Lab 11: Implementation of SHA-1 Algorithm

Course: Information Security & Cryptography (AI Department, SVNIT Surat)

Objective

To implement the Secure Hash Algorithm (SHA-1), an iterated hash function that produces a 160-bit message digest. Students will understand message padding, bitwise operations, and the design of cryptographic hash functions.

Background

SHA-1 (Secure Hash Algorithm 1) is a one-way cryptographic hash function that takes an input message of arbitrary length and produces a fixed-size 160-bit output (digest). It processes input blocks of 512 bits and operates on 32-bit words through a sequence of bitwise operations and modular additions.

Algorithm Overview

SHA-1 follows the general structure of an iterated hash function. The algorithm can be divided into the following major steps:

1. **Padding the message:**
   Given a message x of length $|x|$, compute:

   $$d = (447 - |x|) \bmod 512, \qquad \ell = \text{binary representation of } |x|$$

   Append a single 1 bit, followed by d zero bits, and finally append ℓ (64-bit binary of message length). The resulting message y will have a length that is a multiple of 512 bits.

2. **Initialization:**
   Initialize five 32-bit words as:

   $H_0 = 67452301_{16}$,
   $H_1 = EFCDAB89_{16}$,
   $H_2 = 98BADCFE_{16}$,
   $H_3 = 10325476_{16}$,
   $H_4 = C3D2E1F0_{16}$

3. **Processing each 512-bit block:**
   Divide y into n blocks $M_1, M_2, \ldots, M_n$, each 512 bits. For each block M_i, divide it into sixteen 32-bit words $W_0, \ldots, W_{15}$. Then, for $t = 16$ to 79:

   $$W_t = \text{ROTL}^1(W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16})$$

   Here, ROTL^s denotes circular left shift by s positions for $0 \leq s \leq 31$.

4. **Main loop (80 rounds):**

$$A \leftarrow H_0, \quad B \leftarrow H_1, \quad C \leftarrow H_2, \quad D \leftarrow H_3, \quad E \leftarrow H_4$$

$$\text{for } t = 0 \text{ to } 79 \text{ do: } \begin{cases} \text{temp} \leftarrow \text{ROTL}^5(A) + f_t(B,C,D) + E + W_t + K_t \\ E \leftarrow D \\ D \leftarrow C \\ C \leftarrow \text{ROTL}^{30}(B) \\ B \leftarrow A \\ A \leftarrow \text{temp} \end{cases}$$

5. **Update:**

$$H_0 \leftarrow H_0 + A, \quad H_1 \leftarrow H_1 + B, \quad H_2 \leftarrow H_2 + C, \quad H_3 \leftarrow H_3 + D, \quad H_4 \leftarrow H_4 + E$$

6. **Output:** The final message digest is:

$$H_0 \| H_1 \| H_2 \| H_3 \| H_4$$

Function Definitions

Each round function $f_t(B,C,D)$ and constant K_t are defined as follows:

$$f_t(B,C,D) = \begin{cases} (B \wedge C) \vee (\neg B \wedge D), & 0 \le t \le 19 \\ B \oplus C \oplus D, & 20 \le t \le 39 \\ (B \wedge C) \vee (B \wedge D) \vee (C \wedge D), & 40 \le t \le 59 \\ B \oplus C \oplus D, & 60 \le t \le 79 \end{cases}$$

$$K_t = \begin{cases} 5A827999_{16}, & 0 \le t \le 19 \\ 6ED9EBA1_{16}, & 20 \le t \le 39 \\ 8F1BBCDC_{16}, & 40 \le t \le 59 \\ CA62C1D6_{16}, & 60 \le t \le 79 \end{cases}$$

Lab Tasks

1. Implement the SHA-1 algorithm in C++ using the steps above.
2. Your program should:
   - Take an input message (string).
   - Display the padded binary message.
   - Show intermediate hash values after each block (optional for debugging).
   - Output the final 160-bit (40-hex-digit) SHA-1 message digest.
3. Compare your result with the output of a standard library implementation.
4. Submit your source code and a brief report (1–2 pages) describing:

- Algorithm flow,
- Key observations during implementation,
- Example test cases and their outputs.

Example Input/Output

Note: Although SHA-1 processes the message in 512-bit blocks after padding, the input message can be of any length. The algorithm internally converts the input string into a binary representation, appends padding bits, and processes the resulting bitstring in 512-bit chunks.

- **Example 1 (Short text input):**
  Input: `"The quick brown fox jumps over the lazy dog"`
  Output (expected):
  `2fd4e1c67a2d28fced849ee1bb76e7391b93eb12`
- **Example 2 (Another text input):**
  Input: `"Information Security and Cryptography Lab"`
  Output (expected):
  `2fd42e97d9111551984ac20b7e5dfa4432666165`
- **Example 3 (Empty input):**
  Input: `""` (empty string)
  Output (expected):
  `da39a3ee5e6b4b0d3255bfef95601890afd80709`

Submission

Submit a `.zip` file containing:

- Source code (`.cpp`)
- Report in PDF

Deadline: As announced on the course portal.