Experiment No. 6

Aim: For varying message sizes, generate message digest using MD5 algorithm and check the integrity of message

Theory:

There are two main kinds of encryption algorithms: Symmetric Key Algorithm (SKA) and Asymmetric Key Algorithm (AKA). In additional, there is also another assistant algorithm called: Hash, which compresses message of any length to certain fixed length (message-digest). This process is irreversible. Hash function can be used in many fields such as: digital signature, message integrity test, and message originality etc. Hash algorithm mainly includes: MD×(Message－Digest Algorithm), SHA×(Secure Hash Algorithms), N-Hash, RIPE-MD, and HAVAL etc.

MD5 Algorithm:

MD5 message-digest algorithm is the 5th version of the Message-Digest Algorithm developed by Ron Rivest to produce a 128-bit message digest. MD5 is quite fast than other versions of the message digest, which takes the plain text of 512-bit blocks, which is further divided into 16 blocks, each of 32 bit and produces the 128-bit message digest, which is a set of four blocks, each of 32 bits. MD5 produces the message digest through five steps, i.e. padding, append length, dividing the input into 512-bit blocks, initialising chaining variables a process blocks and 4 rounds, and using different constant it in each iteration.

Use of MD5 Algorithm

It was developed with the main motive of security as it takes an input of any size and produces an output if a 128-bit hash value. To be considered cryptographically secure, MD5 should meet two requirements:

1. It is impossible to generate two inputs that cannot produce the same hash function.

2. It is impossible to generate a message having the same hash value.

Initially, MD5 was developed to store one way hash of a password, and some file servers also provide pre-computed MD5 checksum of a file so that the user can compare the checksum of the downloaded file to it. Most Unix based Operating Systems include MD5 checksum utilities in their distribution packages.

MD5 produces an output of 128-bit hash value. This encryption of input of any size into hash values undergoes 5 steps, and each step has its predefined task.

Step1: Append Padding Bits

- Padding means adding extra bits to the original message. So in MD5 original message is padded such that its length in bits is congruent to 448 modulo 512. Padding is done such that the total bits are 64 less, being a multiple of 512 bits length.
- Padding is done even if the length of the original message is already congruent to 448 modulo 512. In padding bits, the only first bit is 1, and the rest of the bits are 0.

Step 2: Append Length

After padding, 64 bits are inserted at the end, which is used to record the original input length. Modulo 2^64. At this point, the resulting message has a length multiple of 512 bits.

Step 3: Initialize MD buffer.

A four-word buffer (A, B, C, D) is used to compute the values for the message digest. Here A, B, C, D are 32- bit registers and are initialized in the following way

Word A	01	23	45

Word B	89	Ab	Cd
Word C	Fe	Dc	Ba
Word D	76	54	32

Step 4: Processing message in 16-word block

MD5 uses the auxiliary functions, which take the input as three 32-bit numbers and produce 32-bit output. These functions use logical operators like OR, XOR, NOR.

F(X, Y, Z)	XY v not (X)Z
G(X, Y, Z)	XZ v Y not (Z)
H(X, Y, Z)	X xor Y xor Z
I(X, Y, Z)	Y xor (X v not (Z))

The content of four buffers are mixed with the input using this auxiliary buffer, and 16 rounds are performed using 16 basic operations.

Output-

After all, rounds have performed, the buffer A, B, C, D contains the MD5 output starting with lower bit A and ending with higher bit D.

Below are the advantages and disadvantages:

- MD5 Algorithms are useful because it is easier to compare and store these smaller hashes than store a large variable length text. It is a widely used algorithm for one-way hashes used to verify without necessarily giving the original value. Unix systems use the MD5 Algorithm to store the passwords of the user in a 128-bit encrypted format. MD5 algorithms are widely used to check the integrity of the files.
- Moreover, it is very easy to generate a message digest of the original message using this algorithm. It can perform the message digest of a message having any number of bits; it is not limited to a message in the multiples of 8, unlike MD5sum, which is limited to octets.
- But for many years, MD5 has prone to hash collision weakness, i.e. it is possible to create the same hash function for two different inputs. MD5 provides no security over these collision attacks. Instead of MD5, SHA (Secure Hash Algorithm, which produces 160-bit message digest and designed by NSA to be a part of digital signature algorithm) is now acceptable in the cryptographic field for generating the hash function as it is not easy to produce SHA-I collision and till now no collision has been produced yet.
- Moreover, it is quite slow then the optimized SHA algorithm. SHA is much secure than the MD5 algorithm, and moreover, it can be implemented in existing technology with exceeding rates, unlike MD5. Nowadays, new hashing algorithms are coming up in the market, keeping in mind higher security of data like SHA256 (which generates 256 bits of signature of a text).

Implementation:

```
import hashlib
class MD5:
    def __init__(self):
        self.__keys = ["Adnan is a human", "Humans are mamals", "Adnan is a mamal"]
        self.digest_keys = [self.digest(key) for key in self.__keys]
        for key, digest_key in zip(self.__keys,self.digest_keys):
            print(f"key = {key}\nEquivalent MD5 hexadecimal digest = {digest_key}\n")
    def digest(self,string):
        return hashlib.md5(string.encode()).hexdigest()
    def check_in_digest_keys(self,string):
        return self.digest(string) in self.digest_keys
    def digest_strings(self,strings):
        for x in strings:
            print(f"Given string: {x}\nMD5 hexadecimal digest value of given string: {self.digest(x)}")
            print(f"present in digest_keys {self.check_in_digest_keys(x)}\n")
obj = MD5()
given_strings = [
                 "Adnan is a human", "Adnan is not a human",
                 "Humans are reptile", "Humans are mamals",
                 "Adnan is a mamal", "Adnan is a reptile"
                ]
obj.digest_strings(given_strings)
```

Output:

```
key = Adnan is a human
Equivalent MD5 hexadecimal digest = a235b8e8f838ea6096318e4e3cdfb920

key = Humans are mamals
Equivalent MD5 hexadecimal digest = 5a55ebe77084688bf175b64fe537334c

key = Adnan is a mamal
Equivalent MD5 hexadecimal digest = 2dddc6e1eec9674281626983e03798dd

Given string: Adnan is a human
MD5 hexadecimal digest value of given string: a235b8e8f838ea6096318e4e3cdfb920
present in digest_keys True

Given string: Adnan is not a human
MD5 hexadecimal digest value of given string: 959be596779c379e1116378ac2820539
present in digest_keys False

Given string: Humans are reptile
MD5 hexadecimal digest value of given string: b55d5122a5da1821b8d3fbb0e01bfec0
present in digest_keys False

Given string: Humans are mamals
MD5 hexadecimal digest value of given string: 5a55ebe77084688bf175b64fe537334c
present in digest_keys True

Given string: Adnan is a mamal
MD5 hexadecimal digest value of given string: 2dddc6e1eec9674281626983e03798dd
present in digest_keys True

Given string: Adnan is a reptile
MD5 hexadecimal digest value of given string: b11741565fcdde9ae5286280a9f355a8
present in digest_keys False
```