[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Dipean Dasgupta (202151188)

Winter 2023-2024
Lecture 12,13,14 (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 the following conditions are satisfied:
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 $h_k \in H$. Each $h_k : X \to Y$.

While Y is always a finite set in the definition above, may not always be a finite or 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 favourable circumstance. $|X| > 2|3|$.

A function $h : X \to Y$, where X and Y are the same is an unkeyed hash function. An unkeyed hash function can be conceptualised 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," while the output of a keyed hash function is referred to as a "tag."

If $h(x) = y$, then a pair $(x, y) = X \times Y$ 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 $|F_{X,Y}| = M^N$. (This follows because, for each of the N possible inputs $x \in X$, there are M possible values for the corresponding output $h(x) = y$.) 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.

Security of Hash Functions:
Assume that the hash function $h : X- > Y$ is unkeyed. Define $y = h(x)$, given $x \in X$. It is desirable in many cryptographic applications of hash functions that the only method to generate a valid pair (x, y) is to compute $y = h(x)$ by applying the function h to x first.
In total, three problems are defined; if a hash function is to be considered secure, it should be the case that these three problems are difficult to solve.

Preimage
Instance: A hash function $h : X \to Y$ and an element $y \in Y$.
Find: $x \in X$ such that $h(x) = y$.
The issue Preimage asks whether an element $x \in X$ can be found such that h(x) = y given a (possible) message digest y. A value x of that kind would be a preimage of y.A pair (x, y) is legitimate if Preimage can be solved for a given $y \in Y$. One term for a hash function that is one-way or preimage resistant is that it cannot be solved effectively using Preimage.

Algorithm 1 FIND-PREIMAGE(h, y, Q)

Input:
h: A hash function. y: The output value. Q: The size of the set.
Output:

- An element $x \in X$ such that $h(x) = y$.
- "failure" if no such element exists.

1. Choose any subset $X_0 \subseteq X$ such that $|X_0| = Q$.
2. For each $x \in X_0$ do:
If $h(x) = y$, then return x. 3. Return "failure".

Second Preimage
Instance: A hash function $h : X \to Y$ and an element $x \in X$.
Find: $x' \in X$ such that $x' \neq x$ and $h(x') = h(x)$.
The Second Preimage problem asks if $x' \neq x$ can be discovered such that h(x') = h(x), given a message x. The goal is to identify a value x' that would be a second preimage of y. Here, we start with x, which is a preimage of y. Keep in mind that (x', h(x)) is an acceptable pair if this can be accomplished. It is common to refer to a hash function as second preimage resistant when it is incapable of being solved efficiently for Second Preimage.

Algorithm 2 FIND-SECOND-PREIMAGE(h, x, Q)

Input:
h: A hash function. x: element in domain h: Q: The size of the set.
Output:

- An element $x_0 \in X \setminus \{x\}$ such that $h(x_0) = h(x)$.
- "failure" if no such element exists.

1. $y \leftarrow h(x)$
2. Choose $X_0 \subseteq X \setminus \{x\}$ such that $|X_0| = Q - 1$.
3. For each $x_0 \in X_0$ do: If $h(x_0) = y$, then return x_0.
4. Return "failure".

Collision
Instance: A hash function $h : X \to Y$.
Find: $x, x' \in X$ such that $x' \neq x$ and $h(x') = h(x)$.
The issue The collision problem asks if there is any pair of different inputs, x, x', such that h(x') =

h(x). Two legitimate pairs, (x, y) and (x', y), where y = h(x) = h(x'), are produced as a solution to this problem. There are several situations in which we would like to prevent this kind of thing from happening. It's common to refer to a hash function as collision-resistant when Collision cannot be addressed effectively.

Algorithm 3 FIND-COLLISION(h, Q)

Input: h: A hash function. Q: The size of the set.
Output:

- A pair of elements (x, x') such that $h(x) = h(x')$ and $x \neq x'$.
- "failure" if no such pair exists.

1. Choose a set $X_0 \subseteq X$ such that $|X_0| = Q$.
2. For each $x \in X_0$ do:
Let $y_x \leftarrow h(x)$.
If there exists $y_x = y'_x$ for some $x' \neq x$
then return (x, x').
3. Return "failure".

Compression Function:
$h : 0, 1^(m + t) \rightarrow 0, 1^n$
$Secondpreimage, preimage \rightarrow O(2^m)$
$Collision \rightarrow O(2^(m/2)$

Algorithm 4 Compress

Suppose that Compress: $\{0,1\}^(m + t) -> \{0,1\}^m$ 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:
$z_i \leftarrow compress(z_{i-1}||y_i)$

Merkle-Damgard construction

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 $Compress : \{0,1\}^(m+t) -> \{0,1\}^m$ is a collision resistant compression function, where t ¿= 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.

Algorithm 5 MERKLE-DAMGÅRD(x)

---

external compress

comment: compress: $\{0,1\}^{m+t} \to \{0,1\}^m$ where $t \geq 2$

$n \leftarrow |x|$
$k \leftarrow \lfloor n/(t-1) \rfloor$

$d \leftarrow k(t-1) - n$
for $i \leftarrow 1$ **to** k **do**
    $yi \leftarrow Xi$
end for
$Y_k \leftarrow X_k || O^d$
$y_{k+1} \leftarrow thebinaryrepresentationofd$
$z_1 \leftarrow 0^{m+1} || y_1$
$z_1 \leftarrow 0^{m+1} || y_1$
$compress(z_1)$
for $i \leftarrow 1$ **to** k **do**
    $Z_i + 1 \leftarrow Z_i || 1 || Y_i + 1$
    $compress(Z_i + 1)$
end for
$h(x) \leftarrow g_{k+1}$
return $(h(x))$

Secure Hash Algorithm(SHA)

SHA1

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that takes an input message of arbitrary length and produces a fixed-length output, known as a message digest. The output is a 160-bit (20-byte) value that is typically represented as a hexadecimal number that is 40 digits long.

Algorithm

The SHA-1 algorithm consists of four main steps: preprocessing, processing, concatenation, and formatting.

Preprocessing

The input message is padded and divided into blocks of a fixed length (512 bits).

Processing

Each block is processed in 80 rounds, using a series of bitwise operations, modular arithmetic, and logical functions to produce a series of intermediate hash values.

Algorithm 6 SHA-1 Processing

procedure ProcessBlock(*block*)
    $h_0 \leftarrow H_0$
    $h_1 \leftarrow H_1$
    $h_2 \leftarrow H_2$
    $h_3 \leftarrow H_3$
    $h_4 \leftarrow H_4$
    for $i = 0$ to 79 **do**
        $W_i \leftarrow \text{Expand}(W_{i-3}, W_{i-2}, W_{i-1})$
        $T_i \leftarrow \text{CircularShift}(W_i, 1) + \text{CircularShift}(W_i, 8) + \text{CircularShift}(W_i, 14) + \text{CircularShift}(W_i, 16)$
        $T_i \leftarrow (T_i + f_i(h_{i-3}, h_{i-2}, h_{i-1}) + h_i + K_i) \mod 2^{32}$
        $h_i \leftarrow h_{i-4} + T_i$
    end for
    return h_0, h_1, h_2, h_3, h_4
end procedure

In the processing step, the following functions are used:

- $W_i = \text{Expand}(W_{i-3}, W_{i-2}, W_{i-1})$: This function expands a 32-bit value into a 32-bit value using a series of bitwise operations.
- $T_i = \text{CircularShift}(W_i, s)+\text{CircularShift}(W_i, 8)+\text{CircularShift}(W_i, 14)+\text{CircularShift}(W_i, 16)$: This function performs a circular shift on a 32-bit value by a certain number of bits (s).
- $f_i(h_{i-3}, h_{i-2}, h_{i-1})$: This function performs a logical operation on three 32-bit values.
- K_i: This is a constant value that is used in the processing step.

Concatenation:
The intermediate hash values are concatenated to produce the final message digest.

Message Authentication Code(MAC)
A Message Authentication Code (MAC) is a short piece of information used to authenticate a message and verify its integrity. A MAC is generated by using a shared secret key between the sender and receiver, and it is sent along with the message. When the message is received, the receiver generates its own MAC using the same algorithm and shared key, and compares it to the MAC received with the message. If the two MACs match, the message is authenticated and its integrity is verified.

HMAC
HMAC (Hash-based Message Authentication Code) is a specific type of MAC that uses a cryptographic hash function, such as SHA-1, as its underlying hash function. The HMAC algorithm is defined as follows:

$$\text{HMAC} = H((K \oplus \text{opad})||H((K \oplus \text{ipad})||M))$$

where K is the secret key, M is the message, $\oplus$ denotes bitwise XOR, ipad and opad are fixed padding values, and H is the cryptographic hash function.

Algorithm 7 HMAC

procedure HMAC(K, M)
    ipad $\leftarrow$ 0x36 . . . 0x36
    opad $\leftarrow$ 0x5c . . . 0x5c
    $K' \leftarrow K \oplus$ ipad
    $K'' \leftarrow K \oplus$ opad
    $M' \leftarrow H(K', M)$
    $M'' \leftarrow H(K'', M')$
    return M''
end procedure

CBC-MAC

CBC-MAC (Cipher Block Chaining: Message Authentication Code) is another type of MAC that uses a block cipher, such as AES, in CBC mode to generate the MAC. The CBC-MAC algorithm is defined as follows:

$$\text{CBC-MAC}(K, M) = E_K(E_K(\ldots E_K(E_K(IV) \oplus M_1) \oplus M_2) \oplus \ldots) \oplus M_n$$

where K is the secret key, M is the message divided into n blocks, E_K denotes encryption with the block cipher using key K, and $\oplus$ denotes bitwise XOR. The IV (Initialization Vector) is a fixed value or a random value that is used to initialize the CBC mode. **SHA-256**

Algorithm 8 CBC-MAC

procedure CBC-MAC(K, M)
    $IV \leftarrow$ fixed value or random value
    $C \leftarrow IV$
    for $i = 1$ to n **do**
        $C \leftarrow E_K(C \oplus M_i)$
    end for
    return C
end procedure

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that takes an input message of arbitrary length and produces a fixed-length output, known as a message digest. The output is a 256-bit (32-byte) value that is typically represented as a hexadecimal number that is 64 digits long. The algorithm is designed to be deterministic, meaning that the same input will always produce the same output, and to be resistant to various types of attacks, such as collisions and preimage attacks.

Algorithm

The SHA-256 algorithm consists of four main steps: preprocessing, processing, concatenation, and formatting.

Preprocessing

The input message is padded and divided into blocks of a fixed length (512 bits).

Processing

Each block is processed in 64 rounds, using a series of bitwise operations, modular arithmetic, and

logical functions to produce a series of intermediate hash values.

In the processing step, the following functions are used:

Algorithm 9 SHA-256 Processing

procedure ProcessBlock(*block*)
  $h_0 \leftarrow H_0$
  $h_1 \leftarrow H_1$
  $h_2 \leftarrow H_2$
  $h_3 \leftarrow H_3$
  $h_4 \leftarrow H_4$
  $h_5 \leftarrow H_5$
  $h_6 \leftarrow H_6$
  $h_7 \leftarrow H_7$
  for $i = 0$ to 63 **do**
    $W_i \leftarrow \text{Expand}(W_{i-2}, W_{i-15}, W_{i-16})$
    $T_1 \leftarrow \text{Sigma1}(e_i) + \text{Ch}(e_{i-2}, e_{i-15}, e_{i-16}) + e_{i-6} + K_i$
    $T_2 \leftarrow \text{Sigma0}(a_i) + \text{Maj}(a_{i-1}, a_{i-2}, a_{i-3}) + a_{i-7} + K'_i$
    $T_1 \leftarrow T_1 + T_2$
    $T_3 \leftarrow \text{CircularShift}(T_1, 1) + \text{CircularShift}(T_1, 8) + \text{CircularShift}(T_1, 16) + \text{CircularShift}(T_1, 24)$
    $T_3 \leftarrow T_3 + T_1$
    $a_i \leftarrow d_i + T_3$
    $e_i \leftarrow c_i$
    $c_i \leftarrow b_i$
    $b_i \leftarrow a_i$
    $d_i \leftarrow T_3$
  end for
  return $h_0, h_1, h_2, h_3, h_4, h_5, h_6, h_7$
end procedure

- $W_i = \text{Expand}(W_{i-2}, W_{i-15}, W_{i-16})$: This function expands a 32-bit value into a 32-bit value using a series of bitwise operations.
- $T_1 = \text{Sigma1}(e_i) + \text{Ch}(e_{i-2}, e_{i-15}, e_{i-16}) + e_{i-6} + K_i$: This function performs a series of bitwise operations and modular arithmetic on the e values.
- $T_2 = \text{Sigma0}(a_i) + \text{Maj}(a_{i-1}, a_{i-2}, a_{i-3}) + a_{i-7} + K'_i$: This function performs a series of bitwise operations and modular arithmetic on the a values.
- $T_3 = \text{CircularShift}(T_1, 1) + \text{CircularShift}(T_1, 8) + \text{CircularShift}(T_1, 16) + \text{CircularShift}(T_1, 24)$: This function performs a circular shift on a 32-bit value by a certain number of bits.
- $\text{Sigma1}(x) = \text{CircularShift}(x, 19) + \text{CircularShift}(x, 61) + \text{CircularShift}(x, 6)$
- $\text{Sigma0}(x) = \text{CircularShift}(x, 28) + \text{CircularShift}(x, 2) + x$
- $\text{Ch}(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$
- $\text{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$

Concatenation
The intermediate hash values are concatenated to produce the final message digest.