Lab 11 Report: SHA-1 Implementation
Course: Information Security & Cryptography
By- U23AI065

1. Algorithm Flow

The implemented algorithm follows the SHA-1 specification as detailed in the lab manual. The process is as follows:

1. **Padding:** The input message is padded to ensure its length is a multiple of 512 bits. This involves appending a single '1' bit, followed by a variable number of '0' bits, and finally, a 64-bit block representing the original message's length in bits.
2. **Initialisation:** Five 32-bit hash registers (H0 to H4) are initialised to specific hexadecimal constants.
3. **Block Processing:** The padded message is processed in 512-bit (64-byte) chunks. For each chunk:
    - **Word Expansion:** The 64-byte block is divided into sixteen 32-bit words (W0 to W15). These are then expanded into eighty 32-bit words (W0 to W79) using a formula involving ROTL and XOR operations.
    - **Main Loop:** An 80-round loop is executed. Each round uses a different logical function (f_t) and a round constant (K_t) to update five temporary variables (A, B, C, D, E), which were initialised from the main H registers.
4. **Hash Update:** After 80 rounds, the temporary variables (A-E) are added (using modular addition) to the main hash registers (H0-H4).
5. **Output:** Once all blocks are processed, the final 160-bit message digest is produced by concatenating the five 32-bit H registers.

2. Key Observations During Implementation

- **Endianness:** SHA-1 specifies a big-endian byte order. When reading a 512-bit block into the sixteen 32-bit words (W0-W15), it was necessary to manually construct each uint32_t from four uint8_t bytes using bitwise shifts (e.g., (b[0] << 24) | (b[1] << 16) | ...). A simple reinterpret_cast would fail on little-endian machines (like standard x86/x64 PCs).
- **Padding Precision:** The padding logic was the most complex part. A message must be padded correctly for the hash to be valid. The key is to append the '1' bit (0x80), then add 0x00 bytes until the total length in bytes is 56 mod 64 (or 448 mod 512 bits). This leaves the final 8 bytes (64 bits) free for the original message length, which is appended in big-endian order.
- **PDF Constant Typos:** I observed minor typos in the provided constants. The constant for t=0-19 was listed as 5.A82799_16 and for t=20-39 as 6ED9EBA11_W. The implementation used the standard, correct values of 0x5A827999 and 0x6ED9EBA1, respectively, to pass the verification tests.
- **Circular Shift:** The ROTL operation (circular left shift) is not a native C++ operator. It was implemented as a helper function: (value << shift) | (value >> (32 - shift)).

Example Test Cases and Outputs

The implementation was verified against all three test cases provided in the lab manual.

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