So far, we have focused on constructing a large-domain PRF from a small-domain PRF in order to construct a MAC on long messages

↳ Alternative approach: "compress" the message itself (e.g., "hash" the message) and MAC the compressed representation

Still require unforgeability: two messages should not hash to the same value [otherwise trivial attack: if $H(m_1) = H(m_2)$, then MAC on m_1 is also MAC on m_2]

↳ counter-intuitive: if hash value is shorter than messages, collisions always exist — so we can only require that they are hard to find

Definition. A hash function $H : \mathcal{M} \to \mathcal{T}$ is collision-resistant if for efficient adversaries $\mathcal{A}$,

$$\mathrm{CRHFAdv}[\mathcal{A}, H] = \Pr[(m_0, m_1) \leftarrow \mathcal{A} : H(m_0) = H(m_1)] = \mathrm{negl.}$$

As stated, definition is problematic: if $|\mathcal{M}| > |\mathcal{T}|$, then there always exists a collision m_0^*, m_1^* so consider the adversary that has m_0^*, m_1^* hard coded and outputs m_0^*, m_1^*

↳ Thus, some adversary always exists (even if we may not be able to write it down explicitly)

↳ Formally, we model the hash function as being parameterized by an additional parameter (e.g., a "system parameter" or a "key") so adversary cannot output a hard-coded collision

↳ In practice, we have a concrete function (e.g., SHA-256) that does not include security or system parameters

↳ believed to be hard to find a collision even though there are infinitely-many (SHA-256 can take inputs of arbitrary length)

MAC from CRHFs: Suppose we have the following

- A MAC (Sign, Verify) with key-space $\mathcal{K}$, message space $\mathcal{M}_0$ and tag space $\mathcal{T}$ [e.g., $\mathcal{M}_0 = \{0,1\}^{256}$, $\mathcal{M}_1 = \{0,1\}^*$]
- A collision-resistant hash function $H : \mathcal{M}_1 \to \mathcal{M}_0$

Define $S'(k, m) = S(k, H(m))$ and

$V'(k, m, t) = V(k, H(m), t)$

Theorem. Suppose $\Pi_{MAC} = (\mathrm{Sign}, \mathrm{Verify})$ is a secure MAC and H is a CRHF. Then, Π'_{MAC} is a secure MAC. Specifically, for every efficient adversary $\mathcal{A}$, there exist efficient adversaries $\mathcal{B}_0$ and $\mathcal{B}_1$ such that

$$\mathrm{MACAdv}[\mathcal{A}, \Pi'_{MAC}] \leq \mathrm{MACAdv}[\mathcal{B}_0, \Pi_{MAC}] + \mathrm{CRHFAdv}[\mathcal{B}_1, H]$$

__Proof Idea.__ Suppose A manages to produce a valid forgery t on a message m. Then, it must be the case that

- t is a valid MAC on $H(m)$ under Π_{MAC}
- If A queries the signing oracle on $m' \neq m$ where $H(m') = H(m)$, then A breaks collision-resistance of H
- If A never queries signing oracle on m' where $H(m') = H(m)$, then it has never seen a MAC on $H(m)$ under Π_{MAC}. Thus, A breaks security of Π_{MAC}.

[See Boneh-Shoup for formal argument — very similar to above: just introduce event for collision occurring vs. not occurring]

Constructing above is simple and elegant, but __not__ used in practice

- __Disadvantage 1:__ Implementation requires both a secure MAC __and__ a secure CRHF: more complex, need __multiple__ software/hardware implementations
- __Disadvantage 2:__ CRHF is a __key-less__ object and collision finding is an offline attack (does not need to query verification oracle). Adversary with substantial __preprocessing__ power can compromise collision-resistance (especially if hash size is small)

__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 $H(1), H(2), \ldots, H(2^\ell + 1)$

↳ By Pigeonhole Principle, there must be at least __one__ collision — runs in time $O(2^\ell)$ — size of hash output space

__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.__

$$\begin{aligned}
\Pr[\exists i \neq j : r_i = r_j] &= 1 - \Pr[\forall i \neq j : r_i \neq r_j] \\
&= 1 - \Pr[r_2 \notin \{r_1\}] \cdot \Pr[r_3 \notin \{r_2, r_1\}] \cdot \cdots \cdot \Pr[r_\ell \notin \{r_{\ell-1}, \ldots, r_1\}] \\
&= 1 - \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \cdots \cdot \frac{n-\ell+1}{n} \\
&= 1 - \prod_{i=1}^{\ell-1} \left(1 - \frac{i}{n}\right) \\
&\geq 1 - \prod_{i=1}^{\ell-1} e^{-i/n} \quad \text{since } 1 + x \leq e^x \text{ for all } x \in \mathbb{R} \quad \Big[e^x = 1 + x + \underbrace{\frac{x^2}{2} + \frac{x^3}{6} + \cdots}_{\text{positive for all } x > 0} \Big] \\
&= 1 - e^{\sum_{i=1}^{\ell-1} -i/n} = 1 - e^{-\frac{1}{n}\sum_{i=1}^{\ell-1} i} \\
&= 1 - e^{-\frac{(\ell-1)\ell}{2n}}
\end{aligned}$$

automatically holds for $x \leq -1$

dominant term when $|x| < 1$

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 $\sqrt{2^\ell} = 2^{\ell/2}$ — can even do it with __constant__ space! [via Floyd's cycle finding algorithm]

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

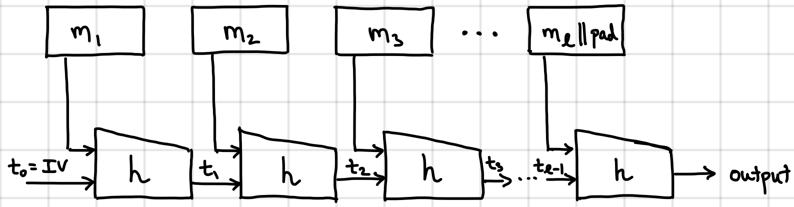
↳ Quantum collision-finding can be done in $2^{\ell/3}$ (cube root attack), though requires more space

$\nearrow$ HMAC (most widely used MAC)

So how do we use hash functions to obtain a secure MAC? Will revisit after studying constructions of CRHFs.

Many cryptographic hash functions (e.g., MD5, SHA-1, SHA-256) follow the Merkle-Damgård paradigm: start from hash function on short messages and use it to build a collision-resistant hash function on a long message:

1. Split message into blocks
2. Iteratively apply compression function (hash function on short inputs) to message blocks



h: compression function

$t_0, \dots, t_\ell$: chaining variables

padding introduced so last block is multiple of block size

$\hookrightarrow$ must also include an encoding of the message length: typically of the form $100\cdots0 \| \langle s \rangle$ where $\langle s \rangle$ is a fixed-length binary representation of message length in blocks

Recall: $100\cdots0$ padding was used in the ANSI standard

if not enough space to include the length, then extra block is added (similar to CBC encryption)

Hash functions are deterministic, so IV is a fixed string (defined in the specification) — can be taken to be all-zeroes string, but usually set to a custom value in constructions

for SHA-256:
$\mathcal{X} = \{0,1\}^{256} = \mathcal{Y}$

Theorem. Suppose $h: \mathcal{X} \times \mathcal{Y} \to \mathcal{X}$ be a compression function. Let $H: \mathcal{Y}^{\leq \ell} \to \mathcal{X}$ be the Merkle-Damgård hash function constructed from h. Then, if h is collision-resistant, H is also collision-resistant.

Proof. Suppose we have a collision-finding algorithm A for H. We use A to build a collision-finding algorithm for h:

1. Run A to obtain a collision M and M' ($H(M) = H(M')$ and $M \neq M'$).
2. Let $M = m_1 m_2 \cdots m_u$ and $M' = m_1' m_2' \cdots m_v'$ be the blocks of M and M', respectively. Let $t_0, t_1, \dots, t_u$ and $t_1' t_2' \cdots t_v'$ be the corresponding chaining variables.
3. Since $H(M) = H(M')$, it must be the case that
$$H(M) = h(t_{u-1}, m_u) = h(t_{v-1}', m_v') = H(M')$$
If either $t_{u-1} \neq t_{v-1}'$ or $m_u \neq m_v'$, then we have a collision for h.
Otherwise, $m_u = m_v'$ and $t_{u-1} = t_{v-1}'$. Since m_u and m_v' include an encoding of the length of M and M', it must be the case that $u = v$. Now, consider the second-to-last block in the construction (with output $t_{u-1} = t_{u-1}'$):
$$t_{u-1} = h(t_{u-2}, m_{u-1}) = h(t_{u-2}', m_{u-1}') = t_{u-1}'$$
Either we have a collision or $t_{u-2} = t_{u-2}'$ and $m_{u-1} = m_{u-1}'$. Repeat down the chain until we have collision or we have concluded that $m_i = m_i'$ for all i, and so $M = M'$, which is a contradiction.

Note: Above constructing is sequential. Easy to adapt construction (using a tree) to obtain a parallelizable construction.