CS 442
Introduction to Cryptography

Lecture 18: Hash Functions

Instructor: Aarushi Goel

Spring 2026

Agenda

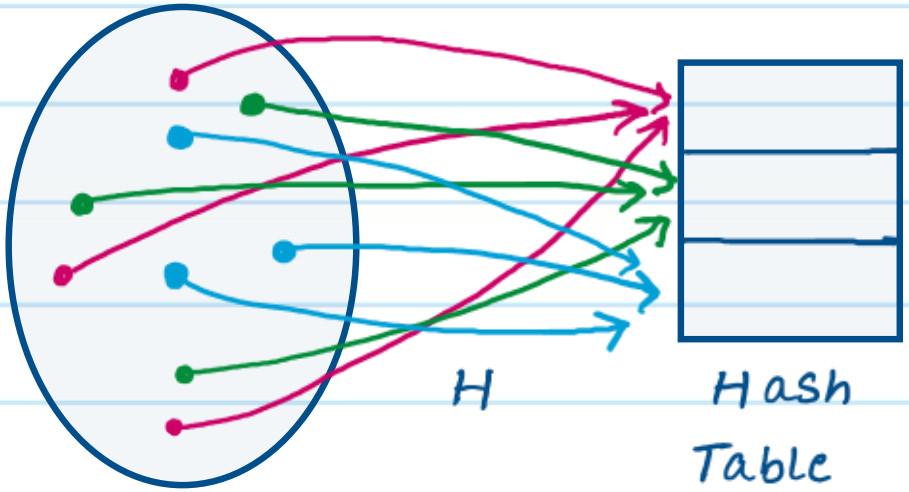
* Collision-resistant hash functions
* Applications.

* HW3 is due on April 5

Hash Functions

* Functions that take inputs of arbitrary length and compress them into short, fixed length outputs.
* $H: \{0,1\}^* \rightarrow \{0,1\}^n$

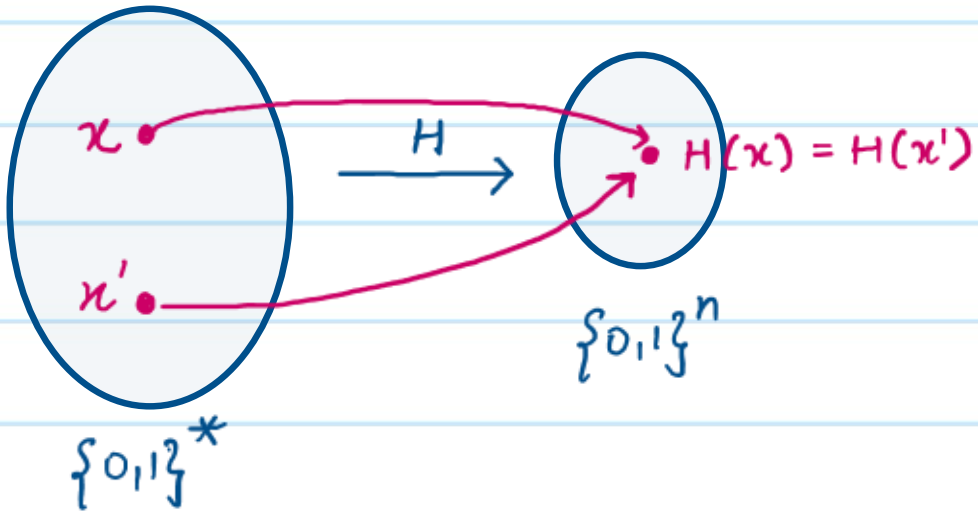
  ($\{0,1\}^*$: arbitrary length)
* Traditional use of hash functions was in data structures to enable faster lookup time when storing a set of elements.



To retrieve an element x it suffices to compute H(x) and probe that row in the hash table to look for x. This is more efficient than looking up x in the original list.

Collisions in Hash Functions

* A hash function naturally has *collisions*, where a collision is a pair of distinct elements x and x', such that $H(x) = H(x')$.
* A good hash function yields few collisions, because when a collision happens, two elements end up in the same row in the hash table, increasing the lookup time.



Collision - Resistant Hash Functions

* Collision-resistant hash functions (CRHF) are cryptographic versions of hash functions, where the goal is not just to minimize the number of collisions, but it should also be infeasible for any probabilistic polynomial-time algorithm to find a collision in H.
* Note that CRHFs are still compression functions, so collisions exist, but we just want such collisions to be hard to find.
* We typically consider keyed CRHFs.

$$H^K(x) = H(K, x)$$

unlike all other primitives we have discussed so far, here we don't care about hiding the key.

* It should be hard to find a collision in H^K, for randomly chosen K, even given K.

Collision-Resistant Hash Functions

Definition: A family of functions $H = \{h_k : D_k \to R_k\}_{k \in \mathcal{K}}$ is a collision-resistant hash function family (CRHF) if:

* Easy to Sample: There exists a PPT Gen, such that $k \leftarrow Gen(n)$, $k \in \mathcal{K}$.
* Compression: $|R_k| < |D_k|$.
* Easy to Evaluate: There exists a polynomial-time algorithm Eval, such that, given $x \in D_k$, $k \in \mathcal{K}$, $Eval(k, x) = h_k(x)$.
* Collision Resistance: For all non-uniform PPT adversaries A, there exists a negligible function $\mu(\cdot)$, such that

$$Pr\left[\begin{array}{l|l} x \neq x' \text{ and } & k \xleftarrow{\$} Gen(n), \\ h_k(x) = h_k(x') & (x, x') \leftarrow A(k, n) \end{array}\right] \leq \mu(n).$$

Hash Functions: Theory vs Practice

Theoretical Constructions

$\{H_K\}_{K \in \{0,1\}^n}$

* typically keyed functions
* there is a security parameter n.
* Security proved under well-studied computational hardness assumptions

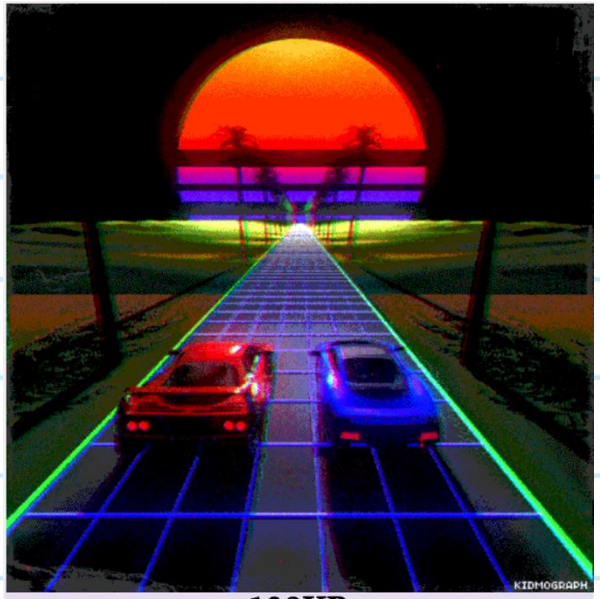
Practical Constructions

eg SHA3, SHA256, ~~SHA1~~, ~~MD5~~, ~~MD4~~ etc

broken

* unkeyed, so single functions, not a family of functions
* No security parameter knob
* Use until someone finds a collision.

Example: Collisions in MD5



139KB

MD5

128 bits

5c827c0eba9cfaa647c1a489bea77c60

139KB

MD5

128 bits

5c827c0eba9cfaa647c1a489bea77c60

Generic Attacks on Hash Functions

* Let $H: \{0,1\}^* \to \{0,1\}^n$ (we drop the key here, and assume it is implicit).
* **Trivial Attack for finding collisions:** Evaluate H on 2^n+1 distinct inputs. By pigeonhole principle, two of the outputs must be equal. This attack takes $O(2^n)$ time. Can we do better?
* **Birthday Attack:** If there are q students in class and assuming each student's birthday is a random $y_i \xleftarrow{\$} [365]$. What is the probability of collision?

  $q = 366 \Rightarrow$ prob $= 1$

  $q = 23 \Rightarrow$ prob $= 1/2$

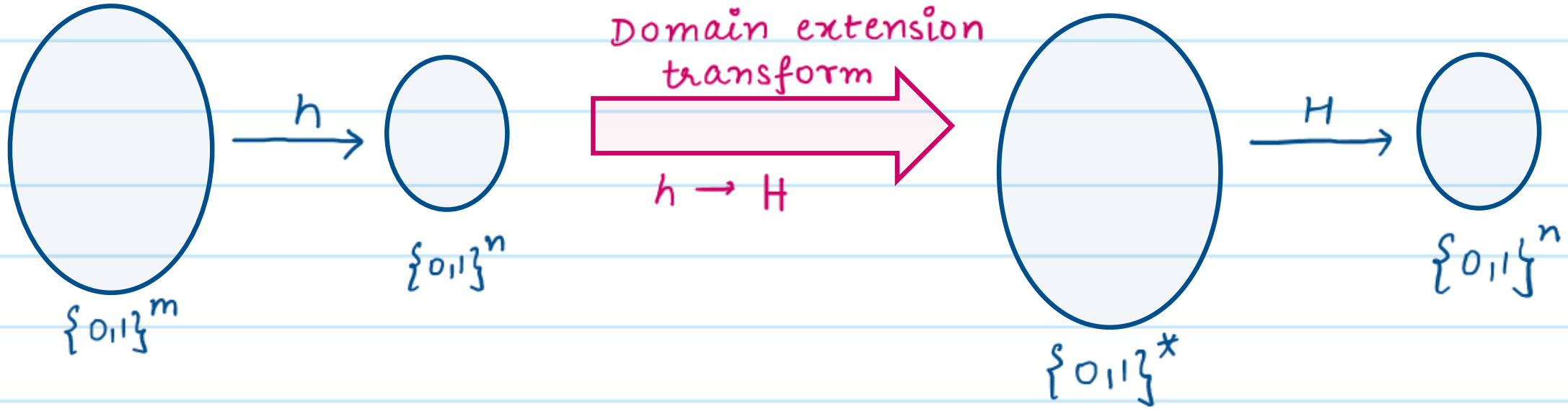
  $q = 70 \Rightarrow$ prob $= 99.9\%$

- In general, birthday analysis shows that for $y_1, \ldots, y_q$ chosen uniformly from $[N]$, the probability of collision is approx $1/2$, when $q \approx \sqrt{N}$.

* In our setting, this means that when the hash function output is on length n (i.e., the range is of size $N = 2^n$), if we evaluate on $q = 2^{n/2}$ distinct inputs, we will see a collision with probability $1/2$.
* From concrete efficiency perspective, for a hash function to resist collision-finding attacks that run in time T, the output length of the hash function must be at least $2 \log T$ bits (i.e., range is of size $2^{2 \log T} = T^2$).
* For $T = 2^{128}$, we need output length of hash function to be at least 256 bits.

Domain Extension

* Hash functions are often constructed by first designing a CRHF that handles fixed-length inputs, and then using domain extension to handle arbitrary-length inputs.



* Merkle-Damgård transform is an example of domain extension.

Merkle - Damgård Transform

* Let $h_K : \{0,1\}^{2n} \to \{0,1\}^n$ be a CRHF
* We can design a CRHF $H_K : \{0,1\}^* \to \{0,1\}^n$ as follows:

1. Let $x \in \{0,1\}^L$ be an L-bit input that we want to hash using H.
2. Parse $x = x_1, \ldots, x_B$, where $B = \frac{L}{n}$ and each $x_i \in \{0,1\}^n$.
   If L is not divisible by n, we pad x with appropriately many 0s.
3. Set $IV = 0^n$ initialization vector.

Claim: If h_K is a CRHF, then so is H_K.

Proof:

Let us assume for the sake of contradiction that it is easy to find x, x', such that $H_K(x) = H_K(x')$. We will show that it must then also be easy to find a collision in h_K, to arrive at a contradiction.

Case1: $|x| \neq |x'|$: Let $x \in \{0,1\}^L$ and $x' \in \{0,1\}^{L'}$. If $H_K(x) = H_K(x')$, then it must be the case that $h_K(., L) = h_K(., L')$. This is a collision on h_K. This means, an adversary who can find collision in H_K, can also find a collision in h_K. But since h_K is a CRHF, it should be hard for any PPT adversary to find such collisions. Hence, it must also be hard for any PPT adversary to find collisions in H_K.

Case 2: $|x| = |x'|$: Let $x, x' \in \{0,1\}^L$. Parse $x = x_1, \ldots, x_B$ & $x' = x'_1 \ldots x'_B$.

Since $x \neq x'$, there must exist at least one $i \in [B]$, such that $x_i \neq x_i'$. Let's pick the largest such i.

if $H_K(x) = H_K(x')$, then it must be the case that $h_K(., x_i) = h_K(., x_i')$. This is a valid collision in h_K. . But since h_K is a CRHF, it should be hard for any PPT adversary to find such collisions. Hence, it must also be hard for any PPT adversary to find collisions in H_K.

$\Rightarrow$ H_K is a CRHF.

Applications of CRHFs

* **Password Authentication:** Instead of storing plaintext passwords on servers, websites can store a hash of the passwords instead. This ensures that the passwords are not compromised even if the server is compromised.

* **Deduplicate Files:** To avoid storing duplicate copies of files, we can use hashes of files as identifiers. We can hash two files to produce identifiers h_1 and h_2. If $h_1 \neq h_2$, this implies $D_1 \neq D_2$. If $h_1 = h_2$, it almost always implies file D_1 = file D_2.

* HMAC: Hash functions can be used as MACs.

  $MAC_K(m) = H(K \| m)$. It is computationally difficult to find another $K \| m'$ that produces the same hash.

* However, when using a Merkle-Damgård based hash, an adversary could potentially attach some additional s to m to get $m' = m \| s$ such that they can easily compute $tag' = H(tag \| s)$

* Therefore, in practice, we use a nested MAC like

$$MAC_K(m) = H(K \| H(K \| m)).$$

* Hash-and-MAC: If we have a MAC scheme for signing fixed length messages, and want to use it for signing arbitrary-length messages, we can hash the message to appropriate length digest and then sign the digest.