6.857 Recitation 2: Hashing

TA: Leo de Castro

Friday February 15, 2019

Today

- Review adversary definition
- Hashing
  - One-wayness
  - Collision resistance
  - Birthday Paradox / Attack
  - Computing collisions (Floyd's 2-finger algorithm)

1 Adversaries

Let's begin by reviewing one of the most important definitions in cryptography: the adversary. A security policy cannot be complete without a definition of adversaries being considered and the adversarial behavior against which the system defends.

Definition 1 (Kerckhoffs's principle)**.** A cryptosystem should be secure even if *everything* about the system, *except the secret key*, is public knowledge.

This gives the adversary a lot of room for malicious behavior. Some adversarial behaviors that we consider are:

- Eavesdropping on messages
- Changing/mauling messages
- Storing messages
- Corrupting parties
- Impersonating parties

Here are some examples of ways an adversary may attack a voting system and the defenses the designer of the system has put in place to address these attacks.

Behavior	**Defense**
Bribing voters	No receipt for the vote
Bribing election officials	Public verifiability
Impersonating voters	Voter identification

In general, we want our systems to be secure against *advanced persistent threats* (APTs).

Definition 2. Advanced Persistent Threat An advanced persistent threat (APT) is an adversary that is

- Highly resourceful
- Has access to sophisticated methods and technology
- Substantially funded for ongoing efforts

If you can argue that your system is secure against advanced persistent threats, you're likely on the right track with your security policy!

2 Hash Functions

In lecture, we defined a primitive called a *hash function* and described some desirable properties of this function.

Definition 3. Hash Function A hash function

$$h\colon \{0,1\}^* \to \{0,1\}^d$$

is an efficiently computable function that takes in a binary string of arbitrary length and maps it to a binary string of length d.

Review of some desirable hash function properties

Definition 4. One-wayness. Given a random $y \in \{0,1\}^d$, it is computationally infeasible to find an x such that $h(x) = y$. [Note: brute force takes $O(2^d)$ in ROM.]

Definition 5. Collision resistance. It is computationally infeasible to find *any* pair x, x' such that $x \neq x'$ and $h(x) = h(x')$.

Definition 6. Targeted collision resistance. Given a target x_t, it is infeasible to find any $x \neq x_t$ such that $h(x) = h(x_t)$.

2.1 Birthday Paradox

Given $h\colon \{0,1\}^* \to \{0,1\}^d$, how long does it take to find a collision?

Theorem 7. *Birthday Paradox Given a random function $g\colon \{0,1\}^* \to \{0,1\}^d$, the time it takes to find a collision is $O(2^{d/2})$.*

Proof. Suppose we have a set of n binary strings. What is the probability that the set contains a collision?

$$\begin{aligned}\mathbb{E}[\text{\# of collisions}] &= \sum_{i,j\in[n],\ i\neq j} \mathsf{Pr}[h(x_i) = h(x_j)] = \binom{n}{2} \cdot 2^{-d} \\ &= \frac{n(n-1)}{2} 2^{-d} \approx n^2 2^{d-1}\end{aligned}$$

This is greater than 1 when $n \geq 2^{d+1/2} \approx 2^{d/2}$ $\square$

The consequence of this theorem is that hash function outputs must be twice as long as the number of bits of security you actually require (i.e. 60 bits of security requires a 120-bit output).

2.2 Computing Collisions

The brute-force method described above takes $O(2^{d/2})$ time and $O(2^{d/2})$ space. We will now give an algorithm for finding collisions that takes $O(2^{d/2})$ time and only $O(1)$ space.

Algorithm 1: Floyd's Two-Finger Cycle Detection Algorithm

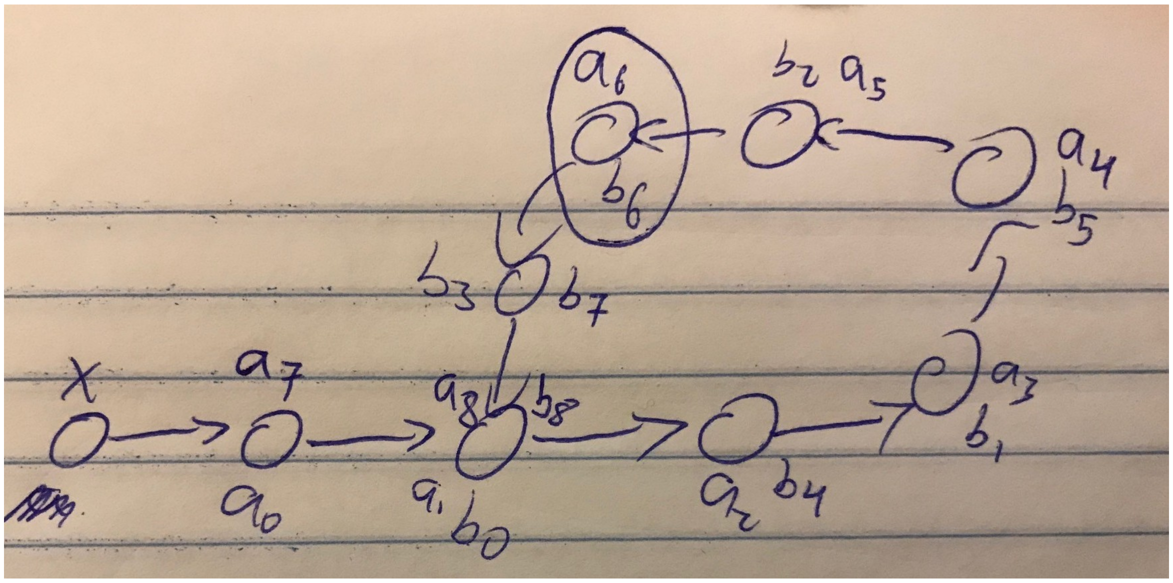
```
Pick a random value $x$.;
$a \leftarrow h(x)$ and $b \leftarrow h(h(x))$;
while $a \neq b$: do
    $a \leftarrow h(a)$;
    $b \leftarrow h(h(b))$;
end
$a \leftarrow h(x)$;
$p_a \leftarrow x$;
$p_b \leftarrow b$;
while $a \neq b$ do
    $p_a \leftarrow a$;
    $p_b \leftarrow b$;
    $a \leftarrow h(a)$;
    $b \leftarrow h(b)$;
end
return $p_a, p_b$
```

Suppose x leads to a cycle of length n, and x started t nodes away from this cycle. If after i iterations we have entered the cycle, then we will have $a = x_{(i-t) \mod n}$ and $b = x_{(2i-t) \mod n}$. When we reach an iteration j such that $j \geq t$ and $j \equiv 0 \mod n$, then we will have $a = b = x_{-t \mod n}$.

How can we use this to find a cycle?

We will start by setting a to the original x value, which is t steps away from the start of the cycle. Since $b = x_{-t \mod n}$, then if we take t steps from both these positions, we will arrive at the same node. If we remember the pre-images for these steps, then we can return these pre-images as the collision.

Below is an example run of this algorithm.



In the graph above, we have $t = 2$ and $n = 8$. We start at a value x and set $a_0 = h(x)$ and $b_0 = h(h(x))$. We set a_i and b_i to be the i^{th} values taken by a and b. The first 7 values taken are for the first `while` loop. At $a_6 = b_6$, we know a collision is found. Note that $6 \equiv -2 \mod n$. Once we found this collision, we move a back to $h(x)$ and begin stepping through one hash at a time until the collision is found, which is where the tail meets the start of the cycle. By remembering the preimages in this phase, we are able to return the colliding elements.

It is clear that the number of hashes is $O(n + t)$. But what are the expected sizes of n and t? Consider a traversal of the graph, starting at a random node. As we hash from one value to the next, we are moving from one node to the next. If we ever see a node more than once, then we have a cycle. Note that this is exactly the same as the birthday paradox above. This means that we expect $n + t$ to be $O(2^{d/2})$. Thus, the running time of Floyd's algorithm is $O(2^{d/2})$.