Tanmay Fadnavis

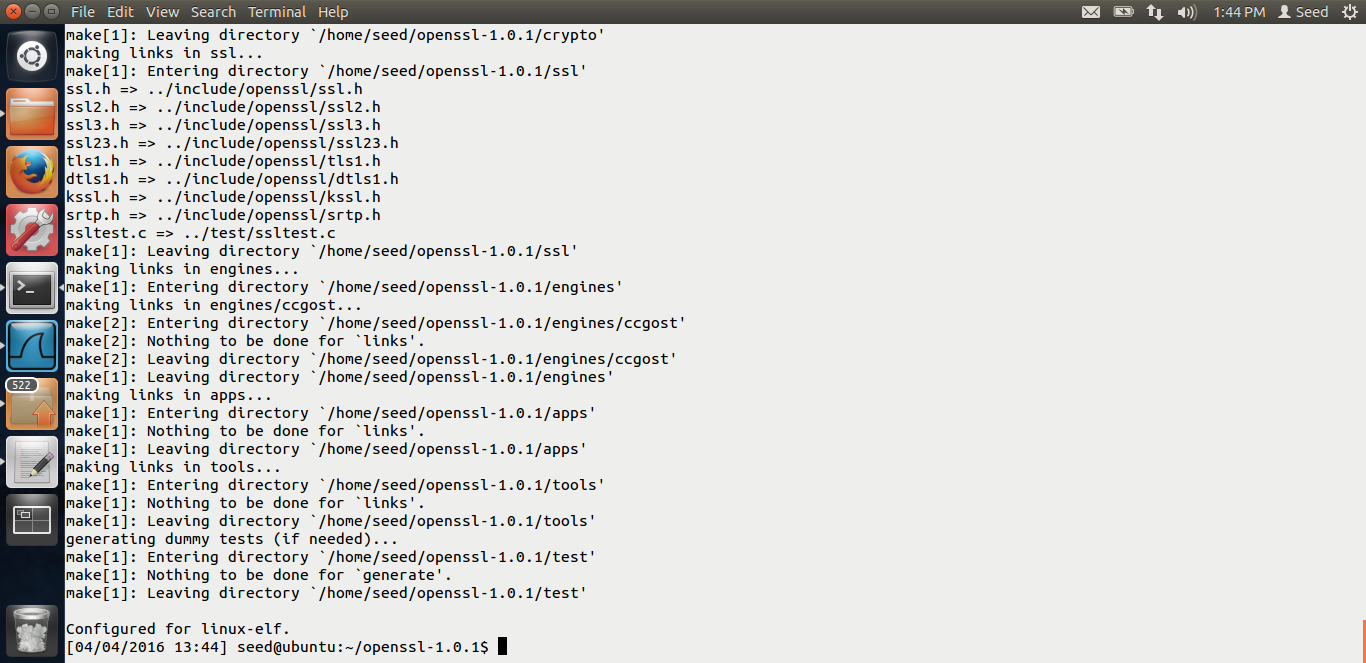
SUID: 971141760

INTERNET Security

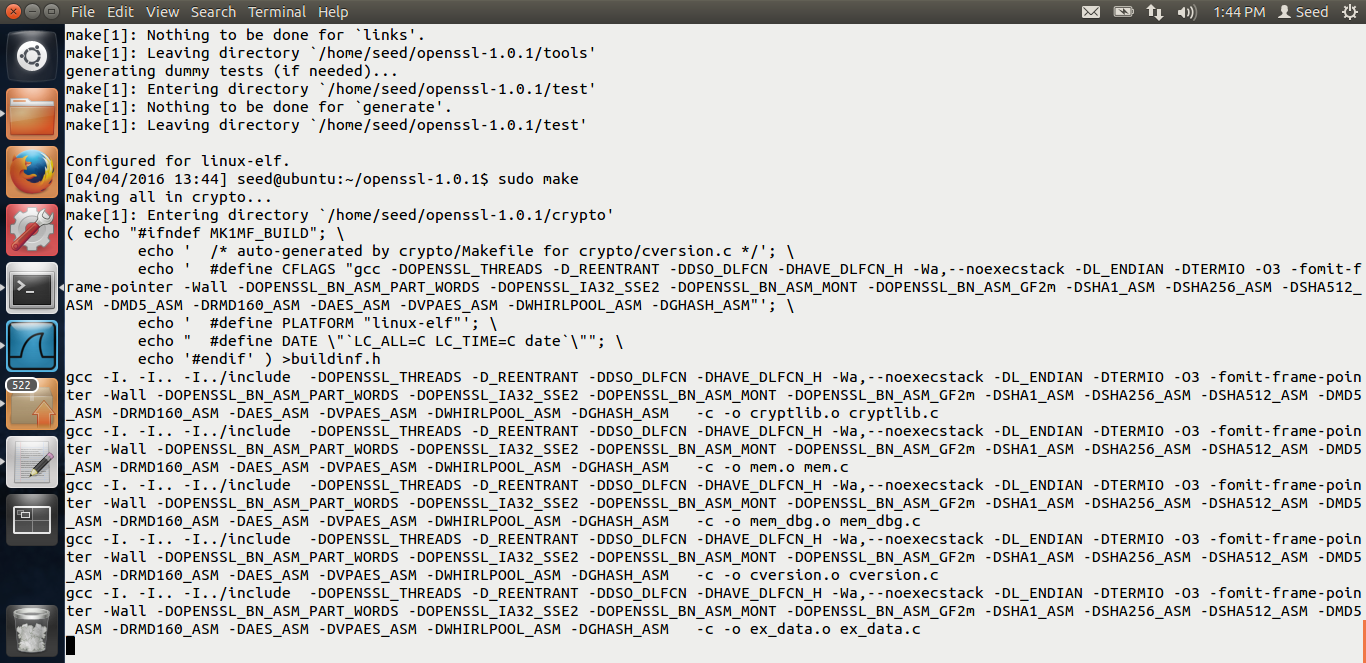
Lab - 8

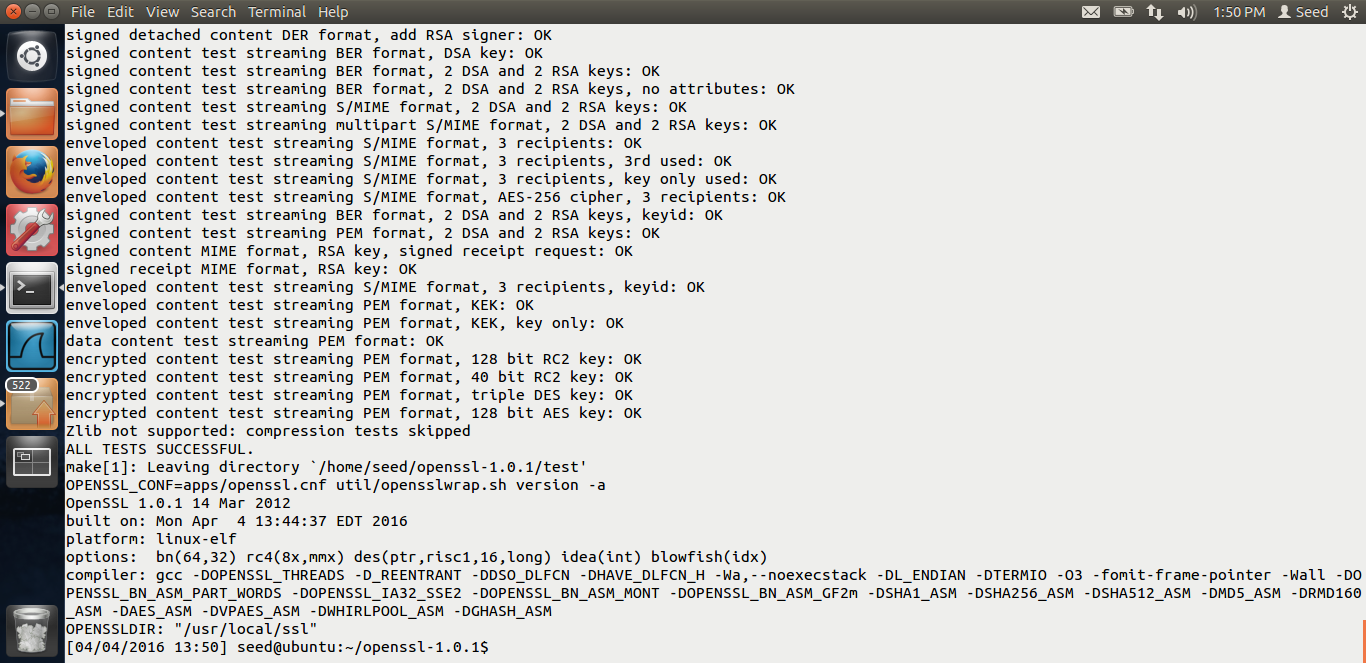
# ENVIRONMENT SETUP

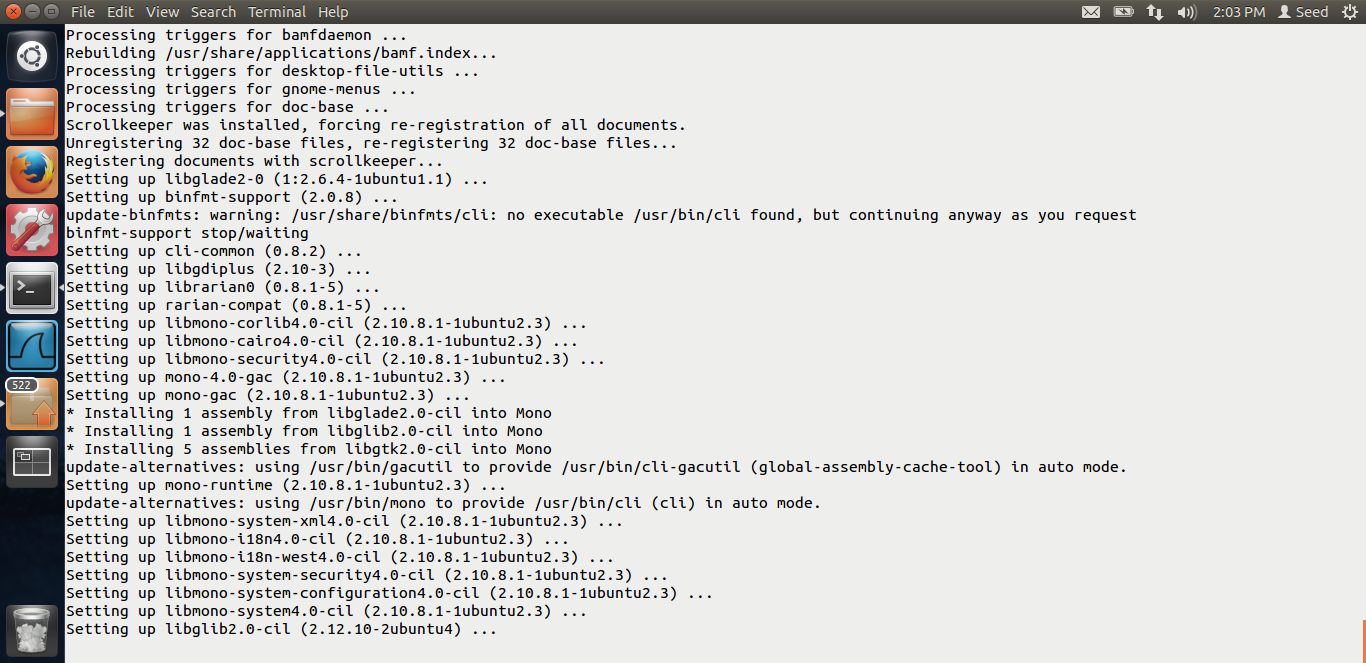
The screen shots for environment set-up are below.



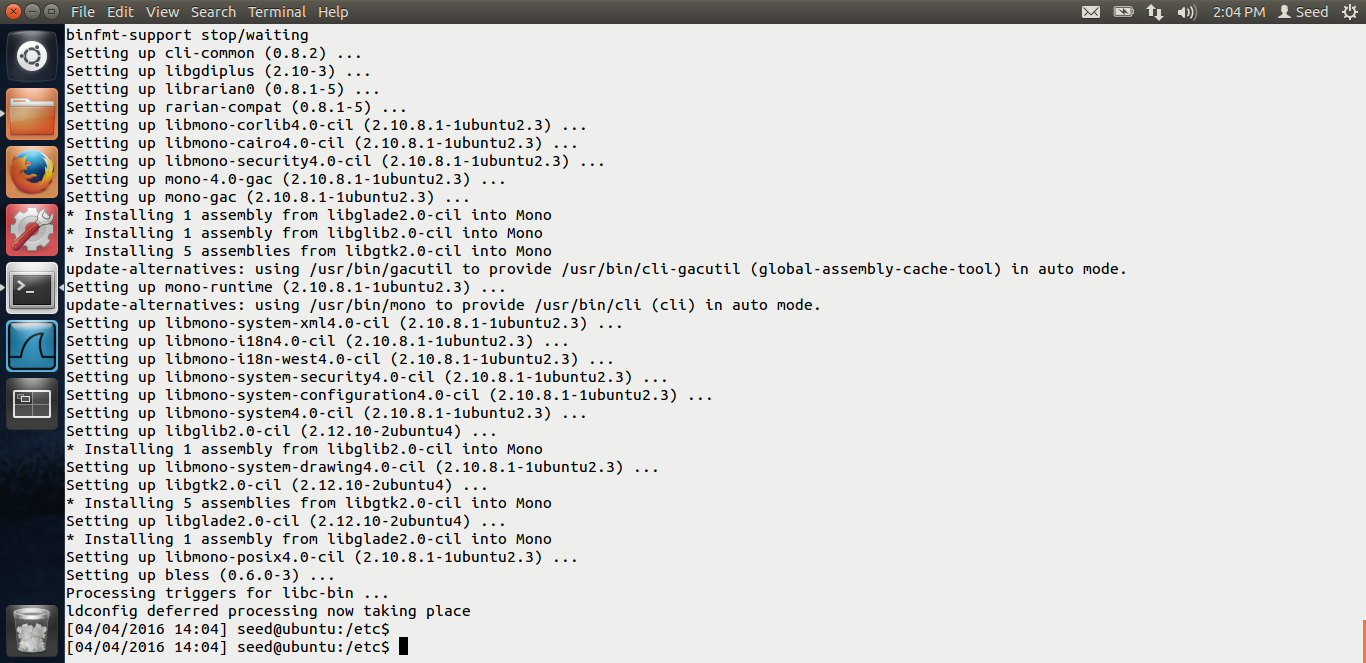
Installing the openssl libraries.







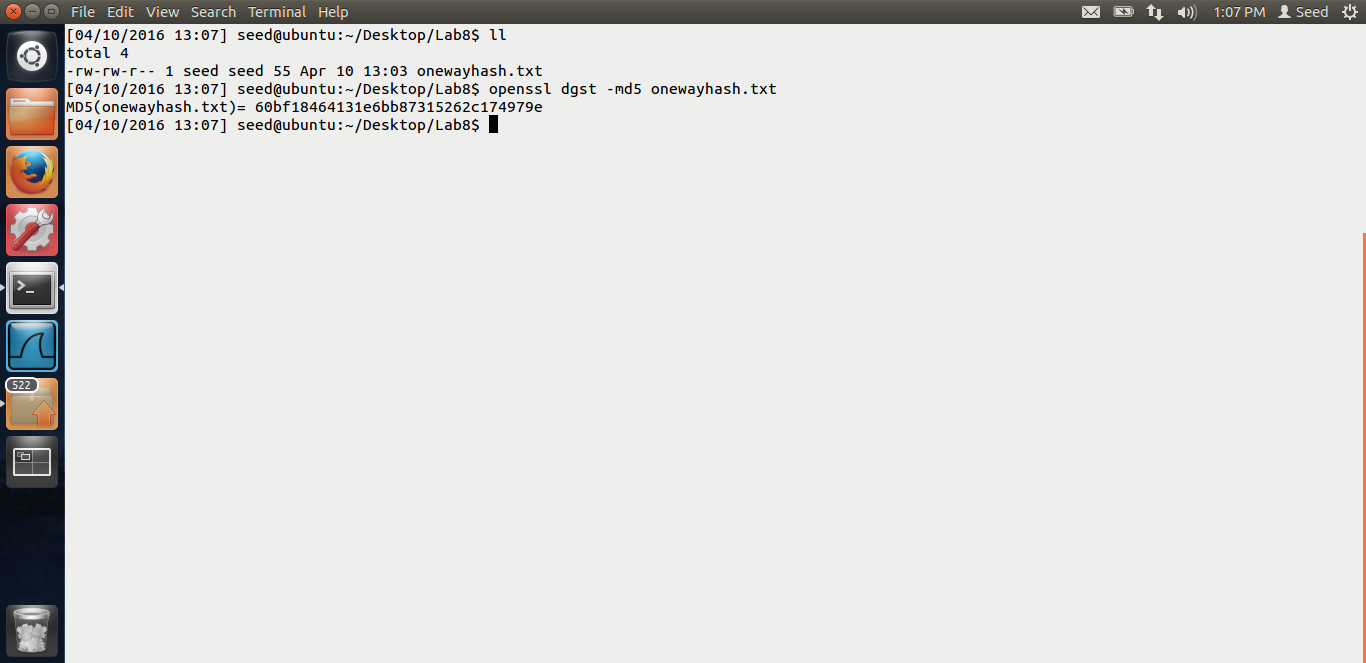
Testing the openssl libraries using the commands given in the lab description.



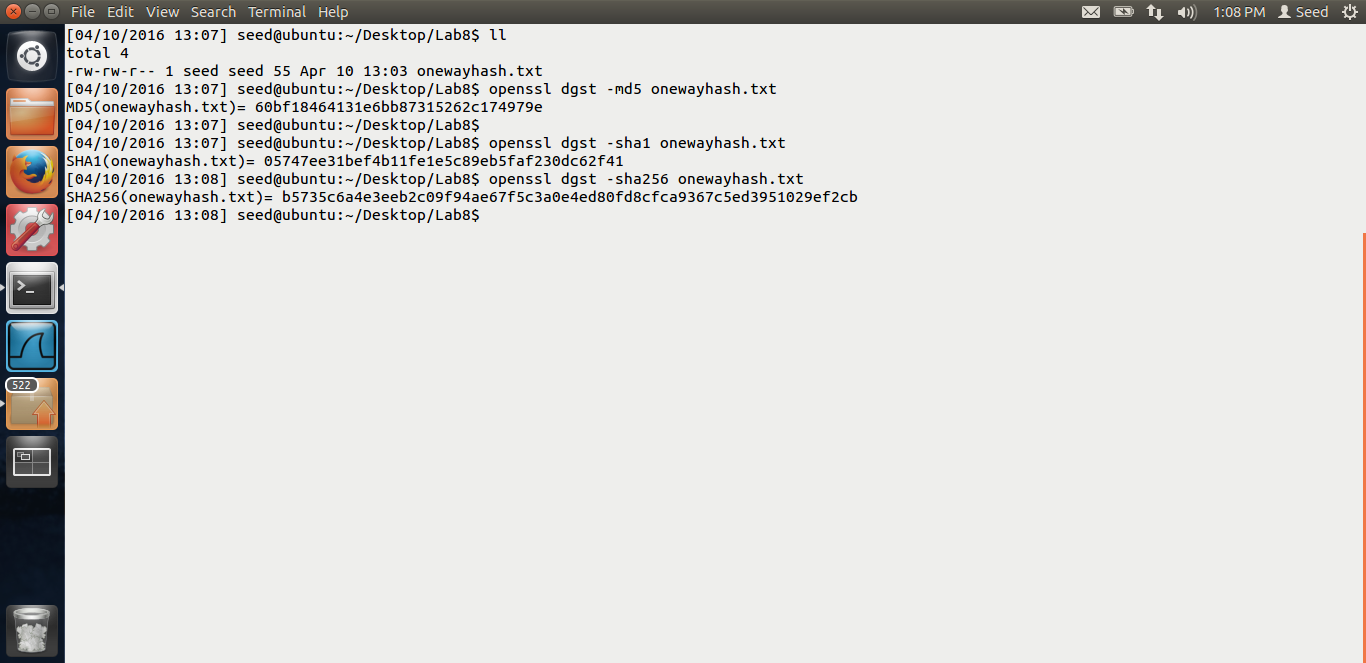
Installing the bless hex editor. The environment setup is done.

# Task 1: Generating message digest and MAC

The screen shots for task-1 are below. This task is just to generate the message digest using various algorithms and using openssl.



In the above screen shot, I have used md5. You can see the output in the screen. It creates 128 bits hash.



I have used sha1 and sha256 algorithms as seen in the above screen shot. SHA1 is 160 bits and SHA256 is 256 bits.

# Task 2: Keyed Hash and HMAC

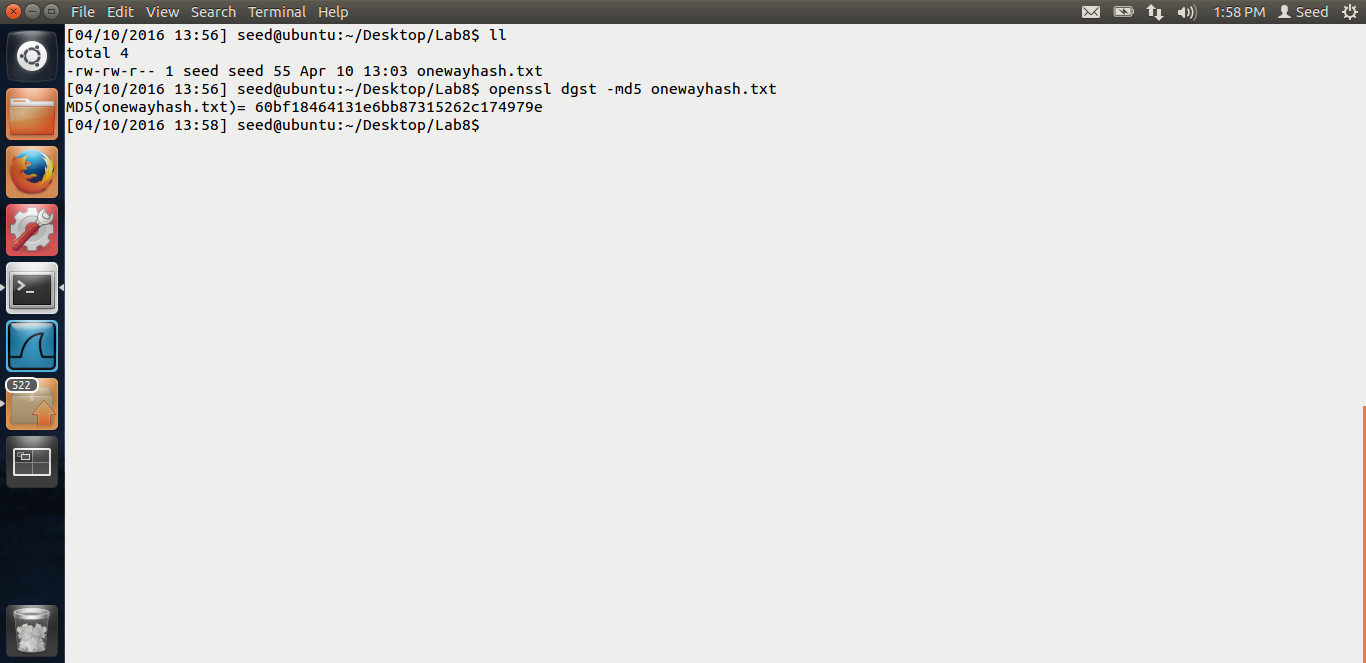
In this task, we are generating keyed hash.



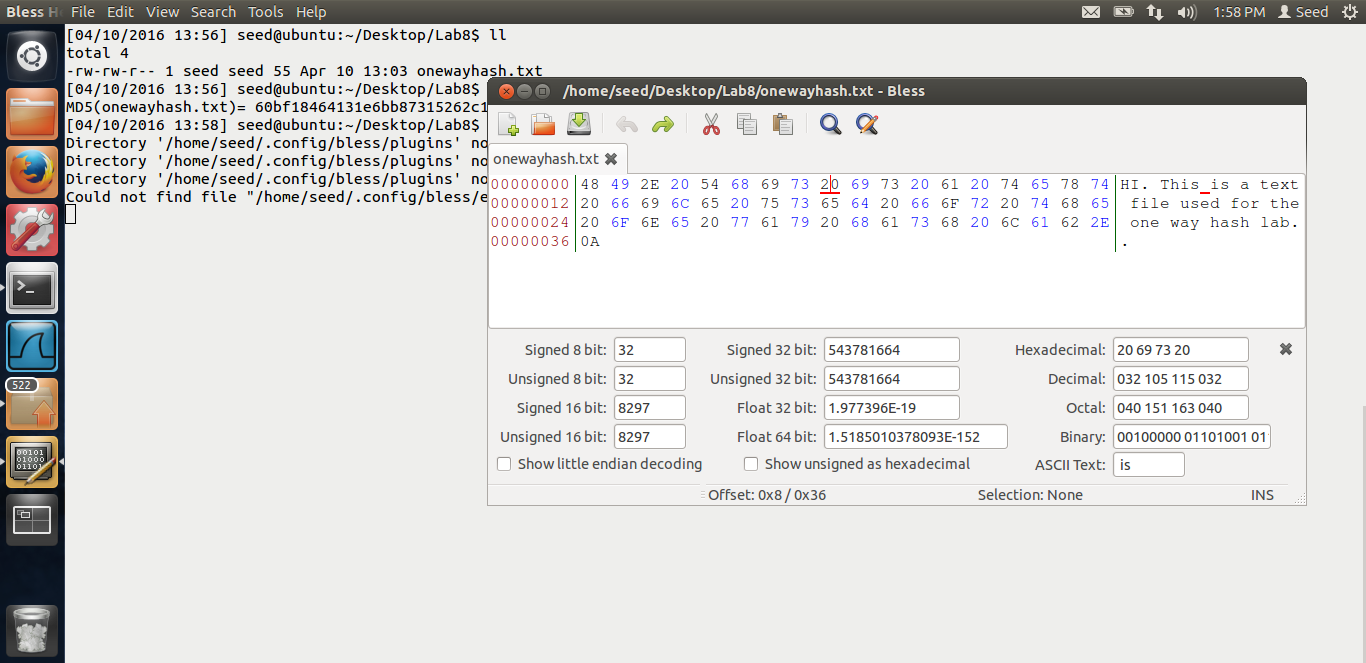
In the above screen shot, we can see that I have used different algorithms, md5,sha1,sha256 and used various keys like “abcdefgh” , and “34fgjk#2”. It generates various hashes.

The observation here is that, the key length need not be same. HMAC is a block based algorithm. Thus, if the key size is greater than block size, the hash of the key is used as key to generate keyed hash. Therefore the key length is not limited by any length.

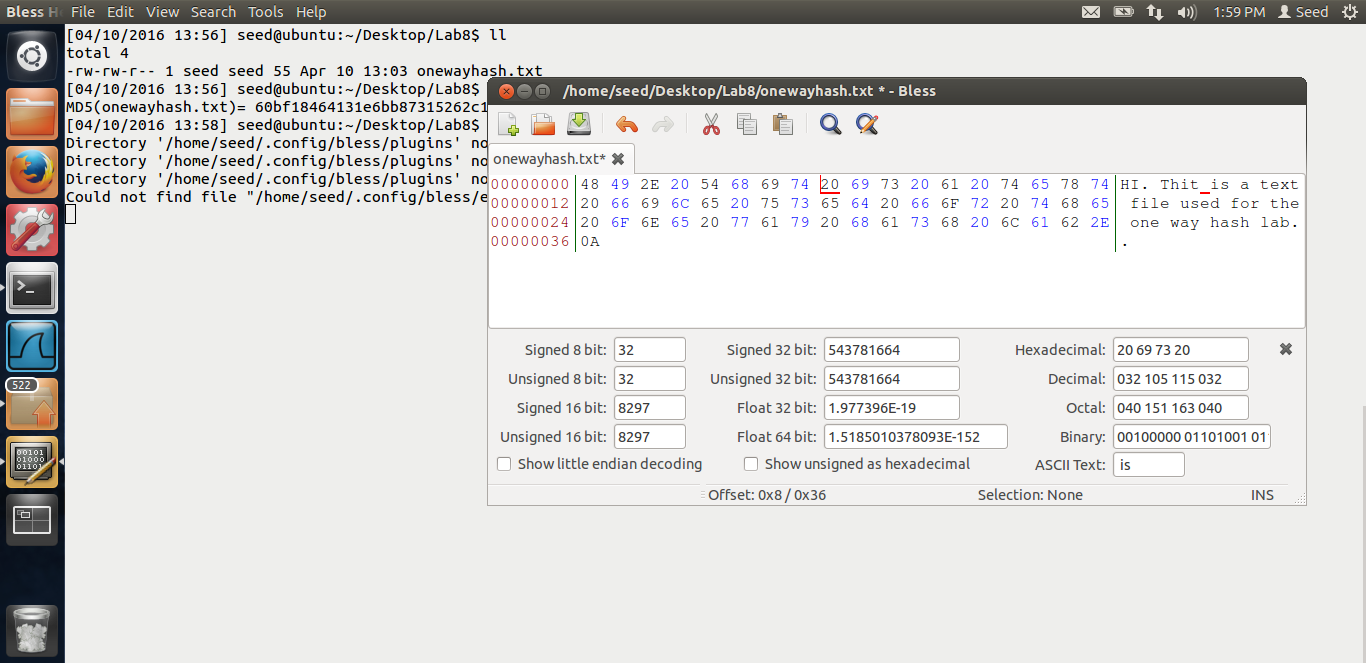
# Task 3: Randomness and one way hash



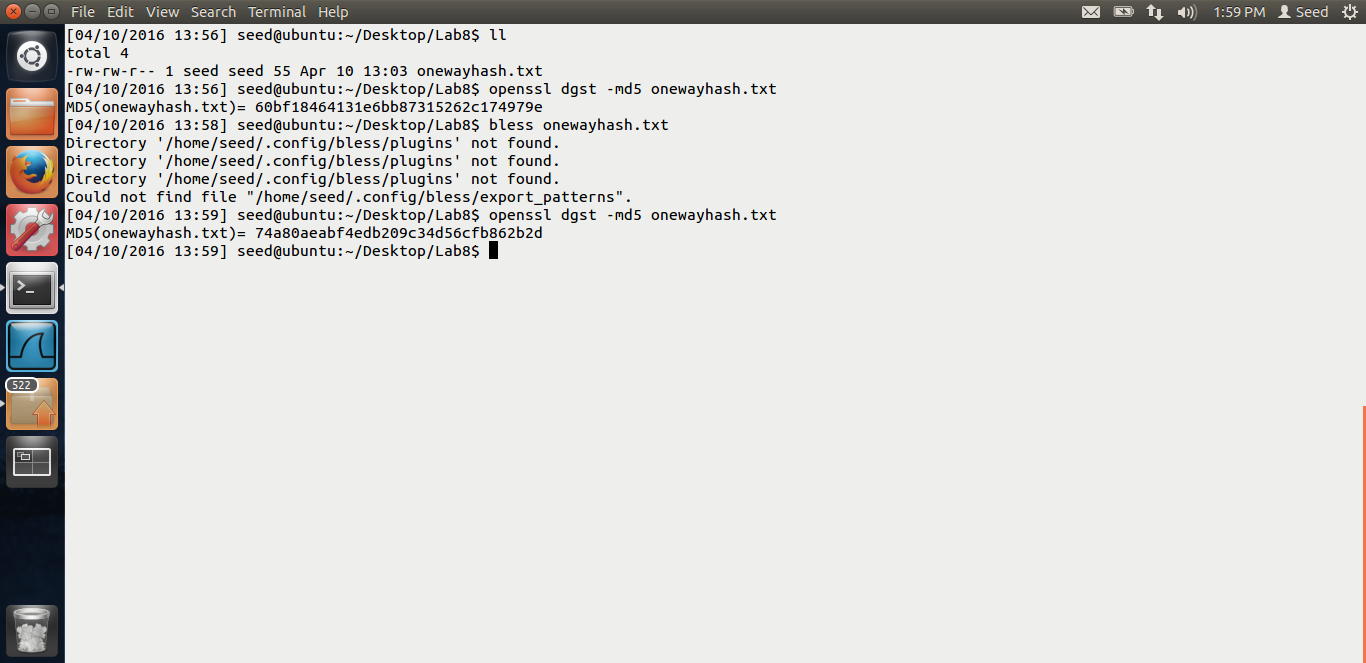
I have created a txt file, named as onewayhash. I generated its hash using the md5 as above screen shot.



This is the original hex code for the text file. Now I will change 1 bit of the hexcode and generate hash again and compare.

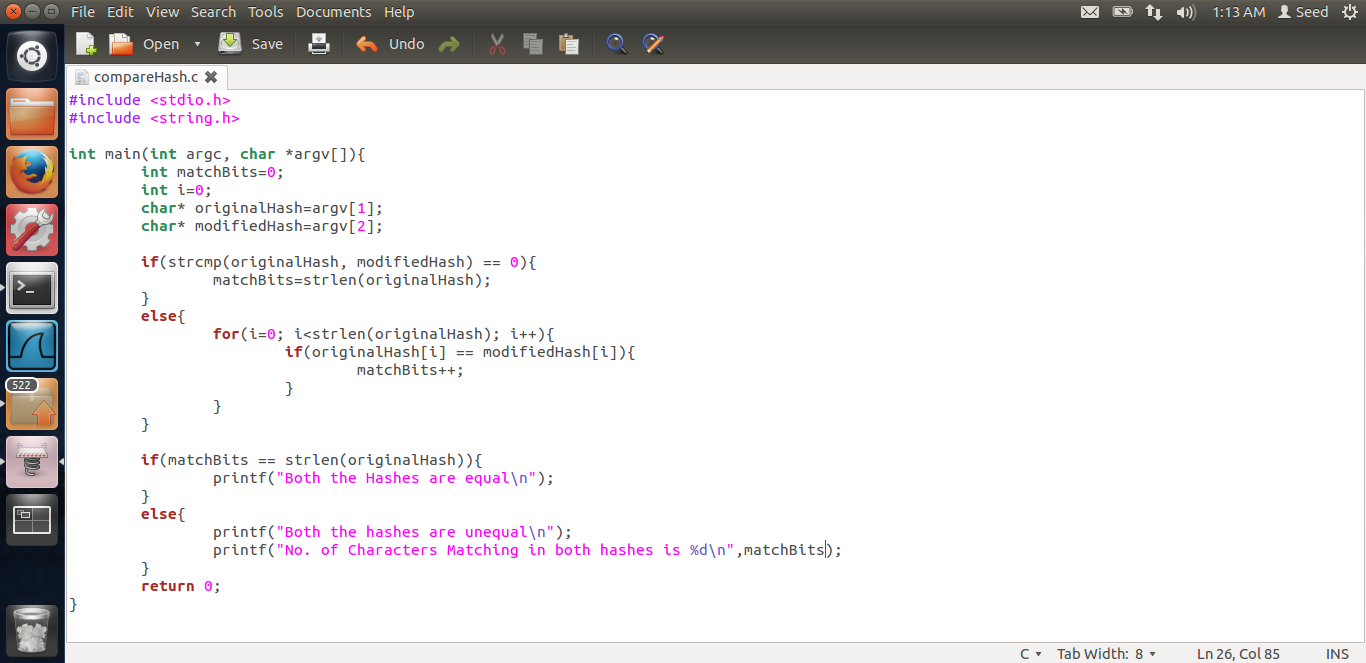


In the above screen shot, we can see that I have changed 1 bit using hex.

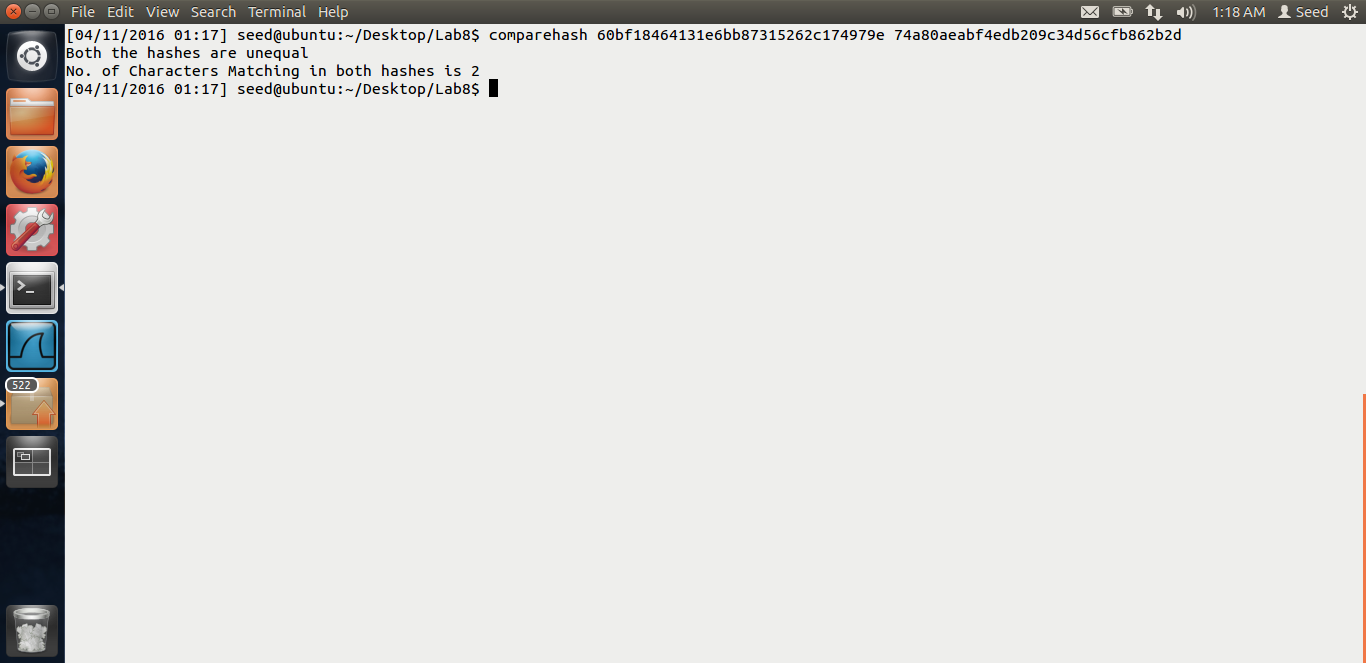


I created a new hash after changing the bit. We can see in the screen shot that this new value is different than the previous one. Thus, changing of a single bit in the hex code creates a completely random hash.

Let us write a simple program to count how many characters in both the hashes are same.



In the above code, I am just comparing the 2 hash values, taking them as argument and counting the number of common bits and printing them.



When I run the above code with the 2 hash values, I get that only 2 characters are matching, i.e. only 16 bits are matching in both the hashes. Thus, we can see that changing a single bit creates various different values of the hashes.

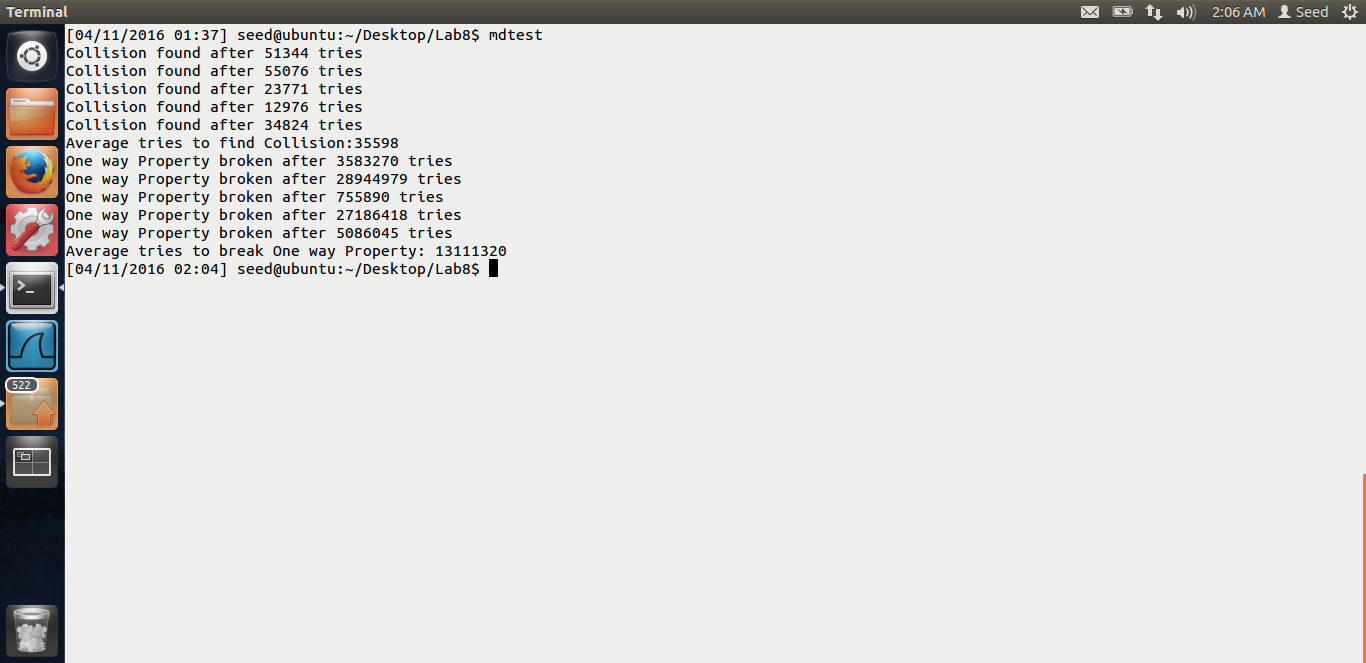
# Task 4: One way property and collision-free property

This task is to investigate the difference between 2 very important properties one way and collision free property.

One way property means when an algorithm produces a hash of a plain text, we won’t be able to produce the plain text given the hash.

Collision free means that no two plain text or messages should produce the same hash.

In this experiment, we will consider only first 24 bits of the hash value.



Firstly, this is the output when I run my code. We could see that, to break the collision, the average tries required were 35598 and to break the one way property, the average tries required were 13111320. Thus breaking one way property is way more difficult than breaking the collision free property.

It is basically difficult to find another message which generates same hash as given message because we have to try lot of combinations to get the message which has the same hash value hence it is difficult to break the one way hash. On the other hand, in collision, you just need to find 2 strings/messages which have the same hash, so the chances of collision is very high, e.g. the birthday paradox.

Mathematical proof: Let us take example num mod 3.

Num mod 3.

I can find many 2 numbers like (4,7) ( 15,12) … But if num is 34, 34 mod 3 is 1, we have very limited numbers like 1,6 .. So breaking the one way property is way difficult.

The code I have used is below.

unsigned char \*GetHash(char \*hash\_type, char \*msg){

EVP\_MD\_CTX \*mdctx;

const EVP\_MD \*md;

unsigned char \*hash\_value;

unsigned char md\_value[EVP\_MAX\_MD\_SIZE];

int md\_len, i;

OpenSSL\_add\_all\_digests();

md = EVP\_get\_digestbyname(hash\_type);

if(!md) {

printf("Unknown message digest %s\n", hash\_type);

exit(1);

}

mdctx = EVP\_MD\_CTX\_create();

EVP\_DigestInit\_ex(mdctx, md, NULL);

EVP\_DigestUpdate(mdctx, msg, strlen(msg));

EVP\_DigestFinal\_ex(mdctx, md\_value, &md\_len);

EVP\_MD\_CTX\_destroy(mdctx);

hash\_value = malloc(md\_len\*sizeof(char));

for(i=0; i<md\_len; i++){

hash\_value[i] = md\_value[i];

}

EVP\_cleanup();

return hash\_value;

}

Above is the function given on the link provided by Dr Du. It returns the hash value.

**Collision:**

int Collision(char \*hash\_type){

int size=10;

char msg1[size], msg2[size];

int count=0;

unsigned char \*hash\_value1, \*hash\_value2;

do{

genRdmStr(msg1, size);

genRdmStr(msg2, size);

hash\_value1 = GetHash(hash\_type, msg1);

hash\_value2 = GetHash(hash\_type, msg2);

count++;

}while(strncmp(hash\_value1, hash\_value2, 3) != 0);

printf("Collision found after %d tries\n", count);

return count;

}

To find collision, I just create 2 random strings and calculate their hash and check whether both hashes are the same.

**One way property:**

int OneWay(char \*hash\_type){

int size = 10;

char msg1[size], msg2[size];

int count=0;

unsigned char \*hash\_value1, \*hash\_value2;

genRdmStr(msg1,size);

hash\_value1 = GetHash(hash\_type, msg1);

do{

genRdmStr(msg2, size);

hash\_value2 = GetHash(hash\_type, msg2);

count++;

}while(strncmp(hash\_value1, hash\_value2, 3) != 0);

printf("One way Property broken after %d tries\n", count);

return count;

}

Here, I have created a hash for a string. Now I am trying various random strings and checking it its hash it same as the hash before. If yes, we have broken the one way property.

int main(int argc, char \*argv[])

{

char \*hash = "md5";

int i=0, count=0;

for(i=0; i<5; i++){

count += Collision(hash);

}

printf("Average tries to find Collision:%d\n", count/5);

count = 0;

for(i=0; i<5; i++){

count += OneWay(hash);

}

printf("Average tries to break One way Property: %d\n", count/5);

return 0;

}

In main I am just calculating the average.

End of lab.