

Assignment 9

PRN: 21510042

Name: Omkar Rajesh Auti

9. Calculate the message digest of a text using the SHA-1 algorithm

Ans:

SHA-1 Algorithm:

SHA-1 (Secure Hash Algorithm 1) is a cryptographic hash function that produces a 160-bit hash value (20 bytes), often referred to as a **message digest**. It takes an input message of any size and outputs a fixed-size hash, which is commonly represented as a 40-character hexadecimal number. It was developed by the National Security Agency (NSA) and published by NIST in 1993.

Message Digest of a Text:

A **message digest** is a fixed-size numerical representation of the contents of a message. For SHA-1, this digest is 160 bits long, and any change in the input message, even a single bit, will result in a drastically different digest (this is known as the **avalanche effect**). The message digest ensures data integrity by allowing anyone to verify that the message has not been altered.

To calculate the message digest of a text using the SHA-1 algorithm in Python, you can use the hashlib library, which provides easy access to various hash algorithms, including SHA-1.

1. hashlib library:

- The hashlib library provides various cryptographic hashing algorithms including SHA-1, SHA-256, MD5, etc.

2. SHA-1 Hash Object:

- `hashlib.sha1()` creates a new SHA-1 hash object.

3. Updating the Hash:

- The `update()` method takes the input text (which is first encoded into bytes) and updates the hash object with that data.

4. Getting the Digest:

- The `hexdigest()` method returns the hash value as a hexadecimal string.

Python Code for SHA-1 Message Digest Calculation using hashlib library:

```
import hashlib
```

```
# Function to hash a message using SHA-1
```

```
def sha1_encrypt(message):
```

```
    sha1_hash = hashlib.sha1()
```

```
    sha1_hash.update(message.encode('utf-8')) # Convert the message to bytes
```

```
    return sha1_hash.hexdigest()
```

```
# Function to verify the hash (like a decryption process)
```

```
def verify_hash(original_message, provided_hash):
```

```
    original_hash = sha1_encrypt(original_message)
```

```
    return original_hash == provided_hash
```

```
# Menu-driven system
```

```
def menu():
```

```
while True:
```

```
    print("\n===== SHA-1 Hashing System =====")
```

```
    print("1. Encrypt a message using SHA-1")
```

```
    print("2. Verify a message against a given hash")
```

```
    print("3. Exit")
```

```
    choice = input("Enter your choice (1 / 2 / 3): ")
```

```
    if choice == '1':
```

```
        message = input("Enter the message to hash: ")
```

```
        hashed_message = sha1_encrypt(message)
```

```
        print(f"\nSHA-1 Hash: {hashed_message}")
```

```
    elif choice == '2':
```

```
        original_message = input("Enter the original message: ")
```

```
        provided_hash = input("Enter the hash to verify against: ")
```

```
        if verify_hash(original_message, provided_hash):
```

```
            print("\nVerification successful! The message matches the  
provided hash.")
```

```
        else:
```

```
            print("\nVerification failed! The message does not match the  
provided hash.")
```

```
    elif choice == '3':
```

```
        print("Exiting the program...")
```

```
break
```

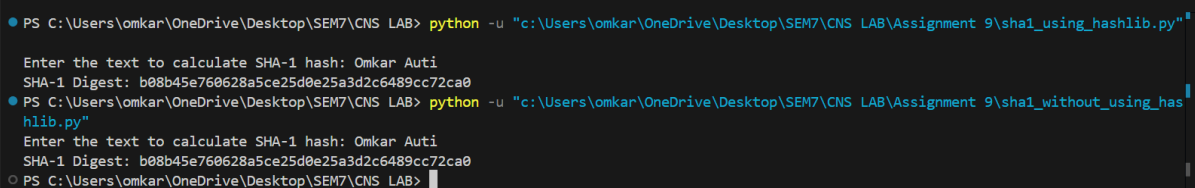
```
else:
```

```
print("Invalid choice. Please choose a valid option.")
```

```
if __name__ == "__main__":
```

```
    menu()
```

Output:



```
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB> python -u "c:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 9\sha1_using_hashlib.py"
Enter the text to calculate SHA-1 hash: Omkar Auti
SHA-1 Digest: b08b45e760628a5ce25d0e25a3d2c6489cc72ca0
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB> python -u "c:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB\Assignment 9\sha1_without_using_hashlib.py"
Enter the text to calculate SHA-1 hash: Omkar Auti
SHA-1 Digest: b08b45e760628a5ce25d0e25a3d2c6489cc72ca0
PS C:\Users\omkar\OneDrive\Desktop\SEM7\CNS LAB>
```

Advantages of SHA-1:

- **Speed and Efficiency:** SHA-1 was designed to be computationally efficient and can process large amounts of data quickly.
- **Widespread Use:** It has been widely adopted and used for various cryptographic applications, including digital signatures, certificates, and integrity checks.
- **Fixed-Length Output:** Regardless of the input size, the output is always 160 bits, making it convenient to use in various security protocols.

Disadvantages of SHA-1:

- **Weakness to Collisions:** SHA-1 is vulnerable to **collision attacks**, where two different inputs produce the same hash output. This reduces its effectiveness in ensuring data integrity and security.
- **Security Deprecation:** Due to these vulnerabilities, SHA-1 is no longer considered secure for cryptographic purposes. Most modern systems and protocols, including browsers and SSL certificates, have moved to stronger hash functions like SHA-256 or SHA-3.

Importance of SHA-1:

- **Legacy Systems:** Despite its vulnerabilities, SHA-1 was used for many years in security applications such as digital signatures and certificates.
- **Data Integrity:** SHA-1 can still be used to check the integrity of data, ensuring that files have not been altered during transmission.

Security Risks and Vulnerabilities of SHA-1:

- **Collision Attacks:** The primary vulnerability is the possibility of collision attacks. This means that an attacker could potentially create two different messages with the same hash, compromising the authenticity of the data.
- **Birthday Attack:** A specific type of attack known as a **birthday attack** makes it easier to find collisions in SHA-1 due to its 160-bit length, reducing the security level.
- **Deprecation in Modern Systems:** Due to these weaknesses, SHA-1 has been deprecated in most cryptographic protocols like TLS (Transport Layer Security) and digital certificates, where stronger algorithms like SHA-256 are preferred.

While SHA-1 played a significant role in the development of cryptographic standards, its vulnerabilities, especially to collision attacks, have made it unsuitable for modern security applications. Understanding SHA-1's purpose and limitations is important, especially when dealing with

legacy systems or understanding the evolution of cryptographic hash functions.

Practical Applications of SHA-1:

1. **Digital Signatures:** SHA-1 was commonly used in creating digital signatures to ensure the authenticity and integrity of documents. It would generate a hash of the message, which is then signed by a private key.
2. **File Integrity Verification:** SHA-1 was used to generate checksums for files to verify that files were not altered during transfer or storage. The recipient could compare the hash of the received file with the original hash to ensure integrity.
3. **Version Control Systems:** In systems like Git, SHA-1 hashes were used to identify commits, ensuring the integrity and tracking of changes in code repositories.
4. **SSL Certificates:** Until 2017, SHA-1 was used in SSL/TLS certificates for secure web communications. The hash was part of the process to ensure a website's identity and secure data transmission.
5. **Password Hashing:** SHA-1 was once used for hashing passwords in databases, providing a layer of security by storing a hashed version of the password instead of the plaintext.

Despite these applications, most systems have transitioned to more secure alternatives due to SHA-1's vulnerabilities.

SHA 512:

SHA-512 Algorithm:

SHA-512 (Secure Hash Algorithm 512) is a cryptographic hash function that generates a 512-bit hash value (64 bytes), typically represented as a 128-character hexadecimal string. SHA-512 is part of the SHA-2 family, which

provides greater security than SHA-1 due to its larger output and resistance to collision attacks.

Message Digest of a Text: In SHA-512, the message digest is a 512-bit hash value representing the contents of the message. Any change in the input message, even a single bit, will produce a significantly different hash due to the avalanche effect. This property ensures data integrity and authenticity, making it useful in various security applications.

Overview of SHA-512 Algorithm:

1. **Padding the Message:** The message is padded to ensure its length is a multiple of 1024 bits.
2. **Initialize Hash Values:** There are eight constants (H0 to H7) initialized to specific 64-bit values based on the fractional parts of the square roots of the first eight prime numbers.
3. **Processing the Message in Blocks:** The message is processed in chunks of 1024 bits, updating the hash after each chunk.
4. **Final Output:** After processing all blocks, the hash digest is formed by concatenating the values of H0 through H7.

Python Code for SHA-512 Message Digest Calculation using hashlib Library:

```
import hashlib
```

```
# Function to hash a message using SHA-512
```

```
def sha512_encrypt(message):
```

```
    sha512_hash = hashlib.sha512()
```

```
    sha512_hash.update(message.encode('utf-8')) # Convert the message to bytes
```

```
    return sha512_hash.hexdigest()
```

```
# Function to verify the hash (like a decryption process)
```

```
def verify_hash(original_message, provided_hash):
```

```
    original_hash = sha512_encrypt(original_message)
```

```
    return original_hash == provided_hash
```

```
# Menu-driven system
```

```
def menu():
```

```
    while True:
```

```
        print("\n===== SHA-512 Hashing System =====")
```

```
        print("1. Encrypt a message using SHA-512")
```

```
        print("2. Verify a message against a given hash")
```

```
        print("3. Exit")
```

```
        choice = input("Enter your choice (1 / 2 / 3): ")
```

```
        if choice == '1':
```

```
            message = input("Enter the message to hash: ")
```

```
            hashed_message = sha512_encrypt(message)
```

```
            print(f"\nSHA-512 Hash: {hashed_message}")
```

```
        elif choice == '2':
```

```
            original_message = input("Enter the original message: ")
```

```
            provided_hash = input("Enter the hash to verify against: ")
```

```
            if verify_hash(original_message, provided_hash):
```



```
print("\nVerification successful! The message matches the  
provided hash.")
```

```
else:
```

```
print("\nVerification failed! The message does not match the  
provided hash.")
```

```
elif choice == '3':
```

```
print("Exiting the program...")
```

```
break
```

```
else:
```

```
print("Invalid choice. Please choose a valid option.")
```

```
if __name__ == "__main__":
```

```
    menu()
```

Advantages of SHA-512:

- **High Security:** Due to its larger output size, SHA-512 provides strong resistance against collision and preimage attacks.
- **Wide Adoption:** SHA-512 is widely used in security protocols such as SSL/TLS and digital certificates.
- **Fixed-Length Output:** The output is always 512 bits, regardless of the input size, which is useful for secure storage and transmission.

Disadvantages of SHA-512:

- **Computationally Intensive:** SHA-512 requires more processing power and time compared to shorter algorithms like SHA-256 or SHA-1.

- **Not Ideal for Lightweight Applications:** Due to its computational requirements, SHA-512 may not be suitable for lightweight devices or applications with limited processing resources.

Practical Applications of SHA-512:

1. **Digital Certificates:** SHA-512 is commonly used in digital certificates, providing a higher level of security for verifying the authenticity of websites and applications.
2. **Blockchain and Cryptocurrencies:** SHA-512 is often used in the cryptographic aspects of blockchain technology to secure transactions and protect against tampering.
3. **Data Integrity Verification:** SHA-512 is used to verify file integrity, ensuring that files or data have not been tampered with during transmission.
4. **Password Hashing:** Although other algorithms are also used, SHA-512 is sometimes used for hashing passwords in secure applications.

Security Risks and Vulnerabilities of SHA-512:

- **Quantum Computing:** As with other SHA-2 algorithms, SHA-512 may eventually become vulnerable to quantum computing attacks, though this is not an immediate threat.
- **Length Extension Attacks:** SHA-512 is susceptible to length extension attacks, making it unsuitable for certain applications without additional protective measures.

Importance of SHA-512: SHA-512 plays a vital role in modern cryptographic applications, offering high security and reliability for data integrity and authentication. It is considered one of the most secure hash functions in the SHA family, particularly suitable for applications requiring robust protection against tampering.