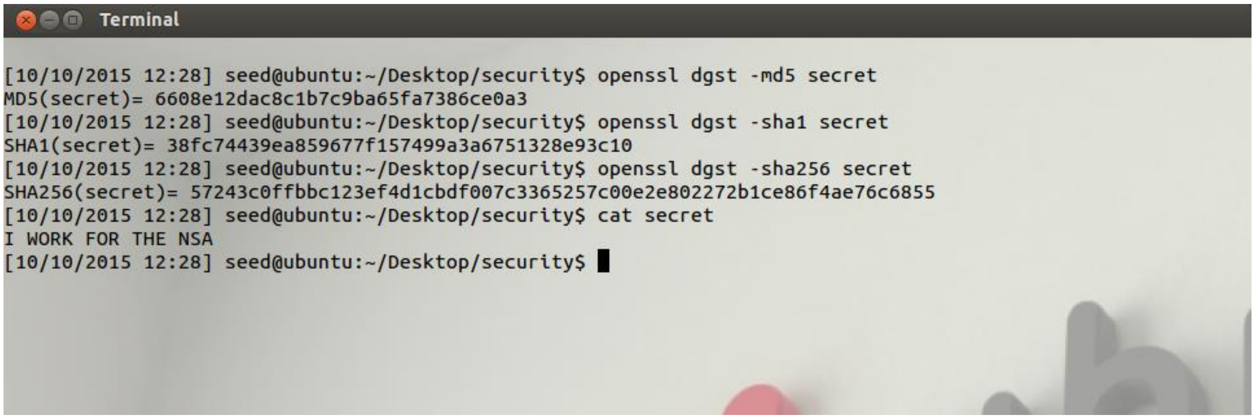
Secret.txt

I WORK FOR THE NSA

Task 1: Generating Message Digest and MAC

Creating a Digest with one-way hash algorithm such as -md5, -sha1, -sha256.



```
[10/10/2015 12:28] seed@ubuntu:~/Desktop/security$ openssl dgst -md5 secret
MD5(secret)= 6608e12dac8c1b7c9ba65fa7386ce0a3
[10/10/2015 12:28] seed@ubuntu:~/Desktop/security$ openssl dgst -sha1 secret
SHA1(secret)= 38fc74439ea859677f157499a3a6751328e93c10
[10/10/2015 12:28] seed@ubuntu:~/Desktop/security$ openssl dgst -sha256 secret
SHA256(secret)= 57243c0ffbbc123ef4d1cbdf007c3365257c00e2e802272b1ce86f4ae76c6855
[10/10/2015 12:28] seed@ubuntu:~/Desktop/security$ cat secret
I WORK FOR THE NSA
[10/10/2015 12:28] seed@ubuntu:~/Desktop/security$ 
```

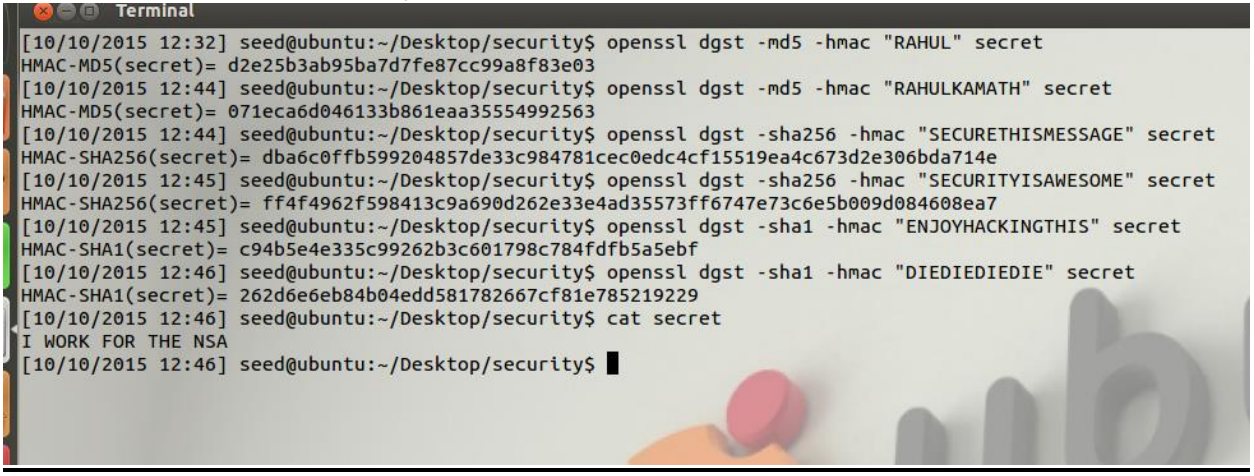
Observation

The digest created for –md5, -sha1, -sha256 are all different for the same input file (secret.txt)
The size of the msg digest increases based on the hash algorithm md5 < sha1 < sha256

Task 2: Keyed Hash and HMAC

In this task we generate a keyed hash using HMAC-MD5, HMAC-SHA256, and HMAC-SHA1

In this we do not need to use a key with a fixed size, since if we use a key with a fixed size the attacker will be able to generate all possible combinations of the key given the fixed length. The user must be able to specify any key of his/her choosing to make it difficult/impossible for an attacker to decode the string.



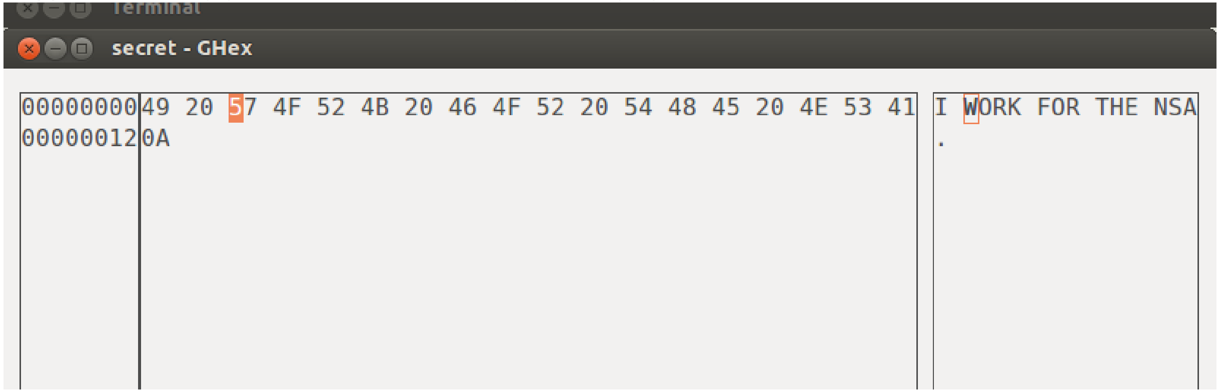
```
[10/10/2015 12:32] seed@ubuntu:~/Desktop/security$ openssl dgst -md5 -hmac "RAHUL" secret
HMAC-MD5(secret)= d2e25b3ab95ba7d7fe87cc99a8f83e03
[10/10/2015 12:44] seed@ubuntu:~/Desktop/security$ openssl dgst -md5 -hmac "RAHULKAMATH" secret
HMAC-MD5(secret)= 071eca6d046133b861eaa35554992563
[10/10/2015 12:44] seed@ubuntu:~/Desktop/security$ openssl dgst -sha256 -hmac "SECURETHISMESSAGE" secret
HMAC-SHA256(secret)= dba6c0ffb599204857de33c984781cec0edc4cf15519ea4c673d2e306bda714e
[10/10/2015 12:45] seed@ubuntu:~/Desktop/security$ openssl dgst -sha256 -hmac "SECURITYISAWESOME" secret
HMAC-SHA256(secret)= ff4f4962f598413c9a690d262e33e4ad35573ff6747e73c6e5b009d084608ea7
[10/10/2015 12:45] seed@ubuntu:~/Desktop/security$ openssl dgst -sha1 -hmac "ENJOYHACKINGTHIS" secret
HMAC-SHA1(secret)= c94b5e4e335c99262b3c601798c784fdfb5a5ebf
[10/10/2015 12:46] seed@ubuntu:~/Desktop/security$ openssl dgst -sha1 -hmac "DIEDIEDIEDIE" secret
HMAC-SHA1(secret)= 262d6e6eb84b04edd581782667cf81e785219229
[10/10/2015 12:46] seed@ubuntu:~/Desktop/security$ cat secret
I WORK FOR THE NSA
[10/10/2015 12:46] seed@ubuntu:~/Desktop/security$ █
```

Task 3: The Randomness of One-way Hash

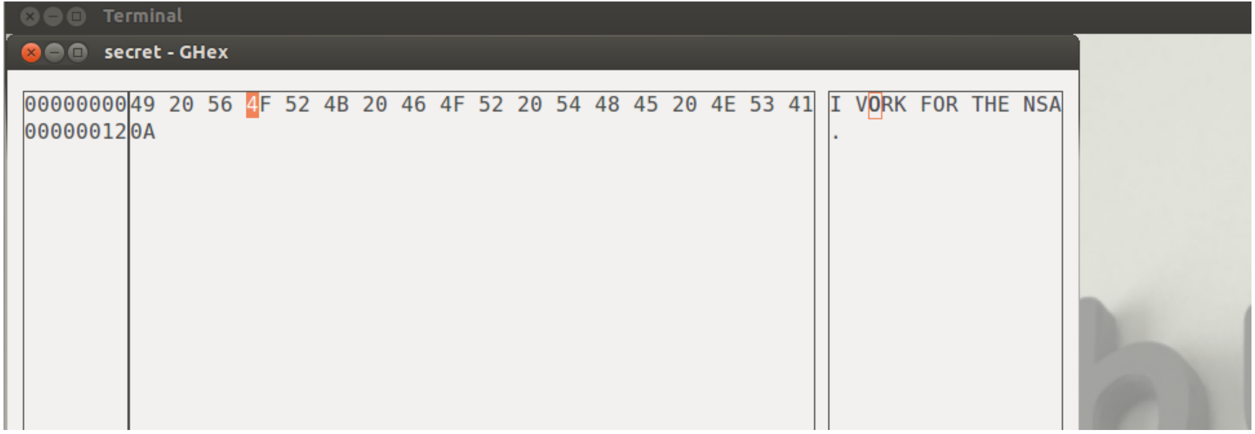
Secret.txt

I WORK FOR THE NSA

1) I used Secret.txt as shown above
2) The hash value H1 for this using **md5,** the hash created is 6608e12dac8c1b7c9ba65fa7386ce0a3
3) I flipped a bit using ghex as shown in the Image , changing 57 to 56 i.e W to V



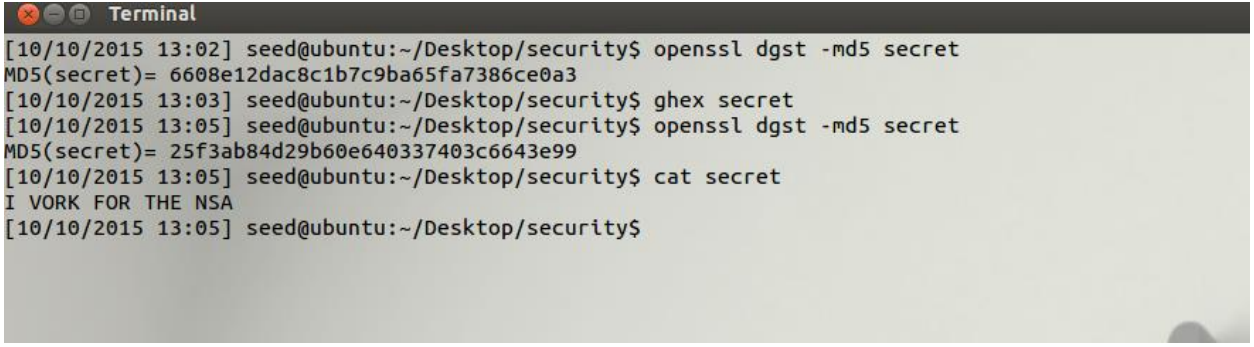
```
0000000049 20 57 4F 52 4B 20 46 4F 52 20 54 48 45 20 4E 53 41  I WORK FOR THE NSA
000000120A                                                     .
```



```
0000000049 20 56 4F 52 4B 20 46 4F 52 20 54 48 45 20 4E 53 41  I VORK FOR THE NSA
000000120A                                                     .
```

4) I generated hash **H1** for the modified file , **MD5 (MODIFIED)** = 25f3ab84d29b60e640337403c6643e99
5) The two has MD5(ORIGINAL) and MD5(MODIFIED) are different this shows that even changing 1 bit can completely change the hash generated

MD5 (ORIGINAL) = 6608e12dac8c1b7c9ba65fa7386ce0a3
MD5 (MODIFIED) = 25f3ab84d29b60e640337403c6643e99

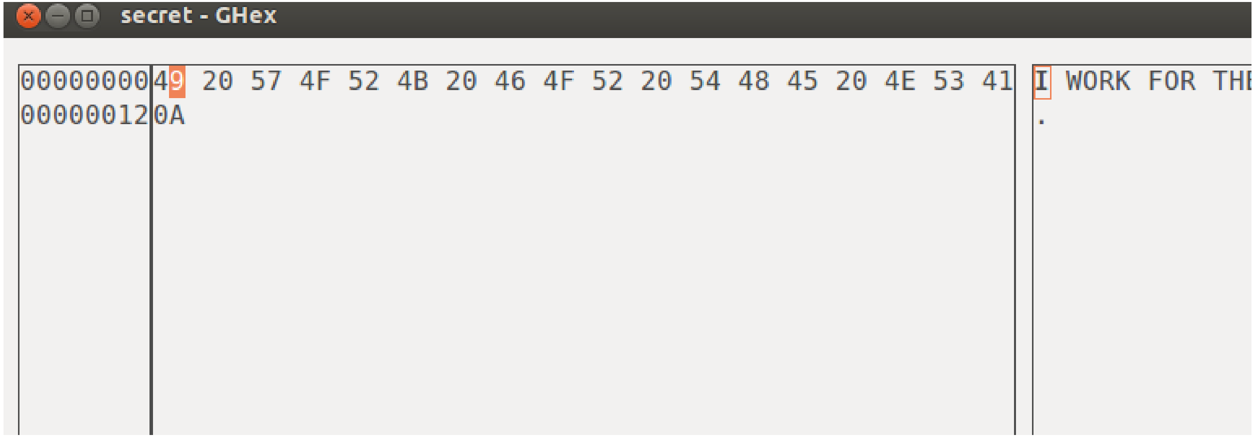


```
[10/10/2015 13:02] seed@ubuntu:~/Desktop/security$ openssl dgst -md5 secret
MD5(secret)= 6608e12dac8c1b7c9ba65fa7386ce0a3
[10/10/2015 13:03] seed@ubuntu:~/Desktop/security$ ghex secret
[10/10/2015 13:05] seed@ubuntu:~/Desktop/security$ openssl dgst -md5 secret
MD5(secret)= 25f3ab84d29b60e640337403c6643e99
[10/10/2015 13:05] seed@ubuntu:~/Desktop/security$ cat secret
I VORK FOR THE NSA
[10/10/2015 13:05] seed@ubuntu:~/Desktop/security$
```

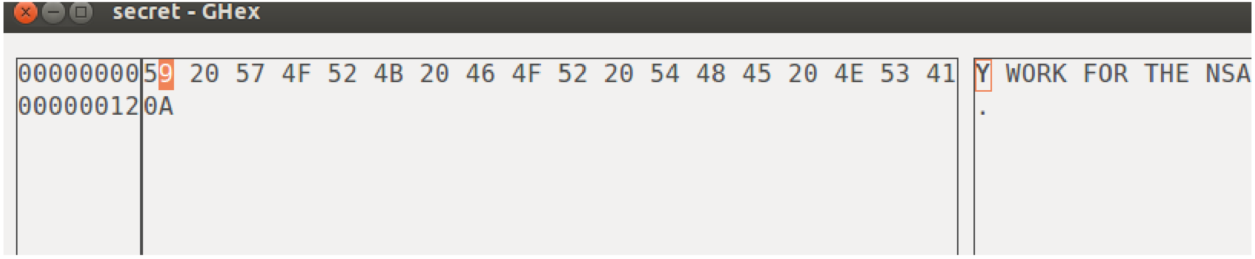
SHA 256

I WORK FOR THE NSA

1) I used Secret.txt as shown above
2) The hash value H1 for this using **SHA256,** the hash created is 57243c0ffbbc123ef4d1cbdf007c3365257c00e2e802272b1ce86f4ae76c6855
3) I flipped a bit using ghex as shown in the Image , changing 49 to 59 i.e I to Y

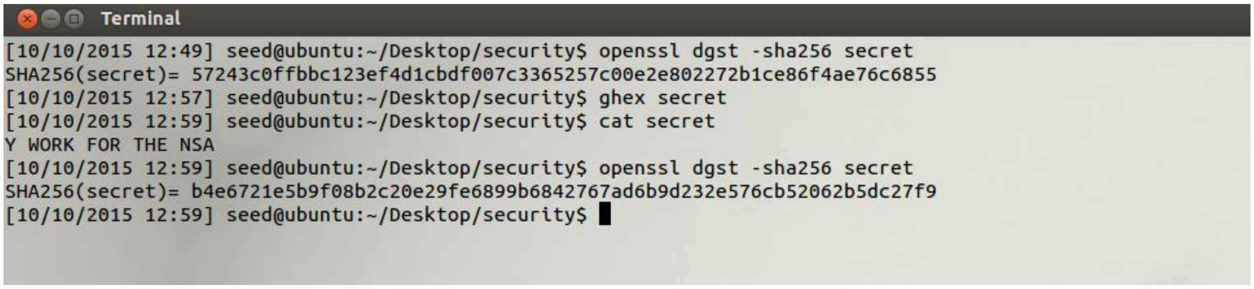


```
00000000 49 20 57 4F 52 4B 20 46 4F 52 20 54 48 45 20 4E 53 41  I WORK FOR THE
00000012 0A                                                     .
```



```
00000000 59 20 57 4F 52 4B 20 46 4F 52 20 54 48 45 20 4E 53 41  Y WORK FOR THE NSA
00000012 0A                                                     .
```

4) I generated hash **H2** for the modified file , **SHA256 (MODIFIED)** = b4e6721e5b9f08b2c20e29fe6899b6842767ad6b9d232e576cb52062b5dc27f9
5) The two hash values **SHA256** (ORIGINAL) and **SHA256** (MODIFIED) are different this shows that even changing 1 bit can completely change the hash generated

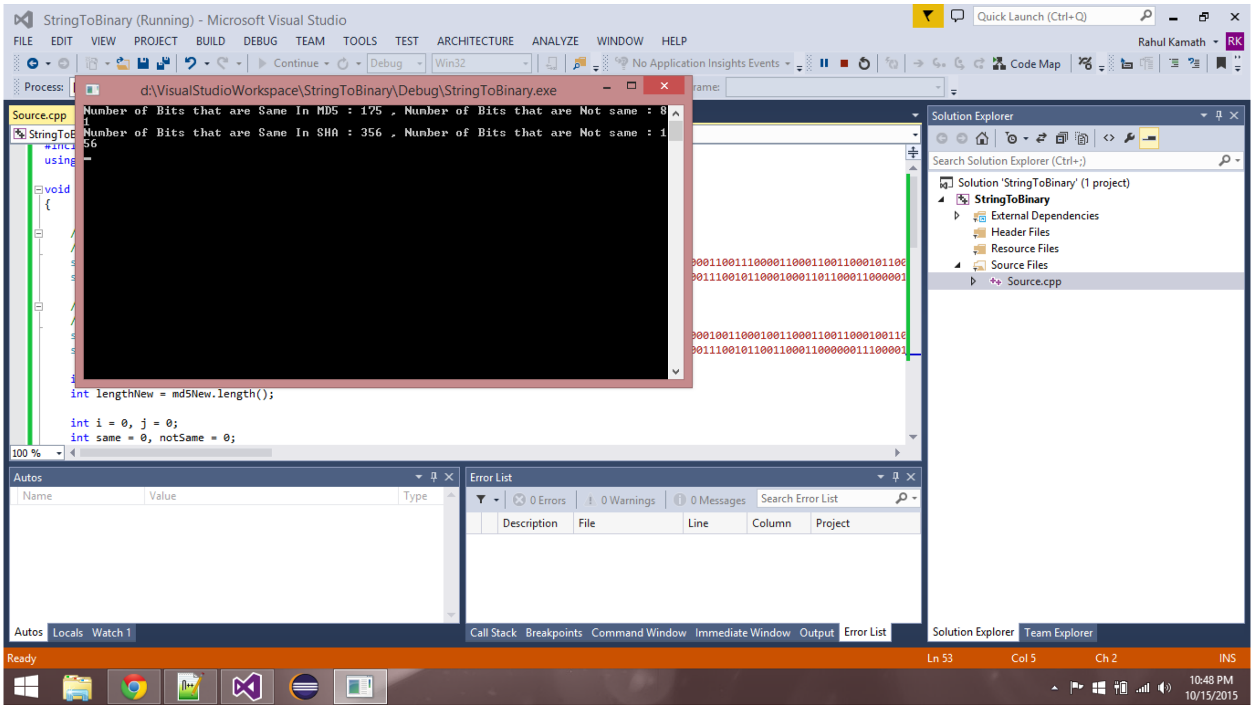


```
[10/10/2015 12:49] seed@ubuntu:~/Desktop/security$ openssl dgst -sha256 secret
SHA256(secret)= 57243c0ffbbc123ef4d1cbdf007c3365257c00e2e802272b1ce86f4ae76c6855
[10/10/2015 12:57] seed@ubuntu:~/Desktop/security$ ghex secret
[10/10/2015 12:59] seed@ubuntu:~/Desktop/security$ cat secret
Y WORK FOR THE NSA
[10/10/2015 12:59] seed@ubuntu:~/Desktop/security$ openssl dgst -sha256 secret
SHA256(secret)= b4e6721e5b9f08b2c20e29fe6899b6842767ad6b9d232e576cb52062b5dc27f9
[10/10/2015 12:59] seed@ubuntu:~/Desktop/security$ █
```

Number of Different bits are :

Number of Bits that are Same In MD5 : 175 , Number of Bits that are Not same : 81

Number of Bits that are Same In SHA : 356 , Number of Bits that are Not same : 156



StringToBinary (Running) - Microsoft Visual Studio
FILE EDIT VIEW PROJECT BUILD DEBUG TEAM TOOLS TEST ARCHITECTURE ANALYZE WINDOW HELP
Source.cpp
Solution Explorer
Autos
Error List
0 Errors 0 Warnings 0 Messages
Ready
Ln 53 Col 5 Ch 2 INS

d:\VisualStudioWorkspace\StringToBinary\Debug\StringToBinary.exe

```
Number of Bits that are Same In MD5 : 175 , Number of Bits that are Not same : 8
1
Number of Bits that are Same In SHA : 356 , Number of Bits that are Not same : 1
56
```

```
int lengthNew = md5New.length();

int i = 0, j = 0;
int same = 0, notSame = 0;
```

The program to calculate this is written in C++ and has been added as StringToBinary.cpp

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

4.1) Explain the one-way and collision-free properties of hash functions.

- One Way Hash means it is impossible to find the original contents of the file from its hash.
- Collision mean finding 2 inputs that hash to the same value, so the hash function should provide a range of random hash values such that no two words/numbers.

Hence any good Hash-Function should

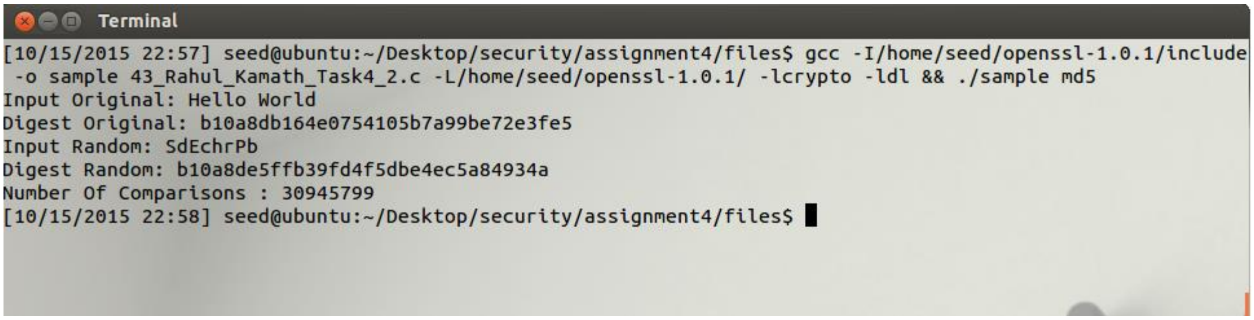
1) Provide a one way hash function making it impossible to find the message from its hash
2) Generate unique hash for every number/string/file such that no two values hash to the same hashValue.

4.2) In this task you will be breaking the one-way property using the brute-force method. Find an input which would give same first 24 bits in MD5 hash as "Hello world". You are required to write your own C program for this task.

The way to do this is

1) Generate a random string [I referred to the pseudocode for random string generation mentioned on Wikipedia and stack overflow]
2) Hash the random string to get the hash digest
3) Use this hash digest and compare the 1st three characters (24bits) with the hash digest of the original message "Hello World"
4) If they all match we have found a match and we display the random generated text to the user
5) Else Go back to 1 till a match is found

Below is the Screen Shot



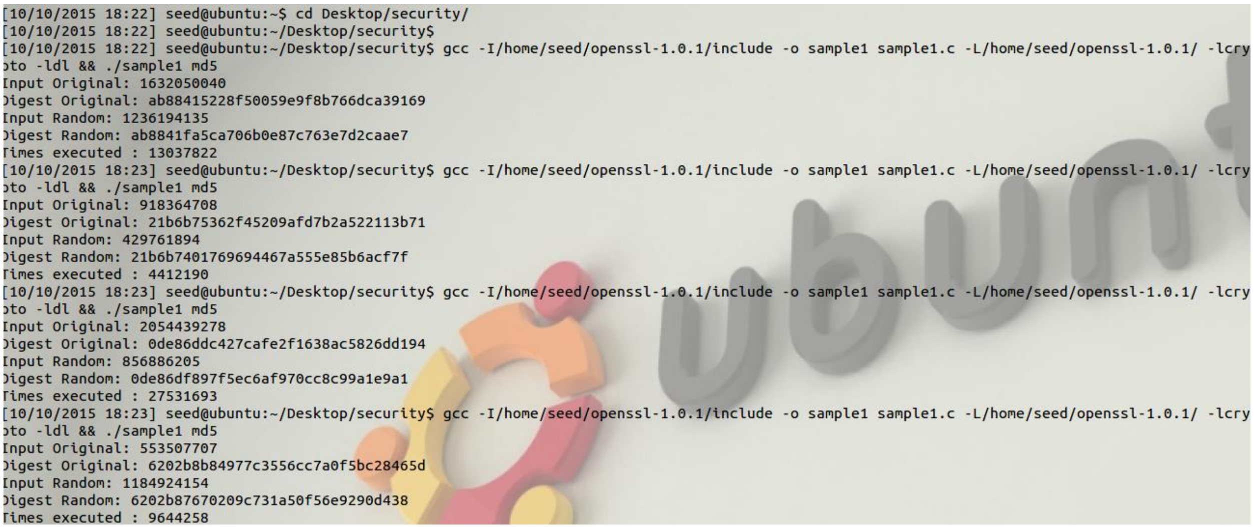
```
[10/15/2015 22:57] seed@ubuntu:~/Desktop/security/assignment4/files$ gcc -I/home/seed/openssl-1.0.1/include
 -o sample 43_Rahul_Kamath_Task4_2.c -L/home/seed/openssl-1.0.1/ -lcrypto -ldl && ./sample md5
Input Original: Hello World
Digest Original: b10a8db164e0754105b7a99be72e3fe5
Input Random: SdEchrPb
Digest Random: b10a8de5ffb39fd4f5dbe4ec5a84934a
Number Of Comparisons : 30945799
[10/15/2015 22:58] seed@ubuntu:~/Desktop/security/assignment4/files$ 
```

4.3) *In this task you will be breaking the collision-free property using the brute-force method. Generate two random numbers and compute their first 24 bits of MD5 hash, repeat until both gives same first 24 bits. You should do this 10 times and report the average number of trials it took to get same first 24 bits.*

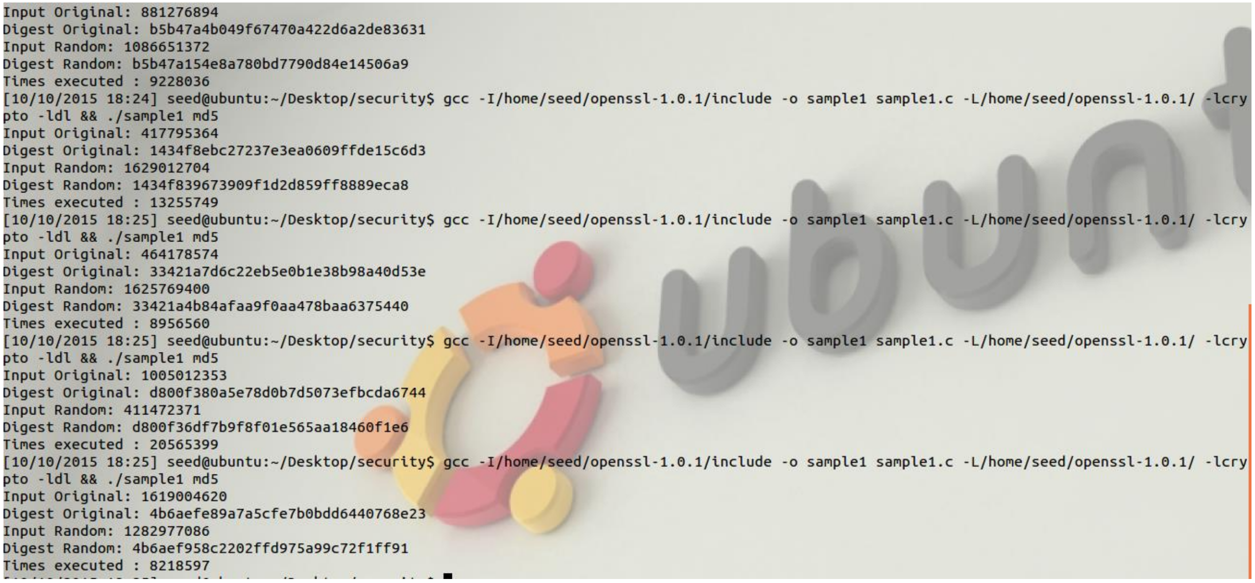
The way to do this is

1) Generate two random Numbers
2) Convert the random number to a string using sprint(), Hash the string to get the hash digest
3) Use the hash digest of the 1st number and the hash digest of the 2nd number and compare the 1st three characters (24bits)
4) If they all match we have found a match and we display the random generated numbers to the user
5) Else go to 1

The attached screen shots show the number of iterations it took to get two random numbers which is displayed as **Times Executed over 10 runs this equals 154480568/10 = 15448056.8**



```
[10/10/2015 18:22] seed@ubuntu:~$ cd Desktop/security/
[10/10/2015 18:22] seed@ubuntu:~/Desktop/security$
[10/10/2015 18:22] seed@ubuntu:~/Desktop/security$ gcc -I/home/seed/openssl-1.0.1/include -o sample1 sample1.c -L/home/seed/openssl-1.0.1/ -lcry
pto -ldl && ./sample1 md5
Input Original: 1632050040
Digest Original: ab88415228f50059e9f8b766dca39169
Input Random: 1236194135
Digest Random: ab8841fa5ca706b0e87c763e7d2caae7
Times executed : 13037822
[10/10/2015 18:23] seed@ubuntu:~/Desktop/security$ gcc -I/home/seed/openssl-1.0.1/include -o sample1 sample1.c -L/home/seed/openssl-1.0.1/ -lcry
pto -ldl && ./sample1 md5
Input Original: 918364708
Digest Original: 21b6b75362f45209afd7b2a522113b71
Input Random: 429761894
Digest Random: 21b6b7401769694467a555e85b6acf7f
Times executed : 4412190
[10/10/2015 18:23] seed@ubuntu:~/Desktop/security$ gcc -I/home/seed/openssl-1.0.1/include -o sample1 sample1.c -L/home/seed/openssl-1.0.1/ -lcry
pto -ldl && ./sample1 md5
Input Original: 2054439278
Digest Original: 0de86ddc427cafe2f1638ac5826dd194
Input Random: 856886205
Digest Random: 0de86df897f5ec6af970cc8c99a1e9a1
Times executed : 27531693
[10/10/2015 18:23] seed@ubuntu:~/Desktop/security$ gcc -I/home/seed/openssl-1.0.1/include -o sample1 sample1.c -L/home/seed/openssl-1.0.1/ -lcry
pto -ldl && ./sample1 md5
Input Original: 553507707
Digest Original: 6202b8b84977c3556cc7a0f5bc28465d
Input Random: 1184924154
Digest Random: 6202b87670209c731a50f56e9290d438
Times executed : 9644258
```



```
Input Original: 881276894
Digest Original: b5b47a4b049f67470a422d6a2de83631
Input Random: 1086651372
Digest Random: b5b47a154e8a780bd7790d84e14506a9
Times executed : 9228036
[10/10/2015 18:24] seed@ubuntu:~/Desktop/security$ gcc -I/home/seed/openssl-1.0.1/include -o sample1 sample1.c -L/home/seed/openssl-1.0.1/ -lcry
pto -ldl && ./sample1 md5
Input Original: 417795364
Digest Original: 1434f8ebc27237e3ea0609ffde15c6d3
Input Random: 1629012704
Digest Random: 1434f839673909f1d2d859ff8889eca8
Times executed : 13255749
[10/10/2015 18:25] seed@ubuntu:~/Desktop/security$ gcc -I/home/seed/openssl-1.0.1/include -o sample1 sample1.c -L/home/seed/openssl-1.0.1/ -lcry
pto -ldl && ./sample1 md5
Input Original: 464178574
Digest Original: 33421a7d6c22eb5e0b1e38b98a40d53e
Input Random: 1625769400
Digest Random: 33421a4b84afaa9f0aa478baa6375440
Times executed : 8956560
[10/10/2015 18:25] seed@ubuntu:~/Desktop/security$ gcc -I/home/seed/openssl-1.0.1/include -o sample1 sample1.c -L/home/seed/openssl-1.0.1/ -lcry
pto -ldl && ./sample1 md5
Input Original: 1005012353
Digest Original: d800f380a5e78d0b7d5073efbcda6744
Input Random: 411472371
Digest Random: d800f36df7b9f8f01e565aa18460f1e6
Times executed : 20565399
[10/10/2015 18:25] seed@ubuntu:~/Desktop/security$ gcc -I/home/seed/openssl-1.0.1/include -o sample1 sample1.c -L/home/seed/openssl-1.0.1/ -lcry
pto -ldl && ./sample1 md5
Input Original: 1619004620
Digest Original: 4b6aefe89a7a5cfe7b0bdd6440768e23
Input Random: 1282977086
Digest Random: 4b6aef958c2202ffd975a99c72f1ff91
Times executed : 8218597
[10/10/2015 18:25] seed@ubuntu:~/Desktop/security$
```

4.4) Based on your findings, which of the two properties do you think is easiest to break?

It is easier to break the one way property in this scenario, since finding two messages that have the same hash is close to impossible. In this we are comparing only the 1st three characters, and this takes close to 1282977686 iterations, if the entire hash had to match it would be literally impossible.

Also I kept a counter to count the number of comparisons to count **4.2** and **4.3**.
As we can see from the screen shot/running the program
4.2, on an average takes: **30945799**
4.3, on an average takes: >= **15448056.8**