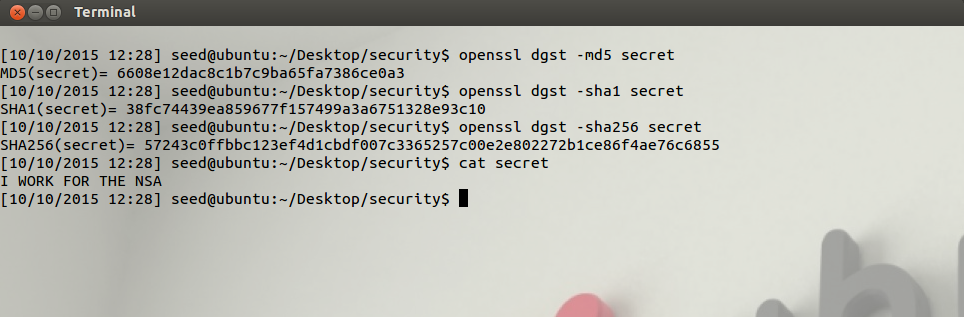
**Secret.txt**

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.



Observation

The digest created for –md5, -sha1, -sha256 are all different for the same input file (secret.txt)

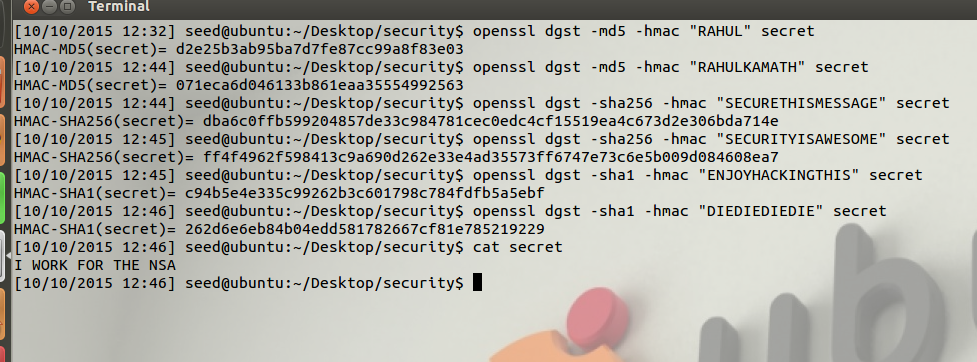
The size of the msg digest increases based on the hash algorithm md5 < sha1 < sha256

**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.

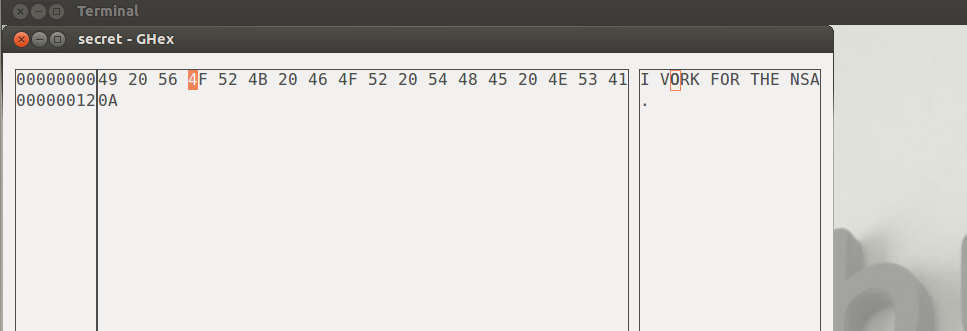
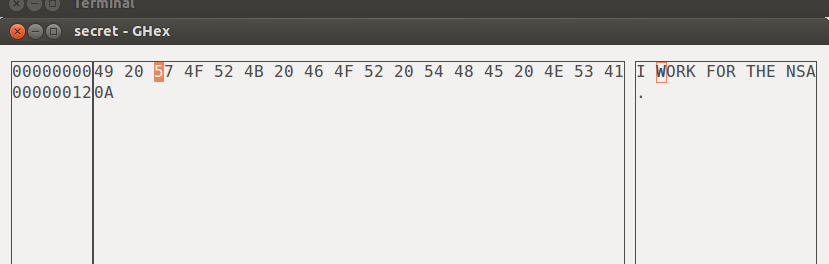


**Task 3: The Randomness of One-way Hash**

**Secret.txt**

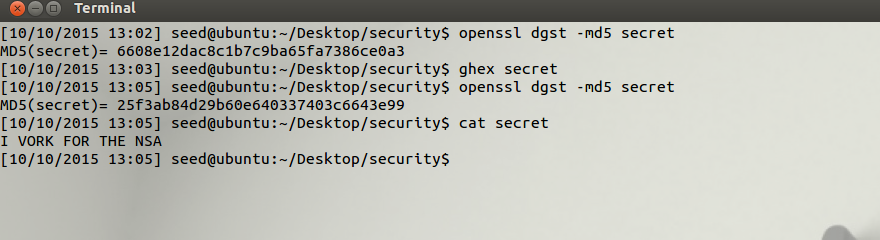
I WORK FOR THE NSA

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



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

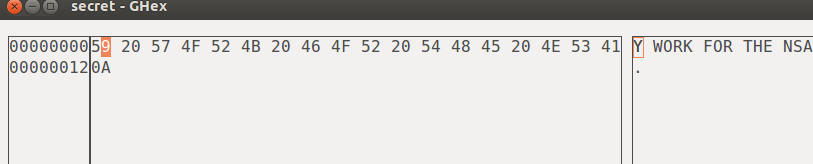
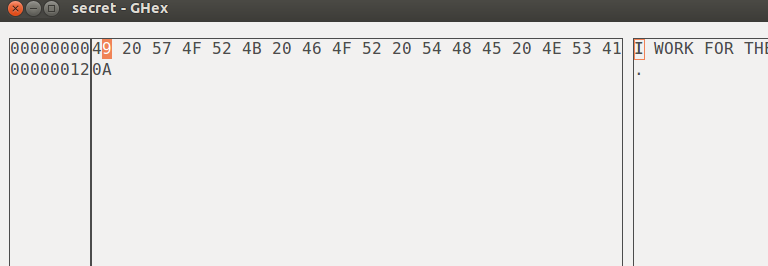
**MD5 (ORIGINAL)** = 6608e12dac8c1b7c9ba65fa7386ce0a3  
**MD5 (MODIFIED)** = 25f3ab84d29b60e640337403c6643e99



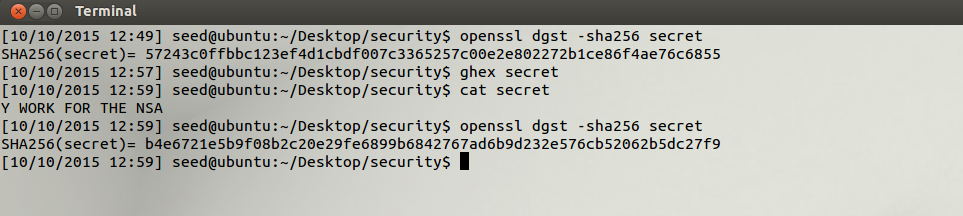
**SHA 256**

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



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



**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*

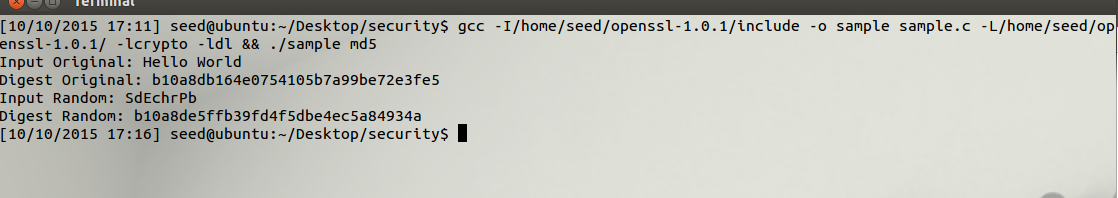
*which would give same first 24 bits in MD5 hash as “Hello world”. You are required to write your own C*

*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**



4.3*) In this task you will be breaking the collision-free property using the brute-force method. Generate*

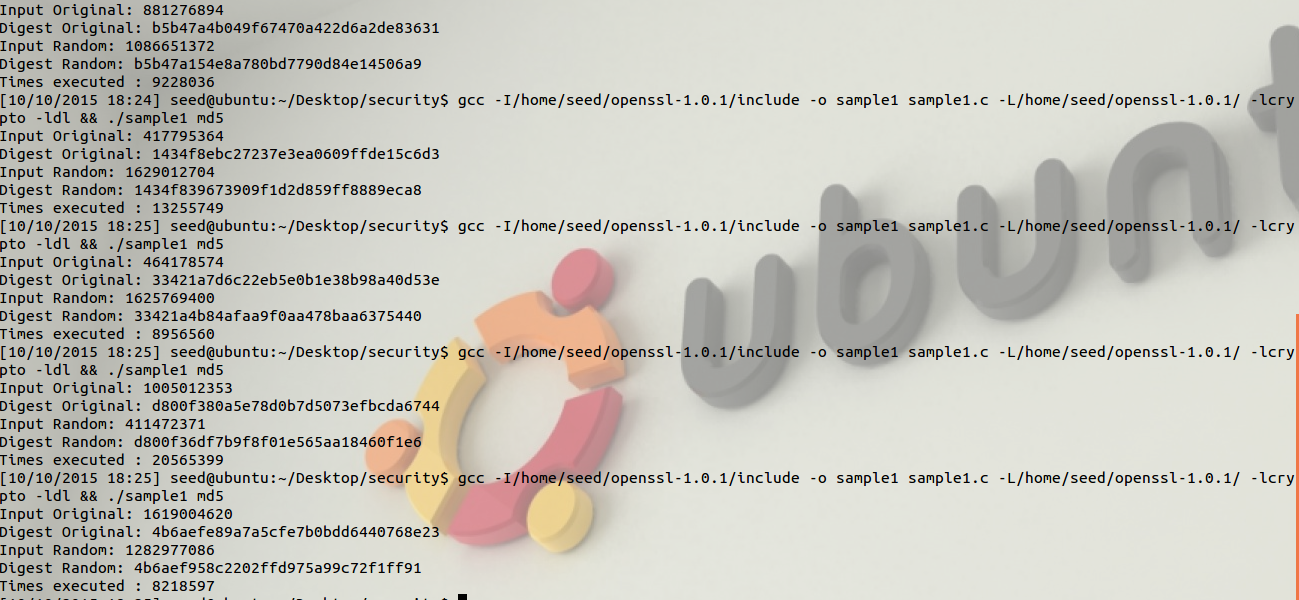
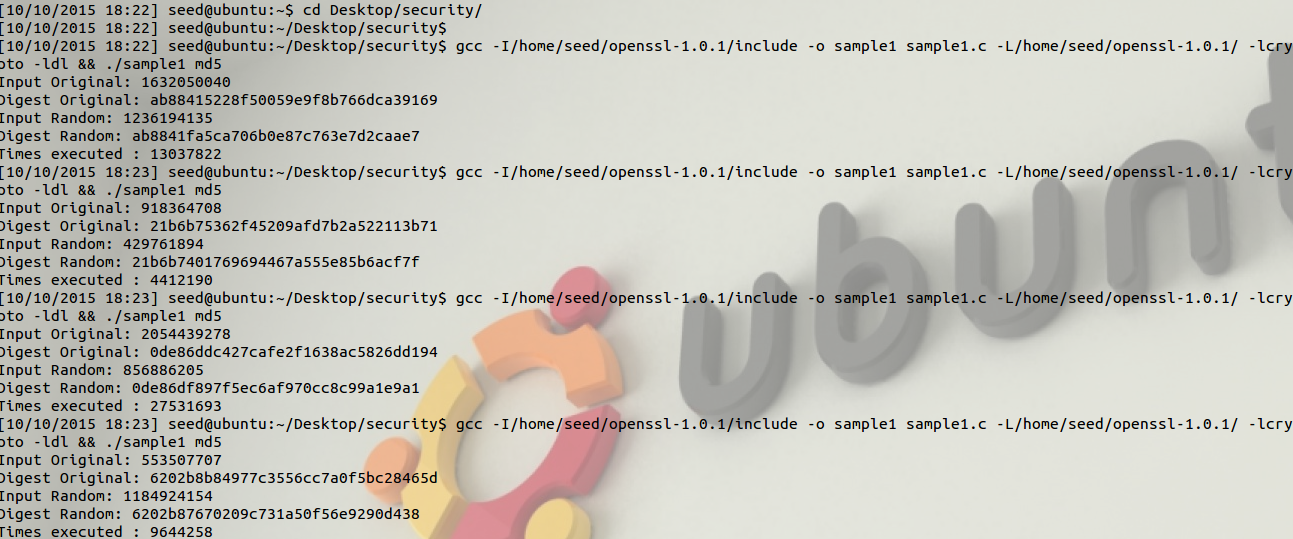
*two random numbers and compute their first 24 bits of MD5 hash, repeat until both gives same first 24*

*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**



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.