CNS LAB

NAME: Parshwa Herwade

PRN: 22510064

BATCH: B1

Experiment No. 09

Title – Calculate the message digest of a text using the SHA-1 algorithm

---

Objectives:

To understand and implement the **SHA-1 (Secure Hash Algorithm 1)** for generating a fixed-length message digest from a given input text. The goal is to demonstrate how data integrity can be maintained and verified using cryptographic hash functions.

Problem Statement:

In digital communications and data storage, ensuring the integrity of data is crucial. One common approach to verifying that a message has not been altered is to compute a **message digest**—a fixed-size string that uniquely represents the input data.

Implement a program that:

1. Accepts a **text input** from the user.
2. Calculates the **SHA-1 hash** (message digest) of the input text using a cryptographic library or custom logic.
3. Outputs the **SHA-1 digest** in **hexadecimal format**.
4. Demonstrates that any change in the input results in a completely different message digest, highlighting the **avalanche effect** of the hash function.

Additionally, briefly explain the role of SHA-1 in real-world applications, such as digital signatures, checksums, and data integrity verification.

Equipment/Tools:

Theory:

Procedure:

Steps:

Observations and Conclusion:

Equipment / Tools

- Java Development Kit (JDK) 8 or later (for MessageDigest).
- Any text editor or IDE (VS Code, IntelliJ, Notepad++).
- Terminal / Command Prompt to compile and run Java.
- (Optional) A web browser for reference about recommended hash functions (SHA-256 / SHA-3) and deprecation notices.

Theory (detailed but concise)

What is SHA-1?

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that maps arbitrary-length input into a fixed 160-bit (20-byte) digest. It's deterministic (same input → same digest), and was designed to provide:

- Preimage resistance (hard to find input from digest),
- Second-preimage resistance (hard to find a different input with same digest),
- Collision resistance (hard to find two different inputs with same digest).

Important: SHA-1 collision attacks have been demonstrated in practice, so SHA-1 is considered **obsolete for collision-resistant uses** (digital signatures, code signing, etc.). Use SHA-256 / SHA-3 / BLAKE2 for modern systems.

High-level SHA-1 algorithm steps

1. **Preprocessing (padding):**
    - Append a single 1 bit to the message.
    - Append k zero bits so that the length (in bits) ≡ 448 (mod 512).
    - Append the original message length as a 64-bit big-endian integer. Now message length is a multiple of 512 bits.
2. **Parse** the padded message into N 512-bit blocks: M[0]...M[N-1].
3. **Initialize five 32-bit words:**
4. H0 = 0x67452301
5. H1 = 0xEFCDAB89
6. H2 = 0x98BADCFE
7. H3 = 0x10325476
8. H4 = 0xC3D2E1F0
9. **For each 512-bit block:**
    - Break it into sixteen 32-bit big-endian words W[0..15].
    - Extend to 80 words: for t = 16..79, W[t] = leftrotate((W[t-3] XOR W[t-8] XOR W[t-14] XOR W[t-16]), 1).
    - Initialize a,b,c,d,e = H0..H4.
    - For t = 0..79:
        - Compute a temporary value temp = leftrotate(a,5) + f_t(b,c,d) + e + W[t] + K_t
            - where f_t and K_t depend on t:
                - $0 \le t \le 19$: f = (b & c) | ((~b) & d), K = 0x5A827999
                - $20 \le t \le 39$: f = b ^ c ^ d, K = 0x6ED9EBA1
                - $40 \le t \le 59$: f = (b & c) | (b & d) | (c & d), K = 0x8F1BBCDC
                - $60 \le t \le 79$: f = b ^ c ^ d, K = 0xCA62C1D6
        - Update e = d; d = c; c = leftrotate(b, 30); b = a; a = temp (mod

2^32).
    - Add a,b,c,d,e to H0..H4 respectively (mod 2^32).
10. **Output:** Concatenate H0||H1||H2||H3||H4 (big-endian) → 160-bit digest, usually represented in hexadecimal.

Properties & limitations

- Output size: 160 bits (hex length 40).
- Avalanche effect: a single-bit change in input causes ≈ half the output bits to flip.
- **Security note:** SHA-1 is **no longer recommended** for collision-resistant applications. Use SHA-2/3.

Procedure (how the program works + how you'll test it)

1. Read a text string from the user (UTF-8).
2. Use Java's MessageDigest.getInstance("SHA-1") to compute the digest bytes.
3. Convert digest bytes to a hexadecimal string and print it.
4. To demonstrate avalanche:
    - Compute digest for original input.
    - Compute digest for a slightly modified input (e.g., change one character or case).
    - Show both hex digests to illustrate the big difference.

CODE:

```
import java.nio.charset.StandardCharsets;
import java.security.MessageDigest;
import java.security.NoSuchAlgorithmException;
import java.util.Scanner;

public class SHA1Demo {

    // convert bytes to hex string
    private static String bytesToHex(byte[] bytes) {
        StringBuilder sb = new StringBuilder(bytes.length * 2);
        for (byte b : bytes) {
            sb.append(String.format("%02x", b & 0xff));
        }
        return sb.toString();
    }

    private static String sha1Hex(String input) throws NoSuchAlgorithmException {
        MessageDigest md = MessageDigest.getInstance("SHA-1");
        byte[] digest = md.digest(input.getBytes(StandardCharsets.UTF_8));
        return bytesToHex(digest);
    }

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in, StandardCharsets.UTF_8.name());
        try {
            System.out.println("SHA-1 Message Digest Calculator (Java)");
            System.out.println("-------------------------------------");
            System.out.print("Enter text: ");
            String text = sc.nextLine();
```

```
            String digest = sha1Hex(text);
            System.out.println("\nInput: \"" + text + "\"");
            System.out.println("SHA-1 digest (hex): " + digest);

            // Avalanche demonstration prompt
            System.out.print("\nEnter a slightly modified version of the text (or
press Enter to auto-modify): ");
            String modified = sc.nextLine();
            if (modified.isEmpty()) {
                // auto modify: flip case of first character if exists, else add 'a'
                if (text.length() > 0) {
                    char c = text.charAt(0);
                    char flipped = Character.isUpperCase(c) ?
Character.toLowerCase(c) : Character.toUpperCase(c);
                    modified = flipped + text.substring(1);
                } else {
                    modified = "a";
                }
                System.out.println("Auto-modified input: \"" + modified + "\"");
            }

            String digest2 = sha1Hex(modified);
            System.out.println("\nModified Input: \"" + modified + "\"");
            System.out.println("SHA-1 digest (hex): " + digest2);

            // quick percent of differing hex chars (informal measure)
            int diffs = 0;
            int len = Math.min(digest.length(), digest2.length());
            for (int i = 0; i < len; i++) if (digest.charAt(i) != digest2.charAt(i))
diffs++;
            System.out.printf("\nHex characters different: %d out of %d (%.1f%%)\n",
diffs, len, 100.0 * diffs / len);

            // show some known test vectors (optional)
            System.out.println("\nSome known SHA-1 digests:");
            System.out.println("  \"\" ->
da39a3ee5e6b4b0d3255bfef95601890afd80709");
            System.out.println("  \"hello world\" ->
2aae6c35c94fcfb415dbe95f408b9ce91ee846ed");
            System.out.println("  \"The quick brown fox jumps over the lazy dog\" ->
2fd4e1c67a2d28fced849ee1bb76e7391b93eb12");

        } catch (NoSuchAlgorithmException e) {
            System.err.println("SHA-1 algorithm not available: " + e.getMessage());
        } finally {
            sc.close();
        }
    }
}
```

RESULTS:

```
PS C:\Users\ASUS\Desktop\CNSL\22510076_CNS_A9> javac SHA1Demo.java
>>
PS C:\Users\ASUS\Desktop\CNSL\22510076_CNS_A9> java SHA1Demo
>>
SHA-1 Message Digest Calculator (Java)
-------------------------------------
Enter text: hello world

Input: "hello world"
SHA-1 digest (hex): 2aae6c35c94fcfb415dbe95f408b9ce91ee846ed

Enter a slightly modified version of the text (or press Enter to auto-modify): Hello World

Modified Input: "Hello World"
SHA-1 digest (hex): 0a4d55a8d778e5022fab701977c5d840bbc486d0

Hex characters different: 37 out of 40 (92.5%)

Some known SHA-1 digests:
  "" -> da39a3ee5e6b4b0d3255bfef95601890afd80709
  "hello world" -> 2aae6c35c94fcfb415dbe95f408b9ce91ee846ed
  "The quick brown fox jumps over the lazy dog" -> 2fd4e1c67a2d28fced849ee1bb76e7391b93eb12
```

Steps (what you will document / report)

1. Explain hash functions and properties (preimage, second-preimage, collision resistance).
2. Describe SHA-1 preprocessing & compression steps (padding, 512-bit blocks, W[0..79], constants).
3. Present the Java program (include code and short explanation of MessageDigest usage).
4. Show program compilation and execution commands.
5. Present test cases and outputs (including empty string and known vectors).
6. Demonstrate avalanche: show original vs slightly modified input digests and quantify difference.
7. Discuss security implications and recommended alternatives.

Observations and Conclusion

Observations

- The SHA-1 digest is always 160 bits (40 hex chars) regardless of input length.
- Small changes in the input (changing h to H, or adding a .) produce completely different digests — illustrating the **avalanche effect**.
- Known test vectors (empty string, “hello world”, “The quick brown fox...”) match standard results — verifying correctness of implementation.
- Using Java's built-in MessageDigest is reliable, efficient and uses native optimized code.

Security Conclusion

- SHA-1 works exactly as a hash function — it reliably converts messages to fixed-size digests and demonstrates the avalanche property.
- However, **SHA-1 should not be used** for collision-sensitive security tasks (code signing, certificate signatures, etc.) because practical collision attacks exist. Use SHA-256 (part of SHA-2) or SHA-3 for real-world systems.
- For integrity checks (non-security critical) SHA-1 can still be used for legacy compatibility, but migration to stronger hashes is strongly recommended.