**Experiment 4**

**Lavanya Bhambure**

**MITU20BTCS0150 2203083 LYISA 1**

**Title:** MDS5/SHA-1 Algorithm

**Problem Statement:** Implement MD5/SHA-1 algorithms for verifying and maintaining the integrity of information.

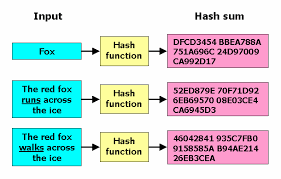
**Aim:** To implement MD5/SHA-1 Algorithm

**Theory:**

MD5 stands for Message Digest and SHA1 stands for Secure Hash Algorithm square measure the hashing algorithms wherever The speed of MD5 is fast compared to SHA1’s speed.

However, SHA1 provides more security than MD5. The construct behind these hashing algorithms is that these square measures generate a novel digital fingerprint of knowledge or message understood as a hash or digest.

|  |  |
| --- | --- |
| **MD5** | **SHA1** |
| Message Digest | Secure Hash Algorithm |
| 128 bit length messages | 160 bit length messages |
| Speed is fast | Speed is slow |
| Comparatively simpler | Comparatively more complex |
| In MD5, if the assailant needs to seek out the 2 messages having identical message digest then assailant would need to perform 2^64 operations. | Whereas in SHA1, assailant would need to perform 2^80 operations which is greater than MD5. |



**Algorithm:**

MD5 Algorithm

1. Initialize variables:
2. Pre-processing (Padding):
   1. Append a '1' bit to the message.
   2. Append '0' bits until the length of the message in bits is congruent to 448 modulo 512.
   3. Append the length of the message (in bits) as a 64-bit little-endian integer.
3. Process the message in 512-bit blocks:
   1. Break the message into 512-bit chunks.
   2. Process each chunk in 64 rounds.
4. Update variables
5. Main loop (64 rounds):
   1. Calculate a new value for A, B, C, and D in each round using bitwise operations and logical functions like XOR, AND, OR, and NOT.
6. Add the computed values to A, B, C, and D.
7. After processing all chunks, the final hash value is A, B, C, D in little-endian order.

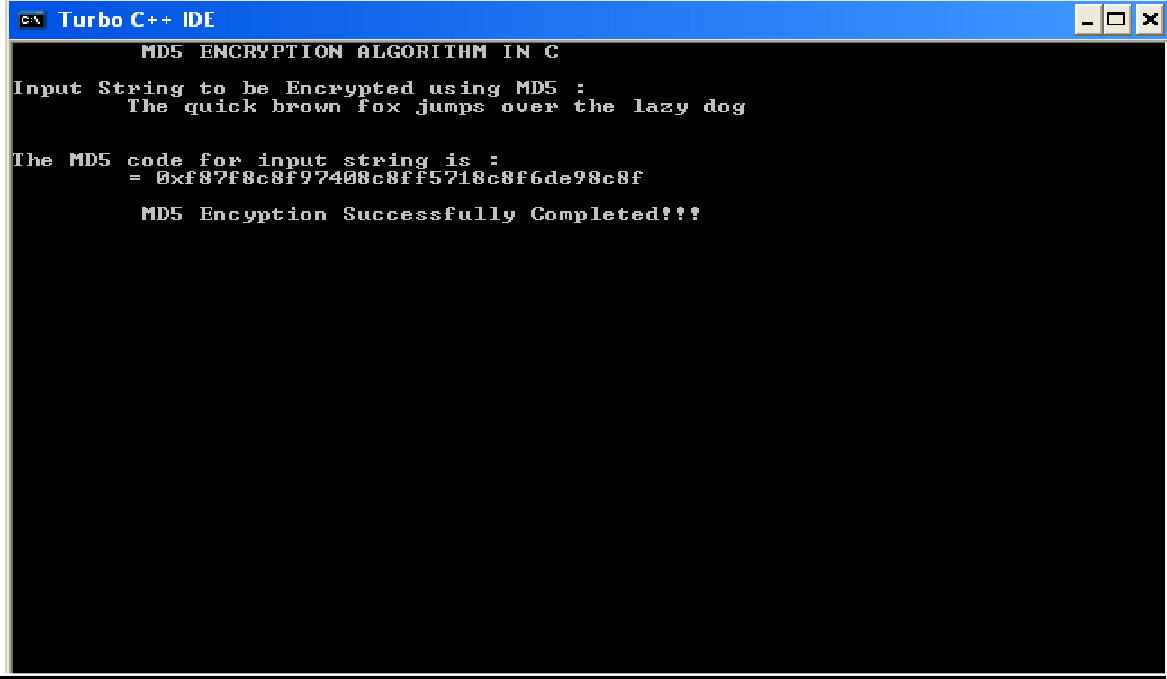
SHA 1 Algorithm

1. Initialize variables
2. Pre-processing (Padding):
   1. Append a '1' bit to the message.
   2. Append '0' bits until the length of the message in bits is congruent to 448 modulo 512.
   3. Append the length of the message (in bits) as a 64-bit little-endian integer.
3. Process the message in 512-bit blocks:
   1. Break the message into 512-bit chunks.
   2. Process each chunk in 80 rounds.
4. Update Variables
5. Main loop (80 rounds):
   1. Calculate a new value for A, B, C, D, and E in each round using bitwise operations and logical functions like XOR, AND, OR, and NOT.
6. Add the computed values to A, B, C, D, and E.
7. After processing all chunks, the final hash value is H0, H1, H2, H3, H4 in big-endian order.

**Code:**

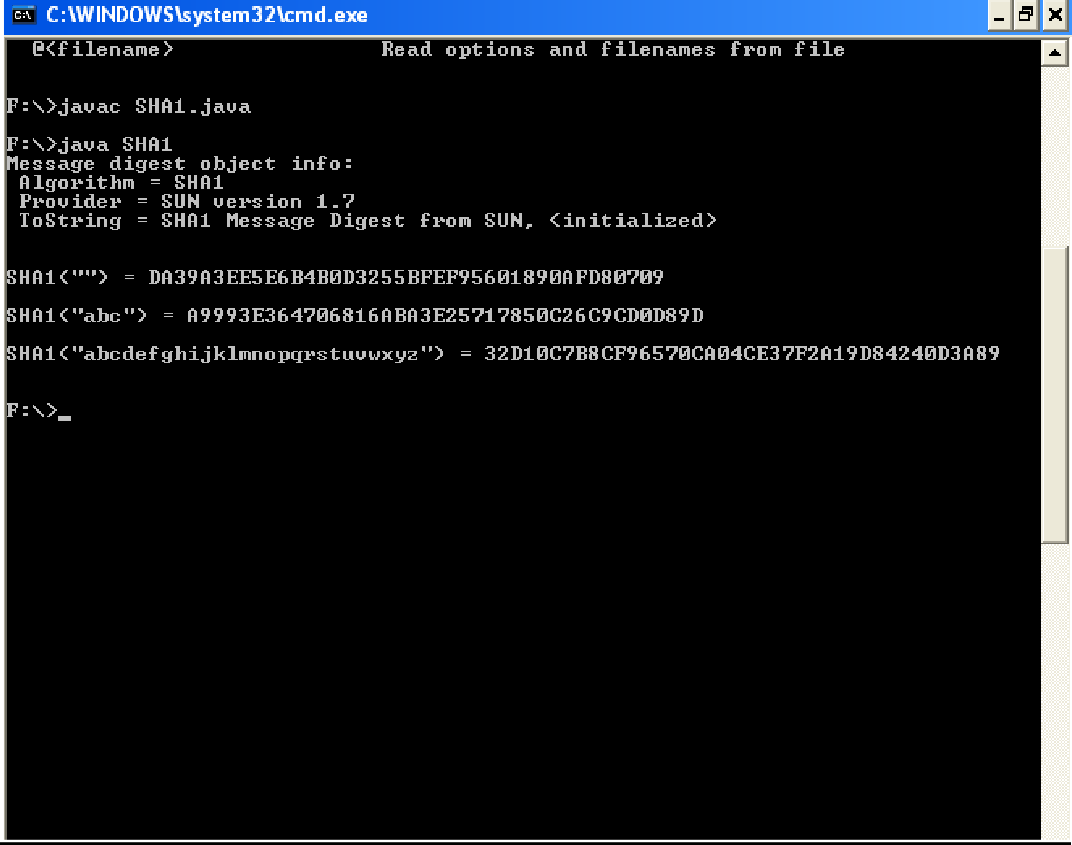
**MD5**

* #include <iostream>
* #include <iomanip>
* #include <sstream>
* #include <string>
* #include <openssl/md5.h>
* std::string md5(const std::string& input) {
* unsigned char digest[MD5\_DIGEST\_LENGTH];
* MD5\_CTX md5Context;
* MD5\_Init(&md5Context);
* MD5\_Update(&md5Context, input.c\_str(), input.size());
* MD5\_Final(digest, &md5Context);
* std::stringstream md5ss;
* for (int i = 0; i < MD5\_DIGEST\_LENGTH; ++i) {
* md5ss << std::hex << std::setw(2) << std::setfill('0') << (int)digest[i];
* }
* return md5ss.str();
* }
* int main() {
* std::string input = "Hello, MD5!";
* std::string md5Hash = md5(input);
* std::cout << "MD5 Hash: " << md5Hash << std::endl;
* return 0;
* }



SHA1 Algoritm:

* import java.security.\*; public class SHA1 {
* public static void main(String[] a) { try {
* MessageDigest md = MessageDigest.getInstance("SHA1"); System.out.println("Message digest object info: "); System.out.println(" Algorithm = " +md.getAlgorithm()); System.out.println(" Provider = " +md.getProvider()); System.out.println(" ToString = " +md.toString()); String input = "";
* md.update(input.getBytes()); byte[] output = md.digest(); System.out.println();
* System.out.println("SHA1(\""+input+"\") =
* +bytesToHex(output)); input = "abc";
* md.update(input.getBytes()); output = md.digest(); System.out.println();
* System.out.println("SHA1(\""+input+"\") = "
* +bytesToHex(output));
* input = "abcdefghijklmnopqrstuvwxyz"; md.update(input.getBytes());
* output = md.digest(); System.out.println();
* System.out.println("SHA1(\"" +input+"\") = "
* +bytesToHex(output)); System.out.println(""); } catch (Exception e) {
* System.out.println("Exception: " +e);
* }
* }
* public static String bytesToHex(byte[] b)
* {
* char hexDigit[] = {'0', '1', '2', '3', '4', '5', '6',
* '7', '8', '9', 'A', 'B', 'C', 'D', 'E', 'F'};
* StringBuffer buf = new StringBuffer(); for (int j=0; j<b.length; j++) {
* buf.append(hexDigit[(b[j] >> 4) & 0x0f]); buf.append(hexDigit[b[j] & 0x0f]); } return buf.toString(); } }



**Applications:**

Both MD5 and SHA-1 are considered weak and insecure for cryptographic purposes due to vulnerabilities that have been discovered over time.

Using these algorithms for cryptographic applications like secure data storage or authentication is strongly discouraged because they can be easily compromised by modern computing resources and techniques. It is recommended to use more secure cryptographic algorithms such as SHA-256, SHA-3, or bcrypt for hashing and encryption in applications where security is a concern.

**Conclusion:**

We have implemented the required algorithm.