COMP 8920: Cryptography Winter 2025

Lecture 11 — February 11, 2025

Prof. Curtis Bright *Scribe: Aidan Bennett*

Overview

In the last lecture we introduced hash functions and defined what it meant for a hash function to be secure.

In this lecture we talk more about hash functions, and the random oracle model.

Hash functions (continued)

If a hash function is well designed, the only way to find the value $h(x)$ should be to evaluate h at x, even if many $h(x_1), h(x_2)$, etc. are already known. As an example of an h not having this property, suppose $h : \mathbb{Z}_n \times \mathbb{Z}_n \to \mathbb{Z}_n$ is the linear function $(x, y) \mapsto ax + by \mod n$. If we have $h(x_1, y_1) = z_1$ and $h(x_2, y_2) = z_2$ then for all for all $r, s \in \mathbb{Z}_n$, we have in $\mathbb{Z}_n$:

$$\begin{aligned} h(rx_1 + sx_2, ry_1 + sy_2) &= a(rx_1 + sx_2) + b(ry_1 + sy_2) \\ &= r(ax_1 + by_1) + s(ax_2 + by_2) \\ &= rh(x_1, y_1) + sh(x_2, y_2) \\ &= rz_1 + sz_2 \end{aligned}$$

So h can be evaluated at other values "directly" without calling h.

The random oracle model is a model of an ideal hash function $h : \mathcal{X} \to \mathcal{Y}$, chosen randomly from $\mathcal{F}^{\mathcal{X},\mathcal{Y}}$ (the set of functions from $\mathcal{X}$ to $\mathcal{Y}$), and we can only evaluate h via a black box "oracle".

Theorem 1. *If $h \in \mathcal{F}^{\mathcal{X},\mathcal{Y}}$ is chosen randomly, and $\mathcal{X}_0 \subseteq \mathcal{X}$ is a set of values for which $h(x)$ is known for $x \in \mathcal{X}_0$, then $\Pr[h(x) = y] = \frac{1}{|y|}$ for all $x \in \mathcal{X} \backslash \mathcal{X}_0$, and $y \in \mathcal{Y}$.*

Proof. (Theorem 5.1 in textbook) □

The reason this is true is because h is chosen uniformly at random. You can think of it like a black box that outputs a uniformly random $y \in \mathcal{Y}$ given any input x, the only constraint is that it doesn't contradict itself; i.e., if you give it the same x twice, it returns the same output, but on each new input the output is selected randomly.

Randomized algorithms can make choices; a Las Vegas algorithm is a randomized algorithm that can output "failure" with some probability, but if it gives a non-failure then its output must be correct.

An (ε, Q)-algorithm denotes an L.V. algorithm with average-case success probability ε and uses Q oracle calls of h. The success probability is over all random choices of h and x and/or y if they are part of the problem. A naive algorithm for the pre-image is seen in Algorithm 1.

Algorithm 1 Trivial algorithm for pre-image

```
Given h and a digest y, and a number of oracle calls Q
Randomly select X_0 ⊆ X of size Q.
for x ∈ X_0 do
    if h(x) = y then
        return x
    else
        return failure
    end if
end for
```

Theorem 2. *The average case probability of success for this algorithm is*

$$\varepsilon = 1 - \left(1 - \frac{1}{m}\right)^Q \qquad \textit{where } m = |\mathcal{Y}|$$

Proof. Fix $y \in \mathcal{Y}$. Let $\mathcal{X}_0 = \{x_1, \ldots, x_Q\}$ and let E_i denote the event that $h(x_i) = y$. By theorem 5.1, the E_i's are independent and $\Pr[E_i] = \frac{1}{m}$, so $\Pr[\overline{E}_i] = 1 - \frac{1}{m}$ is the probability that E_i doesn't happen. Then

$$\begin{aligned} \Pr[E_1 \vee \ldots \vee E_Q] &= 1 - \Pr[\overline{E}_1 \wedge \ldots \wedge \overline{E}_Q] \\ &= 1 - \left(1 - \frac{1}{m}\right)^Q \end{aligned}$$

□

For any fixed y the success probability is given by the above formula, so the average case success probability is this as well.

Note $1 - \left(1 - \frac{1}{m}\right)^Q \approx 1 - \left(1 - \frac{Q}{m}\right) = \frac{Q}{m}$ when Q is small compared to m.

We can give a similar algorithm for 2nd preimage in Algorithm 2.

Algorithm 2 Trivial algorithm for 2nd pre-image

```
Given h, x, Q
y ← h(x)
Select X_0 ⊆ X\{x}, with |X_0| = Q − 1
for x_0 ∈ X do
    if h(x_0) = y then
        return x_0
    end if
end for
return failure
```

This has success probability $1 - \left(1 - \frac{1}{m}\right)^{Q-1}$. Finally, the algorithm for finding any collision in Algorithm 3:

Algorithm 3 Trivial algorithm for collision finding

```
Given hash h and oracle query limit Q
Define lookup table L
for x ∈ X_0 do y_x ← h(x)
    if y_x ∈ L then
        return (x, x')                    ▷ Return x's that cause collision
    else
        L ← y_x
    end if
end for
return failure
```

The probability of success of this works out in a similar way to the "birthday paradox"; how large does a group of people need to be before there is a 50% chance that two people share the same birthday?

Theorem 3. *The chance of finding a collision with this algorithm is*

$$1 - \left(\frac{m-1}{m}\right)\left(\frac{m-2}{m}\right)\cdots\left(\frac{m-Q-1}{m}\right)$$

Proof. The chance of not finding a collision after selecting 2 distinct $x_1, x_2 \in \mathcal{X}_0$ is $\frac{m-1}{m}$ as there are m possibilities for x_2 and $m-1$ of those are failures.

However, the probability of failure when the third $x_3 \in \mathcal{X}_0$ is selected is $\frac{m-2}{m}$ as there are M choices for $h(x_3)$, and $m-2$ do not lead to a collision. So, the probability that no collision is found after Q selections of $x_1, \ldots, x_Q$ is

$$\prod_{i=0}^{Q-1} \frac{m-i}{m}$$

The chance of success is 1 minus this. □

Note $1 - x \approx e^{-x}$ for small x, so

$$\begin{aligned}\prod_{i=0}^{Q-1}\left(1 - \frac{i}{m}\right) &\approx \prod_{i=0}^{Q-1} e^{-\frac{i}{m}} \\ &= e^{-\sum_{i=0}^{Q-1}\frac{i}{m}} \\ &= e^{-\frac{Q(Q-1)}{2m}} \\ &\approx e^{-\frac{Q^2}{2m}}\end{aligned}$$

So the probability that at least one collision is found is $\varepsilon = 1 - e^{-\frac{Q^2}{2m}}$. Solving for Q, we have $-\frac{Q^2}{2m} \approx \ln(1-\varepsilon)$, or $Q \approx \sqrt{2m \ln\left(\frac{1}{1-\varepsilon}\right)}$. So for a 50% chance of success we need $Q \approx \sqrt{2\ln(2)m} \approx 1.17\sqrt{m}$.

So, you expect to query around $\sqrt{m}$ hashes before you have a 50% chance of finding a collision. Thus for a 160-bit hash, you would need to evaluate about 2^{80} times. This is easier than the preimage or the 2nd preimage problems, which need about $Q \approx \frac{m}{2}$ queries to succeed with 50% chance.