

Name of Student: Pushkar Sane		
Roll Number: 45		Lab Assignment Number: 3
Title of Lab Assignment: Implement SHA256 Algorithm.		
DOP: 13-08-2024		DOS: 21-08-2024
CO Mapped:	PO Mapped:	Signature:

Practical No. 3**Aim: Implement SHA256 algorithm****Theory:**

Hashing is the process of converting input data (or a "message") into a fixed-size string of bytes. This fixed-size output is typically a sequence of numbers and letters, known as a hash value, hash code, or simply hash. The function used to perform this conversion is called a hash function.

1. Deterministic: The same input will always produce the same hash value.
2. Fixed Output Length: Regardless of the input size, the output (hash value) is always of a fixed length.
3. Efficient Computation: The hash function should be able to produce the hash value quickly.
4. Preimage Resistance: It should be computationally infeasible to reverse the hash function to retrieve the original input from the hash value.
5. Collision Resistance: It should be difficult to find two different inputs that produce the same hash value.
6. Avalanche Effect: A small change in the input should produce a significantly different hash value.

Common Uses of Hashing

1. Cryptographic Applications:
 - Password Hashing: Passwords are hashed before storage in databases. During login, the input password is hashed and compared to the stored hash, protecting the actual password from exposure.
 - Digital Signatures: Hash functions are used to create a unique hash of a document. This hash is then encrypted to form a digital signature, which can be used to verify the authenticity and integrity of the document.
2. **Data Integrity Verification:**
 - Checksums: Hashes are used in checksums to detect errors or modifications in data during transmission or storage. For example, software distributions often include hash values so users can verify the integrity of the downloaded files.

3. Unique Identifiers:

- Unique IDs: Hash functions are used to create unique identifiers for objects, such as database keys or session tokens.

Example of Hash Functions

1. MD5 (Message Digest Algorithm 5):

- Produces a 128-bit hash value, often rendered as a 32-character hexadecimal number.
- Example: Hash of "hello" is 5d41402abc4b2a76b9719d911017c592.

2. SHA-1 (Secure Hash Algorithm 1):

- Produces a 160-bit hash value, often rendered as a 40-character hexadecimal number.
- Example: Hash of "hello" is f7c3bc1d808e04732adf679965ccc34c.

3. SHA-256 (Secure Hash Algorithm 256-bit):

- Part of the SHA-2 family, produces a 256-bit hash value.
- Example: Hash of "hello" is 2cf24dba5fb0a30e26e83b2ac5b0e8a0.

Short Explanation about SHA256

SHA-256 (Secure Hash Algorithm 256-bit) is a cryptographic hash function that produces a 256-bit (32-byte) hash value, often represented as a 64-character hexadecimal string. It is part of the SHA-2 family of hash functions, which was designed to improve upon the security of the older SHA-1 algorithm.

Key Characteristics of SHA-256

1. Fixed Output Length:

- Regardless of the input size, the output hash value is always 256 bits (32 bytes) long.

2. Deterministic:

- The same input will always produce the same hash value.

3. Pre-image Resistance:

- It is computationally infeasible to reverse the hash function and retrieve the original input from the hash value.

4. Collision Resistance:

- It is difficult to find two different inputs that produce the same hash value.

5. Avalanche Effect:

- A small change in the input drastically changes the hash value.

Example Usage

SHA-256 is commonly used in:

- **Digital Signatures:** To ensure data integrity and authenticity.
- **Password Hashing:** To securely store passwords.
- **Data Integrity Verification:** To check if data has been altered.

Explanation of Python libraries and methods used for implementing the program

In the context of implementing a program like the RSA encryption and decryption example provided earlier, several Python libraries and methods are used. Here's an explanation of the relevant libraries and their methods:

Python Libraries and Methods

1. math Library

- **Purpose:** Provides mathematical functions.
- **Method Used:**
 - `gcd(a, b)`: Calculates the greatest common divisor of `a` and `b`. In the RSA algorithm, it's used to find a public exponent `e` that is coprime with the totient `t`.

```
python
from math import gcd
```

2. hashlib Library

- **Purpose:** Provides hashing algorithms for cryptographic purposes.
- **Method Used:**
 - `hashlib.sha256()`: Creates a new SHA-256 hash object. This is used to generate a SHA-256 hash of a message, ensuring data integrity and security.

Example:

```
import hashlib
sha256 = hashlib.sha256()
sha256.update(message.encode())
hash_value = sha256.hexdigest()
```

3. Python Built-in Functions

- **Purpose:** Built-in functions provide fundamental capabilities in Python.
- **Methods Used:**
 - `input(prompt)`: Reads a line from input, used to get user input.
 - `int()`: Converts a string to an integer, used to ensure numeric inputs.
 - `print()`: Outputs data to the console.

Example:

```
p = int(input("Enter the first prime number (p): "))
q = int(input("Enter the second prime number (q): "))
message = int(input("Enter the message (integer): "))
```

4. Control Flow Statements

- **Purpose:** Direct the flow of execution based on conditions.
- **Statements Used:**
 - `if`: Conditional statements to execute code blocks based on conditions.
 - `while`: Loops that continue execution as long as a condition is true.

Example:

```
if p <= 1 or q <= 1:
    print("Both p and q must be greater than 1.")
```

5. Custom Functions

- **Purpose:** Encapsulate code into reusable components.
- **Functions Defined:**
 - `RSA(p, q, message)`: Implements the RSA algorithm for encryption and decryption.
 - `main()`: The main function to handle user input and call the `RSA` function.
 - `is_prime(num)`: Checks if a number is prime, used for RSA key validation.

Example:

```
def RSA(p: int, q: int, message: int):
    # RSA algorithm implementation

def main():
    # Input handling and RSA function call
```

```
def is_prime(num):  
    # Prime checking logic
```

1. **User Input Handling:** Uses input() to gather values for p, q, and message, and int() to convert these inputs into integers.

```
python
```

```
Copy code
```

```
p = int(input("Enter the first prime number (p): "))
```

2. **Hashing (if used):** Though not directly in the RSA example, if you were using hashlib, you'd use it to hash messages for security purposes.

```
python
```

```
Copy code
```

```
sha256 = hashlib.sha256()
```

```
sha256.update(message.encode())
```

```
hash_value = sha256.hexdigest()
```

3. **Custom Logic:** Includes RSA for performing encryption and decryption, and is_prime to validate prime numbers used in the RSA algorithm.

```
Example:
```

```
# Implement RSA encryption and decryption
```

```
def RSA(p: int, q: int, message: int):
```

```
# Check if num is a prime number
```

```
def is_prime(num):
```

Code:

```
import hashlib
```

```
str1 = input("Enter a string")
```

```
encodedstr = str1.encode()
```

```
result = hashlib.sha256(encodedstr)
```

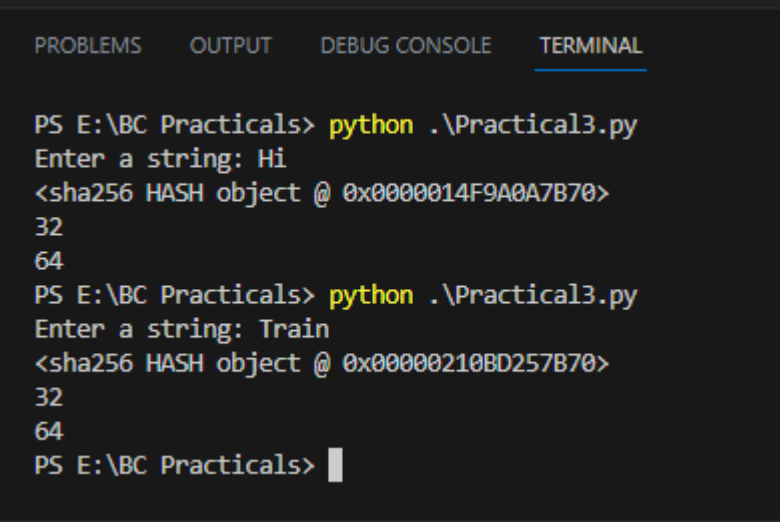
```
print(result)
```

```
# result in hexadecimal format
```

```
result.hexdigest()
```

```
print(result.digest_size)
```

```
print(result.block_size)
```

Output:

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL

PS E:\BC Practicals> python .\Practical3.py
Enter a string: Hi
<sha256 HASH object @ 0x0000014F9A0A7B70>
32
64
PS E:\BC Practicals> python .\Practical3.py
Enter a string: Train
<sha256 HASH object @ 0x00000210BD257B70>
32
64
PS E:\BC Practicals> 
```

Conclusion:

Successfully demonstrated the implementation of SHA256 algorithm.