Experiment No. 4: Analyse the performance of the MD-5/SHA-1 protocol for varying message sizes, test integrity of message

Aim: To analyse the performance of the MD-5/SHA-1 protocol for varying message sizes, test integrity of message

Theory:

MD5 (Message Digest algorithm 5) is a widely used cryptographic hash function with a 128 bit hash value. An MD5 hash is typically expressed as a 32 digit hexadecimal number. MD5 processes a variable length message into a fixed length output of 128 bits. The input message is broken up into chunks of 512 bit blocks (sixteen 32bit little endian integers) ; The message is padded so that its length is divisible by 512. The padding works as follows: first a single bit, 1, is appended to the end of the message. This is followed by as many zeros as are required to bring the length of the message up to 64 bits less than a multiple of 512. The remaining bits are filled up with a 64bit integer representing the length of the original message, in bits.

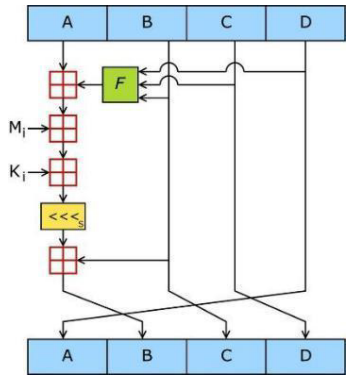


Figure 1: One MD5 operation. MD5 consists of 64 of these operations, grouped in four rounds of 16 operations. *F* is a nonlinear function; one function is used in each round. M_i denotes a 32bit block of the message input, and K_i denotes a 32bit constant, different for each operation.The main MD5 algorithm operates on a 128bit state, divided into four 32bit words, denoted *A*, *B*, *C* and *D*. These are initialized to certain fixed constants. The main algorithm then operates on each 512bit message block in turn, each block modifying the state. The processing of a message block consists of four similar stages, termed *rounds*; each round is

composed of 16 similar operations based on a nonlinear function *F*, modular addition, and left rotation.

Figure 1 illustrates one operation within a round. There are four possible functions *F*; a different one is used in each round:

$F(X,Y,Z) = (X \quad Y) \quad (\quad X \quad Z)$

$G(X,Y,Z) = (X \quad Z) \quad (Y \quad Z)$

$I(X,Y,Z) = Y \quad (X \quad Z)$

, , ,denote the XOR, AND, OR and NOT operations respectively.

Algorithm:

1. Append Padding Bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512. Padding is performed as follows: a single "1" bit is appended to the message, and then "0" bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512. In all, at least one bit and at most 512 bits are appended.

2. Append Length

A 64 bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low order 64 bits of b are used. (These bits are appended as two 32bit words and appended low-order word first in accordance with the previous conventions.) At this point the resulting message (after padding with bits and with b) has a length that is an exact

multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32 bit) words. Let M[0 ... N1] denote the words of the resulting message, where N is a multiple of 16.

3. Initialize MD Buffer

A fourword buffer (A,B,C,D) is used to compute the message digest. Here each of A, B, C, D is a 32bit register. These registers are initialized to the following values in hexadecimal, loworder bytes first):

4. Process Message in 16Word Blocks

We first define four auxiliary functions that each take as input three 32bit words and produce as output one 32bit word.

5. Output

The message digest produced as output is A, B, C, D. That is, we begin with the low order byte of A, and end with the highorder byte of D.

Source Code:

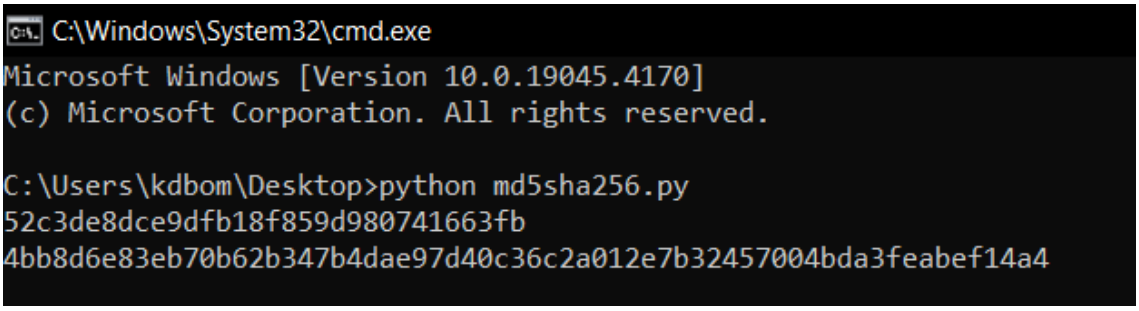
```
import hashlib

str = "vidayvardhini"
res_md5 = hashlib.md5(str.encode())
print(res_md5.hexdigest())

str1 = "vidayvardhini"
res_sha256 = hashlib.sha256(str1.encode())
```

```
print(res_sha256.hexdigest())
```

Output:



Conclusion:

In conclusion, the code computes the MD5 and SHA-256 hash values for the string "vidayvardhini" using Python's hashlib library. These hash values serve as unique representations of the input string, commonly used for data integrity verification and security purposes.