[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Sanidhya Kumar (202151138)
Winter 2023-2024
Lecture (Week 7 & 8)

1 Hash Function

In the field of cryptography, a hash function is a mathematical process that accepts an input, or "message", and outputs a fixed-length string of bytes, usually in the form of a hash value or hash code. Often called a digest, the output is a distinct representation of the input data. Fast and effective hash functions offer a safe and dependable means of confirming the integrity of data, authenticating communications, and creating digital signatures.

A hash family is a four-tuple (X, Y, K, H), where:

1. X is a set of possible messages.
2. Y is a finite set of possible message digests or authentication tags (or just tags).
3. K, the keyspace, is a finite set of possible keys.
4. For each $k \in K$, there is a hash function $h_k \in H$, where $h_k : X \to Y$.

While Y is always a finite set in the definition above, it may not always be a finite set. The function is sometimes referred to as a compression function if X is a finite set and $|X| > |Y|$. In this case, we'll assume the more favorable circumstance. $|X| > 2^{|Y|}$.

1.1 Types of Hash Functions

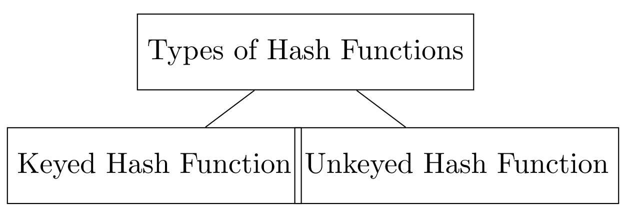


Figure 1: Types of Hash Functions

1.1.1 Unkeyed Hash Function

A function $h : X \to Y$, where X and Y are the same, is an unkeyed hash function. An unkeyed hash function can be conceptualized as a hash family where $|K| = 1$, or one with a single potential key. The output of an unkeyed hash function is commonly referred to as a "message digest."

1.1.2 Keyed Hash Function

If the key is involved in the computation of hashed value then that hash function is known as keyed hash function. The output of a keyed hash function is referred to as a "tag."

1.2 Legitimacy Under Hash Function

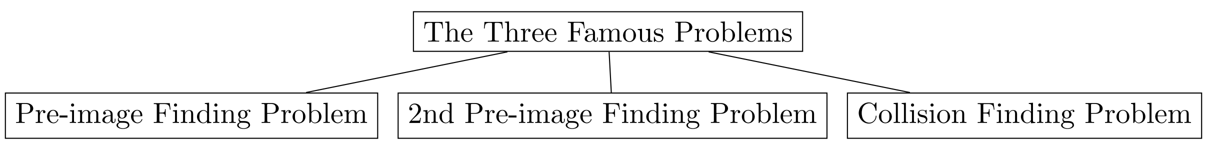
If $h(x) = y$, then a pair $(x, y) \in X \times Y$ is considered legitimate under a hash function h. In this case, h may be an unkeyed or keyed hash function. In this chapter, we mainly cover techniques to stop an opponent from creating specific kinds of valid pairs.

Let $F_{X,Y}$ denote the set of all functions from X to Y. Suppose that $|X| = N$ and $|Y| = M$. Then it is clear that $|F_{X,Y}| = M^N$. Any hash family F consisting of functions with domain X and range Y can be considered to be a subset of $F_{X,Y}$, i.e., $F \subseteq F_{X,Y}$. Such a hash family is termed an (N, M)-hash family.

1.3 Ideal Hash Function

Let $h : P \to S$. h is ideal if, given $x \in P$, to find $h(x)$, you either have to apply h on x or you have to look into the table corresponding to h.

1.4 The Three Famous Problems



1.4.1 Solution for Pre-image Finding Problem

Given $y \in Y$, $h : X \to Y$, $|Y| = M$, find $x \in X$.

Algorithm 1 Finding x such that $h(x) = y$

Input: $X_0 \subseteq X$ such that $|X_0| = Q$
Output: An element $x \in X_0$ such that $h(x) = y$
foreach $x \in X_0$ **do**
  Compute $y' = h(x)$ **if** $y' = y$ **then**
    return x
  end
end

This is also known as the exhaustive search method for finding the pre-image x given y. The probability of the above algorithm returning the correct pre-image is inversely proportional to the time complexity.

The provided equations analyze the probability and time complexity of finding pre-images in set X0. They demonstrate that as the size of X0 increases, both the probability and time complexity decrease, resulting in a more efficient pre-image search.

For example, $X_0 = \{x_1, x_2, \ldots, x_q\}$:

$$\begin{aligned}
P(\text{event } E_i) &= P(h(x_i) = y) = 1 - \frac{1}{M} \quad \text{(as 1 out of the length of } y \text{ will be the outcome)} \\
P(\text{event } E_i') &= 1 - P(E_i) = 1 - \frac{1}{M} \\
P(E_1 \cup E_2 \cup \ldots \cup E_q) &= 1 - P(E_1' \cap E_2' \cap \ldots \cap E_q') \\
&= 1 - P(E_1') \cdot P(E_2') \cdot \ldots \cdot P(E_q') \\
&= 1 - \left(1 - \frac{1}{M}\right)^q \\
&= \frac{Q}{M}
\end{aligned}$$

Therefore, complexity $= O(M/Q) = O(M)$, i.e., the bigger set of X_0, the greater the probability and the lesser the time complexity to find the pre-image.

1.4.2 Solution for 2nd Pre-image Finding Problem

Given x, $h(x)$, find x' such that $h(x) = h(x')$, where X_0 is a subset of X without x, and $|X_0| = Q$. Therefore, we can use the same algorithm as above and perform an exhaustive search. Time complexity $= O(M)$.

1.4.3 Solution for Collision Finding Problem

We have $h : X \to Y$, where $|Y| = M$. We need to find x' and x such that $x' \neq x$ and $h(x) = h(x')$.

Algorithm 2 Finding a pair of elements $\{x, x'\}$ with equal hash values

---

Input: $X_0 \subseteq X - \{x\}$ with $|X_0| = Q$
Output: A pair of elements $\{x, x'\}$ such that $h(x) = h(x')$ and $x \neq x'$
foreach $x' \in X_0$ **do**
  Compute $y_x = h(x)$ and $y_{x'} = h(x')$ **if** $y_x = y_{x'}$ **then**
    return $\{x, x'\}$
  end
end

Let E_i be the event that $h(x_i)$ is not equal to any of $\{h(x_1), h(x_2), \ldots, h(x_{i-1})\}$.

$$\begin{aligned}
\text{For } i = 1, &\quad Pr[E_1] = 1 \quad \text{(since } h(x_1) \text{ should not belong to the empty set).} \\
\text{For } i = 2, &\quad \text{given } E_1, Pr[E_2|E_1] = \frac{M-1}{M} \quad \text{(mapping to all elements except } h(x_1)\text{).} \\
\text{Similarly, for } i = 3, &\quad \text{given } E_1 \cap E_2, Pr[E_3|E_1 \cap E_2] = \frac{M-2}{M}. \\
\text{In general, for } i = k, &\quad \text{given } E_1 \cap E_2 \cap \ldots \cap E_{k-1}, Pr[E_k|E_1 \cap E_2 \cap \ldots \cap E_{k-1}] = \frac{M-(k-1)}{M}.
\end{aligned}$$

Therefore, the collision probability ϵ is given by:

$$\epsilon = 1 - Pr[E_1 \cap E_2 \cap \ldots \cap E_Q]' = 1 - \left(\frac{M-1}{M} \cdot \frac{M-2}{M} \cdot \ldots \cdot \frac{M-(Q-1)}{M}\right).$$

Solving for Q:

$$Q = \sqrt{2 \ln \left(\frac{1}{1-\epsilon} \right) \sqrt{M}}.$$

Hence, the value of Q is approximately the square root of M.

2 Compression Function

$h : \{0,1\}^{m+t} \to \{0,1\}^m$ is a hash function that takes inputs of length $m+t$ and produces outputs of length m. The goal is to construct a function $H : \{0,1\}^* \to \{0,1\}^m$ from h, where H takes inputs of any length and produces outputs of length m.

$h : \{0,1\}^{m+t} \to \{0,1\}^m$
$Secondpreimage, preimage \to O(2^m)$
$Collision \to O(2^{m/2})$

Algorithm 3 Compress

Suppose that Compress: $\{0,1\}^{m+t} \to \{0,1\}^m$ is a compression function.
Input:

- x : An input string of length greater than m + t + 1.

Output:

- $h(x)$: The hash value of the input string x.

Process

- Pad x with 0s to get a string y with a length divisible by t.
- Let $y = y_1||y_2||...||y_r$ where each y_i has length t (except possibly the last one).
- Initialize $z_0 \leftarrow IV$.
  For $i = 1$ to r do:
  $z_i \leftarrow compress(z_{i-1}||y_i)$

---

3 Merkle-Damgard Construction

The Merkle-Damgard construction has the property that the resulting hash function satisfies desirable security properties, such as collision resistance, provided that the compression function does. It helps in constructing a hash function from a compression function.

Suppose **Compress:** $\{0,1\}^{(m+t)} \to \{0,1\}^m$ is a collision-resistant compression function, where $t \geq 1$. So **compress** takes $m+t$ input bits and produces m output bits. We will use **compress** to construct a collision-resistant hash function $h : X \to \{0,1\}^m$; the hash function h takes any finite bitstring of length at least $m+t+1$ and creates a message digest that is a bitstring of length m.

Merkle-Damgård Hash Construction

Input: x: Input message

Output: $h(x)$: Hash value of x

Step 1: Calculate the length of the input message:

$$n = |x| \quad \text{(Length of input message)}$$

Step 2: Determine the number of blocks:

$$K = \left\lfloor \frac{n}{t-1} \right\rfloor \quad \text{(Number of blocks)}$$

Step 3: Calculate the padding size:

$$d = K(t-1) - n \quad \text{(Padding size)}$$

Step 4: For $i = 1$ to $K-1$, set $y_i = x_i$.

Step 5: Pad the last block:

$$y_K = x_K || 0^d \quad \text{(Pad last block)}$$

Step 6: Convert the padding size to binary representation:

$$y_{K+1} = \text{binary}(d) \quad \text{(Binary representation of padding size)}$$

Step 7: Initialize the state:

$$Z_1 = 0^{m+1} || y_1 \quad \text{(Initialize state)}$$

Step 8: Compress the initial state:

$$g_1 = \text{compress}(Z_1) \quad \text{(Compress initial state)}$$

Step 9: For $i = 1$ to K, update the state and compress it:

$$Z_{i+1} = g_i || 1 || y_{i+1} \quad \text{(Update state)}$$
$$g_{i+1} = \text{compress}(Z_{i+1}) \quad \text{(Compress state)}$$

Step 10: Final hash value:

$$h(x) = g_{K+1} \quad \text{(Final hash value)}$$

4 Secure Hash Functions (SHA)

The Secure Hash Algorithm (SHA) was devised by the National Institute of Standards and Technology (NIST) and published as a federal information processing standard (FIPS 180) in 1993. A revised edition, known as SHA-1, was released as FIPS 180-1 in 1995.

There exist three variants of SHA: SHA-160, SHA-256, and SHA-512, generally denoted as $SHA : \{0,1\}^* \to \{0,1\}^n$.

4.1 Types of SHA

Secure Hash Functions (SHA)

SHA-1	**SHA-256**	**SHA-512**
Message Size: $< 2^{64}$ bits	Message Size: $< 2^{64}$ bits	Message Size: $< 2^{128}$ bits
Block Size: 512 bits	Block Size: 512 bits	Block Size: 1024 bits
Word Size: 32 bits	Word Size: 32 bits	Word Size: 64 bits
Message Digest Size: 160 bits	Message Digest Size: 256 bits	Message Digest Size: 512 bits

4.2 SHA I in Detail

For SHA I, the message size is limited to 2^{64} bits. If $|x| \leq 2^{64}$, padding is applied such that y becomes a multiple of 512 bits by appending a single '1' followed by necessary '0's.

The SHA I algorithm involves four distinct functions and five constants, along with four keys and five initial hash values.

Algorithm 4 SHA I Algorithm

Input: x: Input message
Output: $h(x)$: Hash value of x

$n \leftarrow |x|$; $K \leftarrow \left\lfloor \frac{n}{t-1} \right\rfloor$; $d \leftarrow K(t-1) - n$; **for** $i = 1$ *to* $K - 1$ **do**
  $y_i \leftarrow x_i$;
end
$y_K \leftarrow x_K || 0^d$; $y_{K+1} \leftarrow \text{binary}(d)$; $Z_1 \leftarrow 0^{m+1} || y_1$; $g_1 \leftarrow \text{compress}(Z_1)$; **for** $i = 1$ *to* K **do**
  $Z_{i+1} \leftarrow g_i || 1 || y_{i+1}$; $g_{i+1} \leftarrow \text{compress}(Z_{i+1})$;
end
$h(x) \leftarrow g_{K+1}$; **return** $h(x)$;

5 Message Authentication Code (MAC)

A Message Authentication Code (MAC) serves as a cryptographic tool ensuring both the integrity and authenticity of a message or data transmission. Its primary function is to validate that a message hasn't been tampered with during transmission and originates from a trusted source.

5.1 HMAC (Hash-based Message Authentication Code)

HMAC stands for Hash-based Message Authentication Code and is widely adopted for message authentication and integrity verification purposes.

Working Principle:

- HMAC takes the message and a secret key as inputs.
- It utilizes a cryptographic hash function (e.g., SHA-256 or SHA-512) on the message, with the secret key as the hash function's "key".

- The resulting hash output undergoes further processing to generate the MAC.
- The MAC produced ensures both the integrity and authenticity of the message.

5.2 CBC-MAC (Cipher Block Chaining Message Authentication Code)

CBC-MAC, or Cipher Block Chaining Message Authentication Code, is employed for generating a MAC using a block cipher in Cipher Block Chaining (CBC) mode.

Operation:

- CBC-MAC processes fixed-size data blocks. If the message surpasses the block size, it is partitioned into blocks.
- Each block of the message undergoes processing using a symmetric encryption algorithm (e.g., AES) in CBC mode with a secret key.
- The resultant output of CBC-MAC constitutes the MAC for the entire message, which can be appended to or transmitted with the message for verification purposes.
- To thwart certain attacks like length extension attacks, CBC-MAC mandates a distinct initialization vector (IV) for each message.

6 Other SHAs

They are defined in the NIST Standard FIPS (Federal Information Processing Standards) 180-4. Rest of the things we were asked to refer from this PDF: **https://csrc.nist.gov/files/pubs/fips/180-2/final/docs/fips180-2.pdf**. Below is a small overview of the same.

6.1 SHA 256

Properties:

- **Message Size:** $< 2^{64}$ bits
- **Block Size:** 512 bits
- **Word Size:** 32 bits
- **Message digest size:** 256 bits

6.2 SHA-512

Properties:

- **Message Size:** Up to 2^{128} bits
- **Block Size:** 1024 bits
- **Word Size:** 64 bits
- **Message Digest Size:** 512 bits