Hash Functions

Federico Pintore [1]

[1] Mathematical Institute,
University of Oxford

Outline

Outline

Introduction

- Informally speaking, hash functions take long bit strings and output shorter bit strings, called *digests*.
- They are used almost everywhere in Cryptography.
- If you *imagine* that hash functions are truly random (modelled as *random oracles*), then proving the security of some cryptographic schemes becomes achievable (e.g. RSA-OAEP).
- A debate/controversy over the soundness of the random oracle model.

Keyed Hash Functions - A Definition

Definition

A keyed hash function with output length $\ell(n)$ *consists of two PPT algorithms* (KeyGen, H)*, defined as follows:*

- $\text{KeyGen}(1^n)$: *it takes a security parameter* n *and outputs a key* s*.*
- $H(s, x \in \{0,1\}^*)$: *it takes a key* s *and a string* $x \in \{0,1\}^*$*, and outputs a string* $H^s(x) \in \{0,1\}^{\ell(n)}$

If H is defined only for inputs $x \in \{0,1\}^{\ell'(n)}$, then the keyed hash function is said fixed-length. We consider only compression functions, i.e. $\ell'(n) > \ell(n)$.

Security Notions - Collision Resistance

- A keyed hash function determines a keyed function

$$H : \text{KeySet} \times \text{InSet} \to \text{OutSet}$$

  where InSet is $\{0,1\}^*$ or $\{0,1\}^{\ell'(n)}$, and OutSet $= \{0,1\}^{\ell(n)}$.

- We use the notation $H^s(x) := H(s,x)$.

- Given a key s, it should be infeasible for any PPT algorithm to find $x \neq x'$ s.t. $H^s(x) = H^s(x')$ (i.e., a collision).

- Since the domain is larger than its range, collisions always exist, but we want them to be hard to find.

- This time the key is not a secret, i.e. collision resistance should hold even when the key s is in the adversary's hands.

Collision Resistance

Given a keyed hash function (KeyGen, H), an adversary $\mathcal{A}$, and a security parameter n, we define the collision-finding experiment $\text{Hash}^{coll}_{\mathcal{A},H}(n)$ as follows:

- A key s is generated by KeyGen and is given to $\mathcal{A}$.
- Adversary's output: two strings x and x'.
- Experiment's output: 1 iff $x \neq x'$ and $H^s(x) = H^s(x')$

Definition

A keyed hash function (KeyGen, H) *is collision resistant if for all PPT adversaries* $\mathcal{A}$ *we have*

$$\Pr[\text{Hash}^{coll}_{\mathcal{A},H}(n) = 1] \leq \text{negl}(n)$$

Hash Functions in Practice

- They are *unkeyed*.
- What's the reason of using keyed functions?
- Theoretically speaking, you can always output a collision using a constant-time algorithm (a colliding pair - hardcoded in the algorithm itself - is output).
- It is impossible to hardcode a colliding pair for every possible key.
- However, colliding pairs are unknown and computationally hard to find for hash functions used in practice.

Weaker Security Notions

- *Second-preimage* or *target-collision resistance*: given s and a uniform x, it is infeasible for any PPT adversary to find x' s.t. $x \neq x'$ and yet $H^s(x) = H^s(x')$
- *Preimage resistance* or *one-wayness*: given s and a uniform y, it is infeasible for any PPT adversary to find x s.t. $H^s(x) = y$

Note that:

collision resist. $\Rightarrow$ second preimage resist. $\Rightarrow$ preimage resist.

Outline

How to Design a Hash Function?

- **First**, consider a collision-resistant, fixed-length hash function.
- **Second**, apply a domain extension method to deal with arbitrary-length inputs.
- This should maintain the collision-resistance property.
- Merkle-Damgård transform is a very famous approach for domain extension.
- It has been used for MD5 and the SHA family.
- Theoretical implication of Merkle-Damgård transform: if you can compress by a single bit, then you can compress by an arbitrary amount of bits!

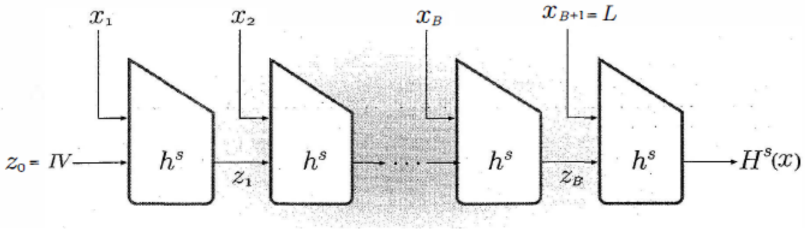
The Merkle-Damgård Transform

Given a fixed-length hash function (KeyGen, h), with input lenght $2n$ and output length n, we construct an arbitrary-length hash function (KeyGen, H) as follows:

- KeyGen : it remains unchanged.
- H : it takes a key s and a string $x \in \{0,1\}^*$ of length $L < 2^n$, and does the following:
    - Pad x with zeros to get a bit string of length $B \cdot n$. Consider the n-bit blocks $x_1, \cdots, x_B$ and set $x_{B+1} \leftarrow L$, where L is encoded as an n-bit string.
    - Set $z_0 \leftarrow 0^n$ (also called IV)
    - Compute $z_i \leftarrow h^s(z_{i-1}||x_i)$, for $i = 1, \cdots, B+1$.
    - Output z_{B+1}.

The Merkle-Damgård Transform

[Katz-Lindell]



Theorem

If (KeyGen, h) *is collision-resistant, then so is* (KeyGen, H).

The Merkle-Damgård Transform

Proof.

We show that a collision in H^s leads to a collision in h^s.

Let $x \neq x'$ of length L and L' s.t. $H^s(x) = H^s(x')$.

We pad x and x' to get $x_1, \cdots, x_B, x_{B+1}$ and $x'_1, \cdots, x'_{B'}, x'_{B'+1}$, where $x_{B+1} = L$ and $x'_{B'+1} = L'$.

- $L \neq L'$: then $H^s(x) = z_{B+1} = h^s(z_B, L) = h^s(z'_{B'}, L') = z'_{B'+1} = H^s(x')$. Hence $z_B||L \neq z'_{B'}||L'$ is a collision for h^s.
- $L = L'$: in this case $B = B'$. Consider $I_i = z_{i-1}||x_i$ and $I'_i = z'_{i-1}||x'_i$ for $i = 1, \ldots, B+2$, where $I_{B+2} = z_{B+1} = z'_{B+1} = I'_{B+2}$. Let N be the largest integer for which $I_N \neq I'_N$ (which exists since $x \neq x'$). Note that $N \leq B+1$.

  $I_{N+1} = z_N||x_{N+1} = z'_N||x'_{N+1} = I'_{N+1} \Rightarrow h^s(I_N) = z_N = z'_N = h^s(I'_N)$.

□

Outline

MAC using Hash Functions

- We present a different approach to construct a MAC for arbitrary-length messages.
- The idea is simple and widely used in practice (e.g. HMAC).
- **Firstly**, use a collision-resistant hash function (KeyGen, H) to hash an arbitrary-long message m down to a fixed-length string $H^s(m)$.
- **Secondly**, apply a fixed-length MAC to $H^s(m)$.

Hash-and-MAC

Given a fixed-length MAC $S_{mac} = (\text{Mac}, \text{Verify})$ for messages of length $\ell(n)$, and a hash function (KeyGen, H) with output length $\ell(n)$, we define a new MAC

$$S'_{mac} = (\text{KeyGen}', \text{Mac}', \text{Verify}')$$

for arbitrary-length messages as follows.

- $\text{KeyGen}'(1^n)$: it takes a security parameter n, and outputs a uniform key $k \in \{0,1\}^n$ and runs the key generator of the hash function to get s. The final key is (k, s).
- $\text{Mac}'((k,s), m \in \{0,1\}^*)$: it outputs $t \leftarrow \text{Mac}_k(H^s(m))$.
- $\text{Verify}'((k,s), m \in \{0,1\}^*, t)$: it outputs 1 iff $\text{Verify}_k(H^s(m), t) = 1$.

HMAC

- The idea is to build a secure MAC for arbitrary-length messages **directly** from a hash function.
- What about defining $\mathsf{Mac}_k(m) = H^s(k||m)$?

HMAC

- The idea is to build a secure MAC for arbitrary-length messages **directly** from a hash function.
- What about defining $\mathsf{Mac}_k(m) = H^s(k||m)$?
- It is NOT secure (if H was constructed using the Merkle-Damgård transform).
- HMAC is a standardised secure MAC that uses two layers of hashing.

HMAC

Let h be a fixed-length hash function with input length $n + n'$ and output length n. Let H be the hash function obtained from applying the Merkle-Damgård transform on h. Let opad and ipad be two fixed strings of length n'.

We define a MAC for arbitrary-length messages as follows:

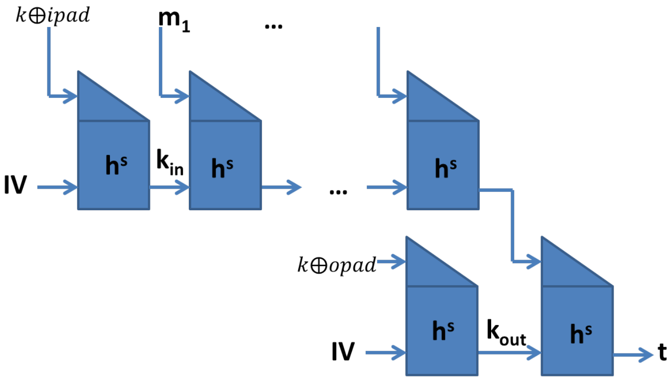
- KeyGen(n): it runs the key generator of the hash function H to get a key s. It also chooses a uniform $k \in \{0,1\}^{n'}$. It outputs the key (s, k).
- Mac$((s,k), m \in \{0,1\}^*)$: it outputs

$$t \leftarrow H^s\big((k \oplus \text{opad}) || H^s((k \oplus \text{ipad}) || m)\big)$$

- Verify$((s,k), m \in \{0,1\}^*, t)$: outputs 1 iff

$$t \stackrel{?}{=} H^s\big((k \oplus \text{opad}) || H^s((k \oplus \text{ipad}) || m)\big)$$

HMAC



We are assuming $n + \ell < n'$ (the length of the message is encoded as a ℓ-bit string).

Analysis of HMAC

- HMAC can be viewed as an instantiation of the hash-and-MAC technique.
- HMAC is very efficient and widely used in practice.

Analysis of HMAC

- HMAC can be viewed as an instantiation of the hash-and-MAC technique.
- HMAC is very efficient and widely used in practice.
- The use of the key in the inner computation allows for hash functions satisfying a weaker assumption to be used, namely hash functions that are *weakly* collision resistant (in this case, the adversary has access to a hash oracle to $H^s_{k_{in}}()$, where k_{in} is a secret value that replaces IV in the Merkle-Damgaard transform).
- Independent keys should be used in the inner and outer computations
- For efficiency reasons, ipad and opad are used to derive two keys from the single key k.

Outline

Generic attacks: The Birthday Attack

- Suppose there are N people in a room. What is the probability that two people have the same birthday?
- How many people do we need to have a probability larger than 1/2 ?

Generic attacks: The Birthday Attack

- Suppose there are N people in a room. What is the probability that two people have the same birthday?
- How many people do we need to have a probability larger than 1/2 ?
- Answer is **23**:

$$\Pr[\text{all distinct}] = \frac{365}{365} \cdot \frac{364}{365} \cdot \frac{363}{365} \cdot \ldots \cdot \frac{365-22}{365} < \frac{1}{2}$$

Generic attacks: The Birthday Attack

- Suppose you choose q elements randomly in a set of N elements. What is the probability that two elements are equal?
- How large should q be with respect to N to have a probability larger than $1/2$?

Generic attacks: The Birthday Attack

- Suppose you choose q elements randomly in a set of N elements. What is the probability that two elements are equal?
- How large should q be with respect to N to have a probability larger than $1/2$?
- Let us try to solve it in a formal way...

The Birthday Problem

- Assume that you are throwing q balls to N bins. Let Coll denote the event that two balls end up being in the same bin. We can show that

$$q(q-1)/4N \leq \Pr[\text{Coll}] \leq q(q-1)/2N$$

- **Upper bound**: Let Coll_i denote the event that the i-th ball falls into an already occupied bin. Then $\Pr[\text{Coll}_i] \leq (i-1)/N$ as there are at most $i-1$ occupied bins.

$$\Pr[\text{Coll}] = \Pr[\bigvee_{i=1}^{q} \text{Coll}_i] \leq$$

$$\leq \sum_{i=1}^{q} \Pr[\text{Coll}_i] \leq 0/N + \cdots + (q-1)/N = \frac{q(q-1)}{2N}$$

The Birthday Problem

Lower bound: Let NoColl_i denote the event of not having any collision after throwing the i-th ball. It holds

$$\Pr[\text{NoColl}_i|\text{NoColl}_{i-1}] = (N-(i-1))/N \tag{1}$$

which is the probability of not falling in any of the previous $i-1$ bins (with $\Pr[\text{NoColl}_1] = 1$). Hence:

$$\Pr[\overline{\text{Coll}}] = \Pr[\text{NoColl}_q] \tag{2}$$

and we have

$$\Pr[\text{NoColl}_q] = \Pr[\text{NoColl}_q \cap \text{NoColl}_{q-1}] =$$

$$= \Pr[\text{NoColl}_q|\text{NoColl}_{q-1}] \cdot \Pr[\text{NoColl}_{q-1}]$$

Iterating the above reasoning, we obtain:

$$\Pr[\text{NoColl}_q] = \prod_{i=1}^{q-1} \Pr[\text{NoColl}_{i+1}|\text{NoColl}_i] \tag{3}$$

The Birthday Problem

From equations (1), (2) and (3)

$$\Pr[\overline{\mathsf{Coll}}] = \prod_{i=1}^{q-1}\left(1 - \frac{i}{N}\right) \tag{4}$$

Since $1 - x \leq e^{-x}$ when $x \leq 1$, and given that $i/N < 1$, thus

$$\Pr[\overline{\mathsf{Coll}}] \leq e^{-\sum_{i=1}^{q-1}(i/N)} = e^{-q(q-1)/2N}. \tag{5}$$

Therefore

$$\Pr[\mathsf{Coll}] \geq 1 - e^{-q(q-1)/2N}$$

where

$$1 - e^{-q(q-1)/2N} \geq q(q-1)/4N$$

if $q < \sqrt{2N}$, since $e^{-x} \leq 1 - x/2$ when $|x| \leq 1$.

Hash Functions: the Birthday Attack

- How does the birthday attack apply to hash functions?
- We have a probability $\approx$ 1/2 when $q \approx N^{1/2}$.
- For a hash function with output length ℓ, the range is of size 2^ℓ.
- When $q \approx 2^{\ell/2}$, the probability of finding a collision is $\approx 1/2$.
- In practice, to make finding collisions as difficult as exaustive search over 128-bit keys, you need a hash function with output length of at least 256 bits.
- This is necessary, but not sufficient!
- There are no generic attacks for preimage and second preimage resistance!

A Better Birthday Attack

- The original birthday attack uses lots of memory storage. It has to store $\mathcal{O}(q) = \mathcal{O}\left(2^{\ell/2}\right)$ values.
- Managing storage for 2^{60} bytes is often more difficult that executing 2^{60} CPU instructions.
- Can we do better?

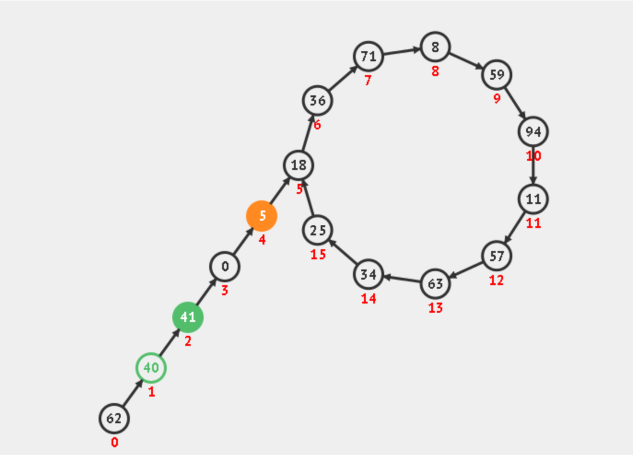
A Better Birthday Attack

- It is based on a cycle-finding algorithm of Floyd.
- We choose a random input x_0.
- We compute $x_i \leftarrow H(x_{i-1})$ and $x_{2i} \leftarrow H(H(x_{2(i-1)}))$ for $i = 1, 2, \ldots$, where $x_i = H^{(i)}(x_0)$.
- We compare x_i and x_{2i} after each iteration.
- If they are equal, then the collision happens somewhere in $x_0, \cdots, x_{2i-1}$.
- We try to find the smallest value of j for which $x_j = x_{j+i}$. The collision will then be (x_{j-1}, x_{j+i-1}).
- The algorithm has same time complexity and success probability as the general birthday attack, but only $\mathcal{O}(1)$ memory, namely, storage of two hashes in each iteration!

A better Birthday Attack

Floyd's cycle finding idea:

`https://visualgo.net/bn/cyclefinding`



A Better Birthday Attack

We are given $H : \{0,1\}^* \to \{0,1\}^\ell$, and we aim to find x, x' s.t. $H(x) = H(x')$.

$$
\begin{array}{l}
x_0 \leftarrow_{\$} \{0,1\}^{\ell+1} \\
x', x \leftarrow x_0 \\
\textbf{for } i = 1, 2, \cdots \textbf{ do} \\
\quad x \leftarrow H(x) = H^{(i)}(x_0) \\
\quad x' \leftarrow H(H(x')) = H^{(2i)}(x_0) \\
\quad \textbf{if } x = x' \textbf{ break} \\
x' \leftarrow x, x \leftarrow x_0 \\
\textbf{for } j = 1 \cdots, i \\
\quad \textbf{if } H(x) = H(x') \quad \textbf{return } x, x' \\
\quad \textbf{else } x \leftarrow H(x) = H^{(j)}(x_0) \\
\qquad\qquad x' \leftarrow H(x') = H^{(i+j)}(x_0)
\end{array}
$$

Further Reading (1)

- Mihir Bellare and Phillip Rogaway.
  Random oracles are practical: A paradigm for designing efficient protocols.
  In *Proceedings of the 1st ACM conference on Computer and communications security*, pages 62–73. ACM, 1993.
- Guido Bertoni, Joan Daemen, Michaël Peeters, and Gilles Van Assche.
  Keccak sponge function family main document.
  Submission to NIST (Round 2), 3:30, 2009.

Further Reading (2)

- Jean-Sébastien Coron, Yevgeniy Dodis, Cécile Malinaud, and Prashant Puniya.
  Merkle-damgård revisited: How to construct a hash function.
  In *Advances in Cryptology–CRYPTO 2005*, pages 430–448. Springer, 2005.
- Pierre Karpman, Thomas Peyrin, and Marc Stevens.
  Practical free-start collision attacks on 76-step sha-1.
  In *Advances in Cryptology–CRYPTO 2015*, pages 623–642. Springer, 2015.
- Neal Koblitz and Alfred J Menezes.
  The random oracle model: a twenty-year retrospective.
  Designs, Codes and Cryptography, pages 1–24, 2015.

Further Reading (3)

- Alfred J Menezes, Paul C Van Oorschot, and Scott A Vanstone.
  Handbook of applied cryptography.
  CRC press, 1996.
- Marc Stevens.
  New collision attacks on sha-1 based on optimal joint local-collision analysis.
  In *Advances in Cryptology–EUROCRYPT 2013*, pages 245–261. Springer, 2013.