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 $H: M \to T$ is collision-resistant if for efficient adversaries A,
$$CRHFAdv[A,H] = Pr[(m_0, m_1) \leftarrow A : H(m_0) = H(m_1)] = negl.$$

As stated, definition is problematic: if $|M| > |T|$, 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 K, message space M_0 and tag space T $\left[e.g., \begin{array}{l} M_0 = \{0,1\}^{256} \\ M_1 = \{0,1\}^* \end{array} \right]$
    − A collision-resistant hash function $H: M_1 \to M_0$
Define $S'(k,m) = S(k, H(m))$ and
        $V'(k, m, t) = V(k, H(m), t)$

__Theorem.__ Suppose $\Pi_{MAC} = (Sign, Verify)$ is a secure MAC and H is a CRHF. Then, Π'_{MAC} is a secure MAC. Specifically, for every efficient adversary A, there exist efficient adversaries B_0 and B_1 such that
$$MACAdv[A, \Pi'_{MAC}] \leq MACAdv[B_0, \Pi_{MAC}] + CRHFAdv[B_1, H]$$

<u>Proof Idea</u>. 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 <u>not</u> used in practice
- <u>Disadvantage 1</u>: Implementation requires both a secure MAC <u>and</u> a secure CRHF: more complex, need <u>multiple</u> software/hardware implementations
- <u>Disadvantage 2</u>: CRHF is a <u>key-less</u> object and collision-finding is an offline attack (does not need to query verification oracle) Adversary with substantial <u>preprocessing</u> power can compromise collision-resistance (especially if hash size is small)

<u>Birthday attack on CRHFs</u>. 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)

<u>Approach 1</u>: Compute $H(1), H(2), \ldots, H(2^\ell + 1)$ size of hash output space
    ↳ By Pigeonhole Principle, there must be at least <u>one</u> collision — runs in time $O(2^\ell)$
<u>Approach 2</u>: 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?

<u>Theorem (Birthday Paradox)</u>. Take any set S where $|S| = n$. Suppose $r_1, \ldots, r_\ell \xleftarrow{R} S$. Then,
$$\Pr[\exists i \neq j : r_i = r_j] \geq 1 - e^{-\frac{\ell(\ell-1)}{2n}}$$

<u>Proof</u>. $\Pr[\exists i \neq j : r_i = r_j] = 1 - \Pr[\forall i \neq j : r_i \neq r_j]$
$= 1 - \Pr[r_2 \notin \{r_1\}] \cdot \Pr[r_3 \notin \{r_2, r_1\}] \cdot \cdots \cdot \Pr[r_\ell \notin \{r_{\ell-1}, \ldots, r_1\}]$
$= 1 - \frac{n-1}{n} \cdot \frac{n-2}{n} \cdot \cdots \cdot \frac{n-\ell+1}{n}$
$= 1 - \prod_{i=1}^{\ell-1} \left(1 - \frac{i}{n}\right)$
$\geq 1 - \prod_{i=1}^{\ell-1} e^{-i/n}$ since $1-x \leq e^{-x}$ $\left(\text{for } x \in [0,1], \text{ can show via Taylor expansion: } e^{-x} = 1 - x + \frac{x^2}{2} - \frac{x^3}{6} + \cdots \atop \geq 1-x\right)$

dominant term ↓ when $|x| \ll 1$

$= 1 - e^{\sum_{i=1}^{\ell-1} -i/n} = 1 - e^{-\frac{1}{n}\sum_{i=1}^{\ell-1} i}$
$= 1 - e^{-\frac{(\ell-1)\ell}{2n}}$

                    number of people in a room
                    to have a common birthday
When $\ell \geq 1.2\sqrt{n}$, $\Pr[\text{collision}] = \Pr[\exists i \neq j : r_i = r_j] > \frac{1}{2}$. [For birthdays, $1.2\sqrt{365} \approx 23$]
                                    ↳ Birthdays not uniformly distributed, but this only <u>increases</u> 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 <u>constant space</u>!
  ↳ For 128-bit security (e.g., 2^{128}), we need the output to be 256-bits (hence <u>SHA-256</u>)
 ↳ Quantum collision-finding can be done in $2^{\ell/3}$ (cube root attack), though requires more space   [via Floyd's cycle finding algorithm]

$\rightarrow$ 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 <u>short</u> messages and use it to build a collision-resistant hash function on a long message:

1. Split message into blocks

2. Iteratively apply <u>compression function</u> (hash function on short inputs) to message blocks
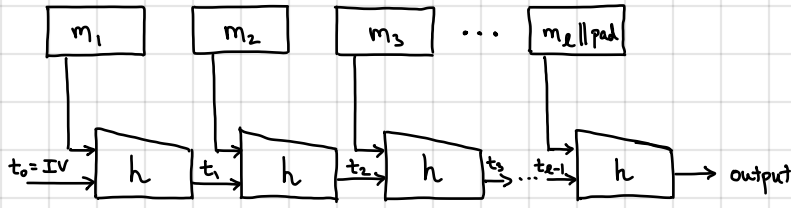
h : compression function

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

padding introduced so last block is multiple of block size

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

Hash functions are <u>deterministic</u>, 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

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

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

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

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

<u>Proof.</u> 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 $t_0, t_1, ..., t_u$ 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.

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

Sufficient now to construct a __compression function__.

Typical approach is to use a block cipher.

__Davies-Meyer__: Let $F: K \times X \to X$ be a block cipher. The Davies-Meyer compression function $h: K \times X \to X$ is then

$m_i \in K$
$t_{i-1} \in X \to \boxed{F} \to \oplus \to t_i \in X$
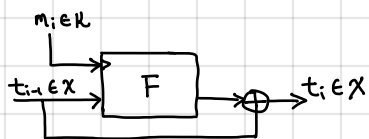
$h(k, x) := F(k, x) \oplus x$

Many other variants also possible: $h(k, x) = F(k, x) \oplus k \oplus x$
[used in Whirlpool hash family]

Need to be careful with design!
- $h(k, x) = F(k, x)$ is __not__ collision-resistant: $h(k, x) = h(k', F^{-1}(k', F(k, x)))$
- $h(k, x) = F(k, x) \oplus k$ is __not__ collision-resistant: $h(k, x) = h(k', F^{-1}(k', F(k, x) \oplus k \oplus k'))$

__Theorem.__ If we model F as an ideal block cipher (ie, a truly random permutation for every choice of key), then Davies-Meyer is collision-resistant.

→ birthday attack run-time: $\sim 2^{80}$
attack ran in time $\sim 2^{64}$ (100,000× faster)

January, 2020: chosen-prefix collision in $\sim 2^{63.4}$ time!

__Conclusion:__ Block cipher + Davies-Meyer + Merkle-Damgård $\implies$ CRHFs
Examples: SHA-1: SHACAL-1 block cipher with Davies-Meyer + Merkle-Damgård ← no longer secure [first collision found in 2017!]
SHA-256: SHACAL-2 block cipher with Davies-Meyer + Merkle-Damgård

- SHA-1 extensively used (eg, git, svn, software updates, PGP/GPG signatures, certificates) → attacks show need to transition to SHA-2 or SHA-3

Why not use AES?
- Block size too small! AES outputs are 128-bits, not 256 bits (so birthday attack finds collision in 2^{64} time)
- Short keys means small number of message bits processed per iteration.
- Typically, block cipher designed to be fast when using __same__ key to encrypt many messages
  ↳ In Merkle-Damgård, __different__ keys are used, so alternate design preferred (AES key schedule is __expensive__)

__Recently:__ SHA-3 family of hash functions standardized (2015)
↳ Relies on different underlying structure ("sponge" function)
↳ Both SHA-2 and SHA-3 are believed to be secure (most systems use SHA-2 — typically much faster)

or even better, a large-domain PRF

Back to building a secure MAC 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)
- __Option 2:__ Start with a concrete construction of a CRHF (eg., Merkle-Damgård or the sponge construction)
  and reason about its properties
  ↳ We will take this approach