Crypto Lab – One-Way Hash Function and MAC

- ## Task 1: Generating Message Digest and MAC

  The digest functions output the message digest of a supplied file or files in hexadecimal. The digest functions also generate and verify digital signatures using message digests.The generic name, dgst, may be used with an option specifying the algorithm to be used. The default digest is sha256.

  The output of different hash algorithm when used with openssl is shown in figure below

```
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -md5 file.txt
MD5(file.txt)= 728614d40107e56c90152d2271905bae
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -sha1 file.txt
SHA1(file.txt)= 48d93825aafedc2e639eaab02ca9ba4abdc8e2a9
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -sha256 file.txt
SHA256(file.txt)= 2678385d52102e22cb08c781d257b59101e00d87ec4662a4cd0f804afc402d
b0
```

- It can be observed that the length of hash key written by md5 is smaller in length when compared to that of sha1 and sha256.
- Then when compared to sha1 and sha256, the length of hash of sha256 is larger than that of sha1 whichindicates that sha256 is more seccure than sha1.
- when compared to md5 and sha1/256, sha1/256 is more stronger.

- ## Task 2: Keyed Hash and HMAC

  In cryptography, a keyed hash message authentication code (HMAC) is a specific type of message authentication code (MAC) involving a cryptographic hash function(hence the ‘H’) in combination with a secret cryptographic key. Any cryptographic hash function, such as MD5 or SHA-1, may be used in the calculation of an HMAC; the resulting MAC algorithm is termed HMAC-MD5 or HMAC-SHA1 accordingly.

  The output of different hash algorithm when used with openssl, hmac and key is shown in figure below

```
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -md5 -hmac "abcdef" file.txt
HMAC-MD5(file.txt)= 9d434825b97329d0fcf1bf984ea71a5b
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -md5 -hmac "1245678" file.txt
HMAC-MD5(file.txt)= c50bae1f7a70e25db2493d1219683645
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -sha1 -hmac "abcdef" file.txt
HMAC-SHA1(file.txt)= 4b53cc9b022c33903d6df2a322234f722ec676c9
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -sha1 -hmac "123456789" file.tx
t
HMAC-SHA1(file.txt)= a99de09fa26fb48d5a7b7ac529082041e9556461
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -sha256 -hmac "abcdef" file.txt
HMAC-SHA256(file.txt)= 721d04f2dc02816a5338a2b78aaec2e393060e24bce2ae84be6098fef
4e95d04
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -sha256 -hmac "123456789" file.
txt
HMAC-SHA256(file.txt)= 301164b898547fc4ee88e2890f44252e2d73a70ec96a1ccc93de9b14a
682f2fc
```

- It can be observed that the length of hash key written by md5 is smaller in length when compared to that of sha1 and sha256. But the length of hash value is same for different keys (adcdef,1245678).
- Then when compared to sha1 and sha256, the length of hash of sha256 is larger than that of sha1 whichindicates that sha256 is more seccure than sha1. In this case also for different key there is different hash value but length of hash value is same.
- when compared to md5 and sha1/256, sha1/256 is more stronger.

- **Task 3: The Randomness of One-way Hash**

1. The text file created for this task is task3.txt and has some content in it.
2. After creating the file hash value of the file is found using both md5 and sha256.
3. Once after the hash is found, the fileis opened in bless and one bit is fliped in it and saved the file as task3_modified.txt.
4. Then again the file hash value of the file is found using both md5 and sha256 for new modified file.
5. Hash values before the flipping of bit and after the flipping of bit is changed but the length remains the same. Program is written to count the number of bits that are similar and the output shows that 3 bits are similar.

The output of hash value before and after the modification is showed below.

```
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -md5 task3.txt
MD5(task3.txt)= c6388fe685978e7d4c39f26a8e25a3b5
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -md5 task3_modified.txt
MD5(task3_modified.txt)= a134f050c828275988c87d30444535b8
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -sha256 task3.txt
SHA256(task3.txt)= 14ce918c45452a9c90edd045cef393f42079508ac1317e3dc486754bc5c72
80e
chiranthan@chiranthan-VirtualBox:~$ openssl dgst -sha256 task3_modified.txt
SHA256(task3_modified.txt)= ca3a72b97d7bd71c5c0fbc051a404652ab1d7860e75f57a0f361
d5935de1dca6
```

Program to count number ofsimilar bit is showed below.

```
def areAnagram(str1, str2):
    count=0
    n1 = len(str1)
    n2 = len(str2)
    if n1 != n2:
        return 0
    for i in range(0, n1):
        if str1[i] == str2[i]:
            count=count+1
    return count
str1 = "c6388fe685978e7d4c39f26a8e25a3b5"
str2 = "a134f050c828275988c87d30444535b8"
num=areAnagram(str1, str2)
print("number of bits same in hash1 and hash2 using md5 are : ",num)
str11="14ce918c45452a9c90edd045cef393f42079508ac1317e3dc486754bc5c7280e"
str22="ca3a72b97d7bd71c5c0fbc051a404652ab1d7860e75f57a0f361d5935de1dca6"
num1=areAnagram(str11, str22)
print("number of bits same in hash1 and hash2 using sha256 are: ",num1)
```

The output of the program to count number of similar bit is showed below.

```
chiranthan@chiranthan-VirtualBox:~$ python strcmp1.py

('number of bits same in hash1 and hash2 using md5 are : ', 3)
('number of bits same in hash1 and hash2 using sha256 are: ', 3)
```

- **Task 4: One-Way Property versus Collision-Free Property**

The task is to write C program s to invoke the message digest functions in openssl's crypto library

Python program to check how many trials it will take you to break the one-way property and collusion free property using the brute-force method is shown below.

*collutionresistant.py - F:\CHIRU\PESIT\1SEM\CSE\LAB4\collutionresist...

File Edit Format Run Options Window Help

```
import random
import string
import hashlib
numTrials = 0

while True:
        randomString1 = ''.join([random.choice(string.ascii_lette
                string.digits + string.punctuation) for n in xran
        randomString2 = ''.join([random.choice(string.ascii_lette
                string.digits + string.punctuation) for n in xran
        if randomString1 == randomString2:
                continue;
        else:
                hash_object_1 = hashlib.md5(randomString1.encode()
                hash_string_1 = hash_object_1.hexdigest()
                hash_object_2 = hashlib.md5(randomString2.encode()
                hash_string_2 = hash_object_2.hexdigest()
                numTrials += 1
                if (hash_string_1[0:6] == hash_string_2[0:6]):
                        break;
print "num trials to break collision-resistant property = "
print numTrials
```

*onewaypropety.py - F:\CHIRU\PESIT\1SEM\CSE\LAB4...

File Edit Format Run Options Window Help

```
import random
import string
import hashlib
HASH_VALUE = '286755'
numTrials = 0

while True:
        randomStr = ''.join([random.choice(string.ascii
                string.digits + string.punctuation) for
        hash_object = hashlib.md5(randomStr.encode())
        hash_string = hash_object.hexdigest()
        numTrials += 1
        if hash_string[0:6] == HASH_VALUE:
                break;
print "num trials to break one-way property = "
print numTrials
```

The output of brute force method to check number of trails to break one way property and collusion free property is shown below

```
chiranthan@chiranthan-VirtualBox:~$ python onewayproperty.py
num trials to break one-way property =
3857765
chiranthan@chiranthan-VirtualBox:~$ python onewayproperty.py
num trials to break one-way property =
4804372
chiranthan@chiranthan-VirtualBox:~$ python onewayproperty.py
^CTraceback (most recent call last):
  File "onewayproperty.py", line 10, in <module>
    randomStr = ''.join([random.choice(string.ascii_letters + string.digits + st
ring.punctuation) for n in xrange(20)])
KeyboardInterrupt
chiranthan@chiranthan-VirtualBox:~$ python onewayproperty.py
num trials to break one-way property =
3789704
```

```
chiranthan@chiranthan-VirtualBox:~$ python collutionresistant.py
num trials to break collision-resistant property =
3096865
chiranthan@chiranthan-VirtualBox:~$ python collutionresistant.py
num trials to break collision-resistant property =
4945202
chiranthan@chiranthan-VirtualBox:~$ python collutionresistant.py

^CTraceback (most recent call last):
  File "collutionresistant.py", line 9, in <module>
    randomString2 = ''.join([random.choice(string.ascii_letters + string.digits
+ string.punctuation) for n in range(20)])
KeyboardInterrupt
chiranthan@chiranthan-VirtualBox:~$ python collutionresistant.py
num trials to break collision-resistant property =
1960752
```

- It can be observed that the average number of trail to break one-way property is 4150613 and average number of trail to break collusion free property is 3334273.
- Based on the observation the property that is easier to break using the brute-force method is one-way property.