as long as the input size is $\geq n$ bits.

Chosen-prefix collisions

Fix $p_1, p_2 \in \{0,1\}^m$, we look for a collision of the form:

$$H(p_1 \| m_1) = H(p_2 \| m_2)$$

- Yields practical attacks: forgery of certificates, malicious GPG / SSH keys
- Flame malware using chosen-prefix collisions on MD5

On the existence of collisions / preimages

There exists collisions & preimages

On the existence of collisions / preimages

There exists collisions & preimages (the message space is much bigger than the hash space).

There **exists** an algorithm that returns in **constant time** a collision for **any** hash function.

$\implies$ however, we don't know how to write it down.

Some examples

MD5 (broken)

- 128-bit hash (RFC 1321, Rivest, 1992)
- Collisions found (Wang, Yu, 2005)
- Forgery of certificates (Stevens et al., 2009)

SHA-0 (broken)

- 160-bit hash (NSA, 1993)
- Collisions (theoretical) in 1998 (Joux, Chabaud)

SHA-1 (broken)

- 160-bit hash
- Theoretical collisions in 2005 (Wang et al.)
- Practical collisions in 2017 (Stevens et al., 2009)
- Chosen-prefix collisions (Leurent, Peyrin, 2020)
- Still used a lot ...

Current standards

SHA-2

- Published by NSA in 2001
- Family of hash functions of 224, 256, 384, 512 bits

SHA-3

- a.k.a. Keccak, winner of an open competition organized by NIST
- Sponge function, published in 2015
- Outputs of 224, 256, 384, 512 bits

Random Functions

Random functions

- What is a **truly random function**? It's a function that we picked at random.
- **Choice 1:** pick the entire function at random before running the algorithm;
- **Choice 2:** ("lazy") build the table of the function by picking random outputs whenever needed.

$\Longrightarrow$ these two cases are equivalent.

For a **random function** $\{0,1\}^* \to \{0,1\}^n$, (second) preimages can be found in time $\mathcal{O}(2^n)$. This is **tight**.

$\Longrightarrow$ a good hash function should offer the same guarantee.

Interlude: birthday paradox

Lemma

Let $y_1, \dots, y_\ell$ be random (uniform) samples in a set of size N. Then there are two distinct i, j such that $y_i = y_j$:

- With prob. at most $\ell^2/2N$
- With prob. at least $\frac{\ell(\ell-1)}{4N}$ if $\ell \leq \sqrt{2N}$

Intuition:

- Each pair has probability $1/N$ of forming a collision
- There are $\ell^2/2$ pairs $\implies$ upper bound
- But they are not independent

Interlude: birthday paradox (ctd.)

Write $NoColl_i$ the event "no collision among $y_1, \ldots, y_i$."

$\Pr[NoColl_\ell] = \Pr[NoColl_1] \cdot \Pr[NoColl_2 | NoColl_1] \cdots \Pr[NoColl_\ell | NoColl_{\ell-1}]$.

Also: $\Pr[NoColl_1] = 1$, and $\Pr[NoColl_{i+1} | NoColl_i] = 1 - i/N$ (the new element must be different from the i previous ones)

$$\implies \Pr[NoColl_\ell] = \prod_{i=1}^{\ell-1} (1 - i/N)$$

Now we do some bounding: $\forall i, 1 - i/N \leq e^{-i/N}$:

$$\Pr[NoColl_\ell] \leq e^{-\sum_{i=1}^{\ell-1} i/N} = e^{-\ell(\ell-1)/2N} \quad .$$

And for $x < 1$, $1 - x/2 \geq e^{-x}$:

$$\Pr[Coll] = 1 - \Pr[NoColl_\ell] \geq 1 - e^{-\ell(\ell-1)/2N} \geq \frac{\ell(\ell-1)}{4N} \quad .$$

Interlude: birthday paradox (ctd.)

The average number of samples to pick before a collision occurs is:

$$\sqrt{\pi/2} \cdot 2^{n/2}$$

Proof:

$$\begin{aligned}\mathbb{E}(\text{nb samples}) &= \sum_{\ell>0} \Pr\left[NoColl_\ell\right] \simeq \sum_{\ell>0} e^{-\ell^2/2^{n+1}} \simeq \int_0^{+\infty} e^{-x^2/2^{n+1}} dx \\ &= \sqrt{\pi/2} \cdot 2^{n/2} \quad .\end{aligned}$$

Random function collisions

Naive algorithm:

Random function collisions

Naive algorithm:

1. pick $\mathcal{O}(2^{n/2})$ random inputs x
2. evaluate them and put the $(H(x), x)$ pairs in a hash table
3. sort by output and find a collision

$\implies$ we have an algorithm in time $\mathcal{O}(2^{n/2})$, memory $\mathcal{O}(2^{n/2})$ to find collisions.

For a **random function** $\{0,1\}^* \to \{0,1\}^n$, collisions can be found in time $\mathcal{O}(2^{n/2})$. This is **tight**.

$\implies$ a good hash function should offer the same guarantee.

Multicollisions

An ℓ-collision of H is a tuple of ℓ distinct entries: $x_1, \ldots, x_\ell$ such that $H(x_1) = \ldots = H(x_\ell)$.

For a **random function** $\{0,1\}^* \to \{0,1\}^n$, ℓ-collisions can be found in time $\mathcal{O}\left(2^{\frac{\ell-1}{\ell}n}\right)$. This is **tight**.

Algorithm: pick $2^{\frac{\ell-1}{\ell}n}$ elements at random $\implies$ $2^{(\ell-1)n}$ tuples $\implies$ one of them satisfies the multicollision property.

Pollard's Rho

A chain

- Consider $H : \{0,1\}^n \rightarrow \{0,1\}^n$ (if the input domain is too large, fix some of the input).
- Take x_0 at random in $\{0,1\}^n$

Evaluate:

$$x_1 = H(x_0), x_2 = H(x_1), \ldots, x_i := H^i(x)$$

A chain

- Consider $H : \{0,1\}^n \to \{0,1\}^n$ (if the input domain is too large, fix some of the input).
- Take x_0 at random in $\{0,1\}^n$

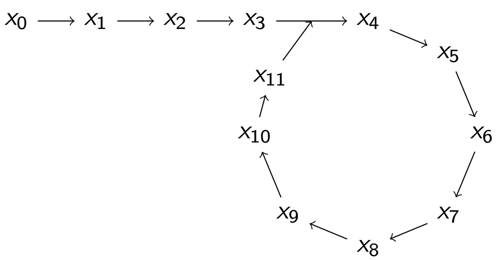
Evaluate:

$$x_1 = H(x_0), x_2 = H(x_1), \ldots, x_i := H^i(x)$$

Fact

The chain **cannot be infinite**. There exists some $i \neq j$ such that $H^i(x) = H^j(x)$.

(Pollard's) Rho



Birthday property!

- The first pair i, j such that $H^i(x) = H^j(x)$ has $i = \mathcal{O}\left(2^{n/2}\right)$ and $j = \mathcal{O}\left(2^{n/2}\right)$;
- $j = i + \ell$ where ℓ is the **cycle length**, i the **tail length**;
- this gives a **collision**.

Floyd's* cycle-finding algorithm

Create two chains:

- **Tortoise**: $x_i = H^i(x)$
- **Hare**: $x_{2i} = H^{2i}(y)$

Iterate until **Tortoise** = **Hare**: $x_i = x_{2i}$.

Fact

- The first i such that $x_i = x_{2i}$ is $\mathcal{O}\left(2^{n/2}\right)$.
- This i is somewhere on the cycle.

*Attributed to Floyd by Knuth, but nobody knows.

Floyd's cycle-finding algorithm

Goal: find the top of the ρ.

- i is somewhere on the cycle: $i < t + \ell$ where t is the tail and ℓ the cycle length
- $x_{2i} = x_i \implies 2i = i + k\ell \implies i = k\ell$ for some k

Create two new chains:

- $x_j = H^j(x)$ (restarting from x)
- $y_j = H^{j+2i}(x)$ (restarting from the Hare's position)

Iterate until $x_j = y_j \iff H^j(x) = H^{j+2i}(x)$

Here ***j* is the top of the ρ!**
$\implies$ retrieve the values before: $H(H^{j-1}(x)) = H(H^{j+2i-1}(x))$ is a collision.

Another loop is necessary if you're looking for the cycle length.

Summary

Input: starting point x_0
Output: a collision of H

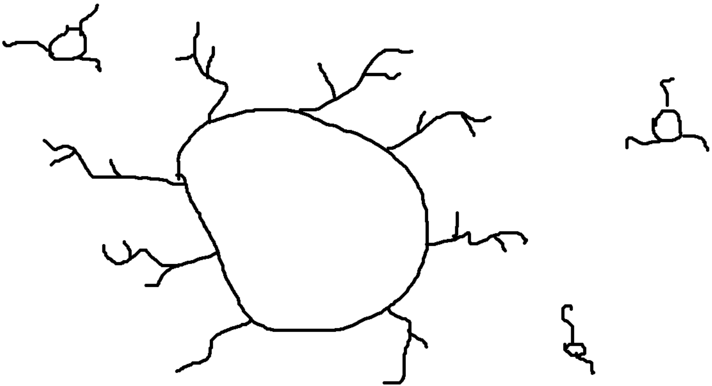
Initialize: $x \leftarrow x_0, y \leftarrow x_0$
repeat
    $x \leftarrow H(x)$, $y \leftarrow H^2(y)$
until $x = y$
Restart: $x \leftarrow x_0$
repeat
    $x' \leftarrow x, y' \leftarrow y$
    $x \leftarrow H(x)$, $y \leftarrow H(y)$
until $x = y$
return x', y'

$\mathcal{O}(2^{n/2})$ time **and small memory**.

Random Function vs. Random Permutation

The graph of a random function

$H\ :\{0,1\}^n \to \{0,1\}^n$:



- There is a large component of size $\simeq 2^{n+1}/3$: a large cycle of length $\sqrt{\pi 2^{n-3}}$, with $\mathcal{O}(2^{n/2})$ trees of size $\mathcal{O}(2^{n/2})$ attached to it
- There are $\mathcal{O}(\log n)$ small components of negligible size, with small cycles

Finding a small cycle

Some cryptanalyses require **small cycles of H** (of length $D \ll 2^{n/2}$):

- Take a random starting point
- Build a chain
- Iterate until $\geq D$ evaluations
- Restart

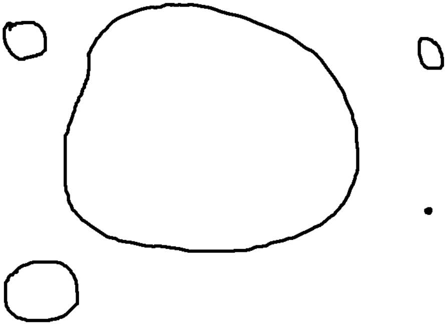
We will collide on the chain with probability $\simeq \frac{D^2}{2^n} \implies$ redo $\frac{2^n}{D^2}$ times $\implies$ total time $\mathcal{O}(2^n/D)$.

The graph of a random permutation

$\Pi : \{0,1\}^n \to \{0,1\}^n$:

The graph of a random permutation

$\Pi : \{0,1\}^n \rightarrow \{0,1\}^n$:



- There are only cycles: the largest one is of size $\mathcal{O}(2^n)$
- There are small cycles of negligible size

Distinguishing

To distinguish a random function from a random permutation, **use the Tortoise-Hare algorithm**.

- If the cycle is not found after $\mathcal{O}(2^{n/2})$ iterates, conclude that this is a permutation
- This algorithm is tight