Recall: **Hash functions**

**Defn:** A hash function is a PPT function $H: \{0,1\}^* \rightarrow \{0,1\}^s$ that maps arbitrary-length bit strings into fixed-length bit strings.

The output of a hash function is called a "hash", "digest", or "fingerprint" of the input.
Recall: Collision-finding game

Let $E$ be the event that $m_0 \neq m_1$ yet $H(k, m_0) = H(k, m_1)$.

$$\text{Adv}_{\text{collision}}(A) := \Pr[E]$$
Recall: **Second-preimage resistance**
(Target pre-image resistance)

Let $E$ be the event that $m_0 \neq m_1$ yet $H(k, m_0) \neq H(k, m_1)$.

$\text{Adv}^{2\text{-preimage}}(A) := \Pr[E]$

Challenger (C)

$k \leftarrow \text{Gen}(1^s)$

Attacker (A)

$m_0, m_1 \in \{0, 1\}^*$

$m_0 \leftarrow \text{Gen}(1^s)$

$m_0 \leftarrow \text{Gen}(1^s)$
Recall: Preimage resistance (One-wayness)

Let $E$ be the event that $H(k, m) \neq y$

$\text{Adv}_{\text{preimage}}(A) := \Pr[E]$
Recall: Hash functions

Definition: A (keyed) hash function with output length $\ell(s)$ is a pair of PPT algorithms $(\text{Gen}, H)$ such that

- $\text{Gen}(1^s)$ outputs a random $s$-bit key $k \in \{0, 1\}^s$
- $H(k, x)$ outputs a fingerprint (or digest) $y \in \{0, 1\}^{\ell(|k|)}$

A cryptographic hash function is a (keyed) hash function that is (i) collision-resistant, (ii) preimage resistant, and (iii) second-preimage resistant.
The Random Oracle Model

- Many protocols use hash functions as a publicly accessible oracle that produces "random-looking" outputs
  - PRFs, PRGs, and PRPs also produce "random-looking" outputs, but they all assume the attacker doesn't know the seed/key
- Intuitively, the random oracle model is what you get when you assume (in a security proof) that the hash function produces truly random outputs
  - No function can do this, so we model it by providing the challenger and attacker with access to a random oracle
The Random Oracle Model

Challenger (C) — Random Oracle (RO) — Attacker (A)

Ryan Henry
Random oracles

Q: Why might random oracles be useful?

A: They allow us to use outputs of a hash function in places where a security definition requires uniform random values.

- If $x$ has not been queried to $H$, then $H(x)$ is uniform; thus, there is no strategy for adaptively choosing inputs to yield outputs with a desired relationship.
Extractability and programmability

- In a reduction proof, the reduction algorithm answers any random oracle queries
  - Extractability: The reduction can behave like a man-in-the-middle between the attacker and the RO
    The reduction algorithm gets to see all queries and responses
  - Programmability: The reduction algorithm can change responses from the RO
    If relationships between outputs make the reduction more powerful, the reduction algorithm can induce them

- A hash function $\Rightarrow$ no extractability or programmability
Q: What do security proofs in the random oracle model guarantee about security in the real world?

A: The answer is unclear...

- On the one hand, no hash function can implement an RO so such proofs say "nothing" about the real world.
- On the other hand, a proof of security in ROM implies (roughly) that any attack on system must exploit a weakness in the hash function.
- To date, no "real" system proven secure in the ROM has ever succumbed to attacks due to the non-randomness of the hash function.
That's all for today, folks!