How long does the output of a CRHF have to be?

Birthday attack on CRHFs. Suppose we have a hash function $H: \{0,1\}^n \to \{0,1\}^\ell$. How might we find a collision in H (without knowing anything more about H)

Approach 1: Compute $H(1), H(2), \ldots, H(2^\ell + 1)$ (size of hash output space)

↳ By Pigeonhole Principle, there must be at least one collision — runs in time $O(2^\ell)$

Approach 2: Sample $m_i \xleftarrow{R} \{0,1\}^n$ and compute $H(m_i)$. Repeat until collision is found.

How many samples needed to find a collision?

Theorem (Birthday Paradox). Take any set S where $|S| = n$. Suppose $r_1, \ldots, r_\ell \xleftarrow{R} S$. Then,

$$\Pr[\exists i \neq j : r_i = r_j] \geq 1 - e^{-\frac{\ell(\ell-1)}{2n}}$$

Proof.

$$\begin{aligned}
\Pr[\exists i \neq j : r_i = r_j] &= 1 - \Pr[\forall i \neq j : r_i \neq r_j] \\
&= 1 - \Pr[r_2 \notin \{r_1\}] \cdot \Pr[r_3 \notin \{r_2, r_1\}] \cdot \cdots \cdot \Pr[r_\ell \notin \{r_{\ell-1}, \ldots, r_1\}] \quad \text{(conditioned on } r_1, \ldots, r_{i-1} \text{ being distinct)} \\
&= 1 - \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \cdots \cdot \frac{n-\ell+1}{n} \\
&= 1 - \prod_{i=1}^{\ell-1} \left(1 - \frac{i}{n}\right) \\
&\geq 1 - \prod_{i=1}^{\ell-1} e^{-i/n} \quad \text{since } 1 + x \leq e^x \text{ for all } x \in \mathbb{R} \quad \text{(automatically holds for } x \leq -1\text{)} \\
&= 1 - e^{\sum_{i=1}^{\ell-1} -i/n} \\
&= 1 - e^{-\frac{1}{n}\sum_{i=1}^{\ell-1} i} \\
&= 1 - e^{-\frac{(\ell-1)\ell}{2n}}
\end{aligned}$$

$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{6} + \cdots$ (dominant term when $|x| < 1$; positive for all $x > 0$)

When $\ell \geq 1.2\sqrt{n}$, $\Pr[\text{collision}] = \Pr[\exists i \neq j : r_i = r_j] > \frac{1}{2}$. [For birthdays, $1.2\sqrt{365} \approx 23$] (number of people in a room to have a common birthday)

↳ Birthdays not uniformly distributed, but this only increases collision probability. [Try proving this!]

For hash functions with range $\{0,1\}^\ell$, we can use a birthday attack to find collisions in time $\sqrt{2^\ell} = 2^{\ell/2}$

↳ For 128-bit security (e.g., 2^{128}), we need the output to be 256-bits (hence SHA-256)

↳ Quantum collision-finding can be done in $2^{\ell/3}$ (cube root attack), though requires more space

can even do it with constant space! [via Floyd's cycle finding algorithm]

Back to building a secure MAC (or even better, a large-domain PRF) from a CRHF — can we do it more directly than using CRHF + small-domain MAC?

↳ Main difficulty seems to be that CRHFs are keyless but MACs are keyed

Idea: include the key as part of the hashed input

By itself, collision-resistance does not provide any "randomness" guarantees on the output

↳ For instance, if H is collision-resistant, then $H'(m) = m_0 \| \cdots \| m_{10} \| H(m)$ is also collision-resistant even though H' also leaks the first 10 bits/blocks of m

↳ Constructing a PRF/MAC from a hash function will require more than just collision resistance

- Option 1: Model hash function as an "ideal hash function" that behaves like a fixed truly random function (modeling heuristic called the random oracle model — will encounter later in this course)
- Option 2: Start with a concrete construction of a CRHF (e.g., Merkle-Damgård or the sponge construction) and reason about its properties
  ↳ We will take this approach

Suppose H is a Merkle-Damgård hash function built from a secure compression function

Several ways to build a keyed function:

1. Prepend key: $F(k, m) := H(k \| m)$
   ↳ Insecure due to structure of Merkle-Damgård: can mount an "extension attack:" given $H(k \| m)$, can compute $H(k \| m \| m')$ by extending Merkle-Damgård chain
2. Append key: $F(k,m) := H(m \| k)$
   ↳ Similar to hash-then-MAC construction and vulnerable to same offline attack: adversary finds a collision in the Merkle-Damgård prefix and uses that to construct a forgery
   ↳ Structure exploited in SHA-1 collision demonstration (can generate arbitrary collisions once prefix matches) → for SHA-1, they used PDF files
3. Envelope method: $F(k,m) := H(k \| m \| k)$
4. Two-key nest: $F((k_1, k_2), m) := H(k_2 \| H(k_1 \| m))$

} for reasonable pseudorandomness assumptions on h (e.g., both $F_1(k,m) := h(k,m)$ and $F_2(k,m) := h(m,k)$ is a PRF), both of these constructions are secure PRFs on a variable-size domain

hash-based MAC

HMAC is a PRF/MAC based on the two-key nest (though with correlated keys):

$HMAC(k, m) := H(k_1 \| H(k_2, m))$

where $k_1 \leftarrow k \oplus ipad$ and $k_2 \leftarrow k \oplus opad$

and ipad and opad are fixed strings (specified in the HMAC standard)

0x36 repeated 0x5C repeated

Security: Since k_1 and k_2 are correlated, need to make stronger assumption on security (e.g., h remains pseudorandom under a related-key attack)

Instantiations: Typically, denoted HMAC-H where H is the hash function

e.g., HMAC-SHA1

HMAC-SHA256 — one of the most widely-used MAC on the web (used in SSL/TLS, IPsec, SSH, and more)

HMAC for key-derivation: Recall that under reasonable assumptions, HMAC is a secure PRF

In many protocols, we need to derive multiple keys from a single master key (e.g., derived from a password)

↳ To derive multiple independent cryptographic keys, a PRF is a natural primitive:

$k_{enc} \leftarrow HMAC(k_{master}, \text{"enc"})$

$k_{mac} \leftarrow HMAC(k_{master}, \text{"mac"})$

} PRF security says derived keys are computationally indistinguishable from uniform

derived keys master key tag (just has to be unique)

This approach is used in TLS and IPsec to derive session keys during session setup

↳ General paradigm is the "expand" step in hash-based key-derivation (HKDF — RFC 5869)

↳ Consists of two procedures:

- Extract: derive a master key from entropy source (e.g., a user password)
- Expand: derive sub-keys from the master key

Both steps rely on HMAC