[CS304] Introduction to Cryptography and Network Security

Course Instructor: Dr. Dibyendu Roy
Scribed by: Raghav Agiwal (202151124)

Winter 2023-2024
Lecture (Week 7 & 8)

Week 7 & 8:

Algorithm for Finding a Second Preimage

Given a hash function h that maps inputs from a set X to a set Y, where $|Y| = m$, our objective is to identify another input x' apart from a given input x, such that $h(x) = h(x')$, but x and x' are distinct. Let's denote X_0 as a subset of X excluding x', meaning X_0 contains elements from X except for x'. The size of X_0 is denoted as q, indicating the number of elements in X_0. The computational complexity of this operation is of order n.

The ultimate aim is to find an input x' that, when hashed, produces the same hash value as the given input x, but x' is not the same as x.

Collision Finding Algorithm

Consider a hash function $h : X \to Y$, where $|X| = n$ and $|Y| = m$. The objective of the collision finding algorithm is to discover two distinct inputs x and x' from the set X such that $h(x) = h(x')$, where $x \neq x'$.

Let $X_0 = \{x_1, x_2, \ldots, x_q\}$ be a subset of X excluding x', and let's assess the probability of finding a collision within X_0.

For each pair x, x' in X_0, the algorithm computes:

$$Y_x = h(x),$$

$$Y_{x'} = h(x').$$

If there exists some $x \neq x'$ such that $Y_x = Y_{x'}$, then the algorithm returns the pair (x, x') as the collision pair. Otherwise, it returns failure.

It's essential to note that the success of finding a collision significantly depends on the selection of X_0. The algorithm's effectiveness relies on the diversity and coverage of inputs in X_0. Choosing a comprehensive subset X_0 enhances the chances of finding collisions efficiently.

Probability of Success in Finding a Collision

To determine the probability of success in finding a collision, let's analyze the events and their complements as described:

- Let E_i be the event that $h(x_i)$ belongs to the set $\{h(x_1), h(x_2), \ldots, h(x_{i-1})\}$.
- If E_i complement occurs, it indicates a collision.
- $h(x_i) \neq \{\}$ is always true for $i = 1$.

- $p(E_1)$ is a deterministic event, indicating the probability that $h(x_1)$ belongs to the empty set, which is always 1.
- For $i > 1$, $p(E_i)$ indicates the probability that $h(x_1)$ belongs to the set $\{h(x_1)\}$, implying that m different hash values map to $m - 1$ elements.
- E_2 represents the event that $h(x_2)$ equals some value z where z belongs to Y excluding $h(x_1)$.

2.1 The Probability of Success

If E_i is the event $h(x_i) \in \{h(x_1), h(x_2), \ldots, h(x_{i-1})\}$.

If E_i^c complement occurs, we get the collision $h(x_i) \neq \{\}$ is always true for $i = 1$. $p(E_1) \rightarrow$

$$h(x_1) \in \{\}.$$

$p(E_1)$ is a deterministic event.
$p(E_2) \rightarrow h(x_1) \in \{h(x_1)\}$.
i.e., m set $h(x_2)$ maps to $m - 1$ elements.
E_2: $h(x_2) = z, z \in Y - \{h(x_1)\}$.
Collision finding algorithm for success:
If $h(x_i) \in \{h(x_1), \ldots, h(x_{i-1})\}$ for any i.
Calculate union of E_i^c.

$$\epsilon = \text{probability of collision.}$$

If $\epsilon = 0.5$,

$$= 1 - x$$

$$Q^{-1} = Ye^{-i/M}$$

$$i = 1$$

$$Q^2 - Q \approx -2M \ln(1 - \epsilon)$$

$$\surd \approx 1.177M$$

If $\epsilon = 0.9$, i.e., if we want to have the collision probability of 0.9

$$\surd = 2.145M$$

$$\surd Q \approx M$$

is the complexity of our algorithm.

Secure hash function $h : \{0,1\}^* \rightarrow \{0,1\}^m$.
h is a secure hash function. Preimage $\rightarrow O(2^m)$ Collision $\rightarrow O(2^{m/2})$
Compression Function: $h : \{0,1\}^{m+t} \rightarrow \{0,1\}^m, t \geq 1$.
h is a secure hash function. Preimage, 2nd preimage $\rightarrow O(2^m)$ Collision $\rightarrow O(2^{m/2})$
Target

Construct $H : \{0,1\}^* \to \{0,1\}^m$ from h.
Security of H will completely depend on the security of h.
Given $x \in \{0,1\}^*$, $|x|$: length of x:

$$|x| \geq m + t + 1$$

From x construct y using a public function such that $y \equiv O \pmod{t}$:

$$y = \begin{cases} x|f(|x|) \equiv O \pmod{t} & \text{if } |x| \equiv 0 \pmod{t} \\ x|0^d|f(|x|+d) \equiv O \pmod{t} & \text{otherwise} \end{cases}$$

Select $IV \in \{0,1\}^m$ (IV is public).
$y = y_1|y_2|\ldots|y_r$ such that $|y_i| = t$, $1 \leq i \leq r$.
H:

$$\begin{aligned} z_0 &= IV \\ z_1 &= h(z_0|y_1) \\ z_2 &= h(z_1|y_2) \\ &\cdots \\ z_r &= h(z_{r-1}|y_r) \end{aligned}$$

$z_r = H(x)$ is known as an iterative hash function.

Merkle-Damg and Construction

The Merkle-Damgård construction is a method used to construct cryptographic hash functions from a compression function. It is widely used in many hash functions like MD5, SHA-1, and SHA-2.

Here's a breakdown of the Merkle-Damgård construction:

1. **Compression Function**:
   - Given a compression function $compress : \{0,1\}^{m+t} \to \{0,1\}^m$ where $t \geq 2$, it takes a block of input and compresses it into a fixed-size output.
2. **Input Partitioning**:
   - The input message X is divided into blocks $X_1, X_2, \ldots, X_k$, each of size $t-1$ bits.
   - X is padded to ensure its length is a multiple of $t-1$.
3. **Processing Blocks**:
   - For $i = 1$ to $k-1$, each block X_i is processed to produce an intermediate hash value y_i.
   - The last block X_k is processed differently. It is concatenated with padding 0^d where d is the number of bits required to make the length of X a multiple of $t-1$, and y_{k+1} is set to the binary representation of d.
4. **Construction of Intermediate Hash Values**:
   - $z_1 = opad \parallel y_1$, where $opad$ is a fixed padding value.
   - $g_1 = compress(z_1)$.

- For $i = 1$ to k, $z_{i+1} = g_i \parallel ipad \parallel y_{i+1}$, where $ipad$ is another fixed padding value.
- $g_{i+1} = compress(z_{i+1})$.

5. **Final Hash Output**:
   - The final hash output $h(x)$ is set to g_{k+1}, the last computed intermediate hash value.

Lecture 14:

Secure Hash Functions

A secure hash function is a cryptographic algorithm that takes an input (or message) and produces a fixed-size string of bytes, typically represented in hexadecimal format. These hash functions have several key properties:

1. **Deterministic**: For a given input, the hash function always produces the same output.
2. **Fast Computation**: The hash function should be computationally efficient to compute the hash value for any given input.
3. **Preimage Resistance**: Given a hash value, it should be computationally infeasible to find the original input.
4. **Second Preimage Resistance**: Given an input, it should be computationally infeasible to find a different input that produces the same hash value.
5. **Collision Resistance**: It should be computationally infeasible to find two different inputs that produce the same hash value.

For SHA-1 (Secure Hash Algorithm 1), the following steps outline its operation:

1. **Padding the Message**: The input message x is padded to ensure its length satisfies certain conditions.
2. **Message Processing**: The message is divided into blocks and processed sequentially.
3. **Initialization**: Initial hash values (H0 to H4) and constants (Kt) are predefined.
4. **Message Schedule**: Each block of the message is expanded into a set of words.
5. **Main Loop**: A series of bitwise operations, rotations, and additions are performed on the message schedule.
6. **Update Hash Values**: The hash values are updated iteratively based on the main loop operations.
7. **Final Hash Output**: The final hash value is obtained by concatenating the updated hash values.

The mathematical notation for the secure hash function SHA-1 is as follows:

$$
\begin{aligned}
d &= (447 - |x|) \mod 512 \\
l &= \text{binary}(|x|), \text{max of 64 bits} \\
y &= x \parallel 1 \parallel 0^d \parallel l \\
|x| + d &\equiv 447 \mod 512 \\
|y| &\equiv 0 \mod 512 \\
\text{ROTL}(x) &= \text{circular left shift of } x \text{ by } s \text{ positions}
\end{aligned}
$$

Define the function $F_t(b, c, d)$ as follows:

$$
F_t(b,c,d) = \begin{cases} (b \wedge c) \vee (\neg b \wedge d) & \text{if } 0 \leq t \leq 19 \\ b \oplus c \oplus d & \text{if } 20 \leq t \leq 39 \\ (b \wedge c) \vee (b \wedge d) \vee (c \wedge d) & \text{if } 40 \leq t \leq 59 \\ b \oplus c \oplus d & \text{if } 60 \leq t \leq 79 \end{cases}
$$

$$
\begin{aligned}
\text{SHA-1}(X) &= \text{SHA-1-PAD}(A) \\
y &= M_1 \parallel M_2 \parallel \ldots \parallel M_n, \quad |M_i| = 512
\end{aligned}
$$

Initial hash values:

$$
\begin{aligned}
H_0 &= 67452301 \\
H_1 &= EFCDAB89 \\
H_2 &= 98B4D\Delta FE \\
H_3 &= 10325476 \\
H_4 &= CBDZEIFO
\end{aligned}
$$

Constants:

$$
K_t = \begin{cases} 5A827999 & \text{if } 0 \leq t \leq 19 \\ 6ED9EBA1 & \text{if } 20 \leq t \leq 39 \\ 8F1BBCDC & \text{if } 40 \leq t \leq 59 \\ CA62C1DC & \text{if } 60 \leq t \leq 79 \end{cases}
$$

Message Authentication Code (MAC)

A Message Authentication Code (MAC) is a cryptographic technique used to verify the integrity and authenticity of a message. It is generated using a secret key shared between the sender and receiver. Here's an explanation of how MAC works:

1. **Generation**:
   - Alice wants to send a message M to Bob securely. She generates a MAC MAC by applying a hash function to the message M along with a secret key K. This ensures that only Alice and Bob, who share the secret key, can generate and verify the MAC.
2. **Verification**:

- Bob receives the message M along with the MAC MAC. He applies the same hash function to the received message M using the shared secret key K to compute a MAC (MAC_1).
- Bob compares the computed MAC (MAC_1) with the received MAC (MAC). If they match, Bob accepts the message M as authentic and unaltered.

Mathematical Notation for MAC:

$$\begin{aligned}
\text{Alice:} \quad & c = \text{enc}(M, K) \xrightarrow{\sim C} c = \text{enc}(M, K) \\
\text{MAC:} \quad & \text{MAC} = \text{hash}(M, K) \xrightarrow{\text{MAC}} \sim \text{MAC} \\
\text{Bob:} \quad & \text{dec}(\sim c, K) = \sim M \\
& \text{hash}(\sim M, K) = \text{MAC}_1 \\
\text{If} \quad & \text{MAC}_1 = \sim \text{MAC}, \quad \text{accept} \sim M
\end{aligned}$$

HMAC (Hash-based Message Authentication Code)

The HMAC (Hash-based Message Authentication Code) is a mechanism used for verifying the integrity and authenticity of a message through cryptographic means. It utilizes a hash function and a secret key to generate a fixed-size hash value, which serves as the authentication code.

Given:

$$\begin{aligned}
\text{ipad} &= 36\ 36\ ...\ 36 && (512 \text{ bits}) \\
\text{opad} &= 5c\ 5c\ ...\ 5c && (512 \text{ bits}) \\
K &\quad \text{is the secret key}
\end{aligned}$$

HMAC Calculation:

$$\text{HMAC}(x) = \text{SHA-1}(K \oplus \text{opad}) \parallel \text{SHA-1}((K \oplus \text{ipad}) \parallel x)$$

In this equation:

- $\oplus$ denotes the bitwise XOR operation.
- $\parallel$ represents concatenation.
- x is the message to be authenticated.

The HMAC function involves two steps:

1. **Key Transformation**: The secret key K is XORed with both ipad and opad to generate two derived keys.
2. **Hashing**: The message x is hashed along with the derived keys using the SHA-1 hash function. The resulting hashes are concatenated to form the HMAC.

HMAC provides a robust method for ensuring the integrity and authenticity of messages, even in the presence of attackers trying to tamper with or forge messages. It is widely used in various cryptographic protocols and systems to provide message authentication.

CBC-MAC (Cipher Block Chaining Message Authentication Code)

CBC-MAC (Cipher Block Chaining Message Authentication Code) is a method for generating a message authentication code (MAC) using a block cipher algorithm. It operates by encrypting the input message in a chained manner using a secret key, resulting in a fixed-size authentication tag.

Input:

$x = x_1 \parallel x_2 \parallel \ldots \parallel x_n$, where x_i represents the i-th block of the message.

K is the secret key.

Initial vector $\text{iv} = 00\ldots0$.

Process:

- Initialize $y_0 = \text{iv}$.
- For each block x_i of the message:
  - Compute $y_i = \text{Enc}(y_{i-1} \oplus x_i, K)$, where Enc represents the encryption function of the block cipher algorithm.
- Return the last block y_n as the CBC-MAC authentication tag.

In this process:

- $\oplus$ denotes the bitwise XOR operation.
- Enc is the encryption function of the block cipher algorithm, which typically uses symmetric encryption algorithms like AES (Advanced Encryption Standard).

CBC-MAC provides a fixed-size authentication tag for the input message, ensuring message integrity and authenticity. It is commonly used in various cryptographic protocols and systems for message authentication purposes.

SHA-256 (Secure Hash Algorithm 256)

SHA-256 (Secure Hash Algorithm 256) is a cryptographic hash function that produces a fixed-size output of 256 bits (32 bytes). It is designed to take an input message of arbitrary length and generate a unique and deterministic hash value, commonly represented as a 64-character hexadecimal string.

Mathematical Overview:

- **Input**: Let M be the input message, which can have any length.
- **Padding**: The input message M is padded to ensure its length is a multiple of 512 bits.
- **Processing**: The padded message is divided into blocks of 512 bits each. Each block is processed sequentially using a series of bitwise operations, modular addition, logical functions (AND, OR, XOR), and circular shifts.
- **Compression Function**: SHA-256 uses a compression function f to process each block of the message.
- **Message Schedule**: The 512-bit message block is further divided into sixteen 32-bit words, which are expanded into a schedule of 64 32-bit words.

- **Round Function**: Each round of SHA-256 involves processing the message schedule through a series of 64 rounds.
- **Hash Value**: After processing all blocks, the final hash value is obtained by concatenating the outputs of the compression function applied to each block.

SHA-256 is widely used in various cryptographic applications due to its properties such as preimage resistance, second preimage resistance, and collision resistance.

Functions and Sigma Functions used in SHA-256

Functions

- $\text{ROTR}_n(x) = (x \ggg n) \vee (x \lll (w - n))$
- $\text{SHR}_n(x) = x \ggg n$
- $C(x, y, z) = (x \wedge y) \oplus (\neg x \wedge z)$
- $\text{Maj}(x, y, z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z)$

Sigma Functions

- $\Sigma_0(x) = \text{ROTR}_2(x) \oplus \text{ROTR}_{13}(x) \oplus \text{ROTR}_{22}(x)$
- $\Sigma_1(x) = \text{ROTR}_6(x) \oplus \text{ROTR}_{11}(x) \oplus \text{ROTR}_{25}(x)$

SHA-256 Algorithm

The SHA-256 algorithm processes a message into a 256-bit hash value:

- Divide the message into 512-bit blocks.
- For each block:
  - Perform message schedule expansion.
  - Update the hash value using a compression function.
- Concatenate the hash values to obtain the final hash value.