Assignment 2 – AES and Hashing

Aksh Ravishankar 101134841

SYSC 4810

10/16/22

Task 1

Problem 1:

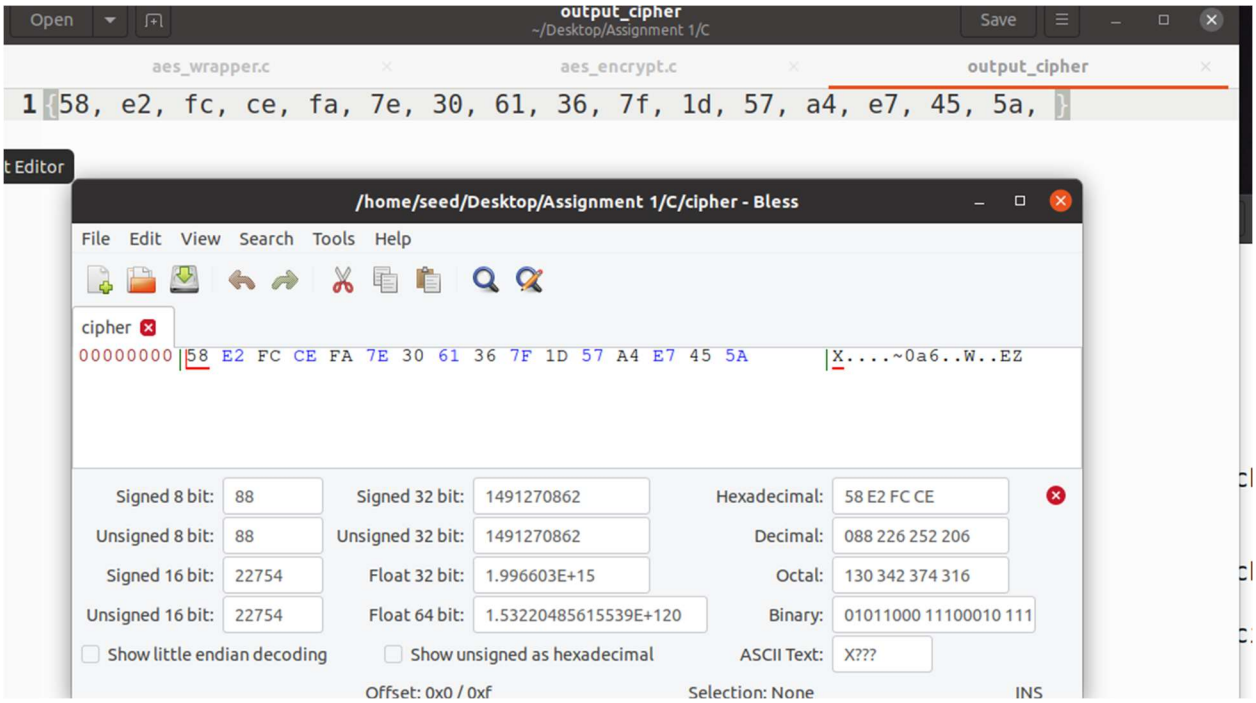

Figure 1: All 0 Output cipher

```
18 typedef unsigned char u8;
19 typedef unsigned int c8;
20
21 void main()
22 {
23         u8 key[]   = {0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,
   0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00};
24         u8 in[]    = {0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,
   0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x01};
25         u8 out[16];
```

Figure 2: All 0 input

All 0 terminal command:

```
openssl enc -aes-128-ecb -e -in plain -out cipher -K
00000000000000000000000000000000
```

Problem 2:

```
18 typedef unsigned char u8;
19 typedef unsigned int c8;
20
21 void main()
22 {
23         u8 key[]   = {0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,
   0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00};
24         u8 in[]    = {0x10  ,0x11  ,0x34  ,0x84  ,0x10  ,0x00  ,0x00  ,0x00  ,
   0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x00  ,0x01};
25         u8 out[16];
```

Figure 3: Student Number Input

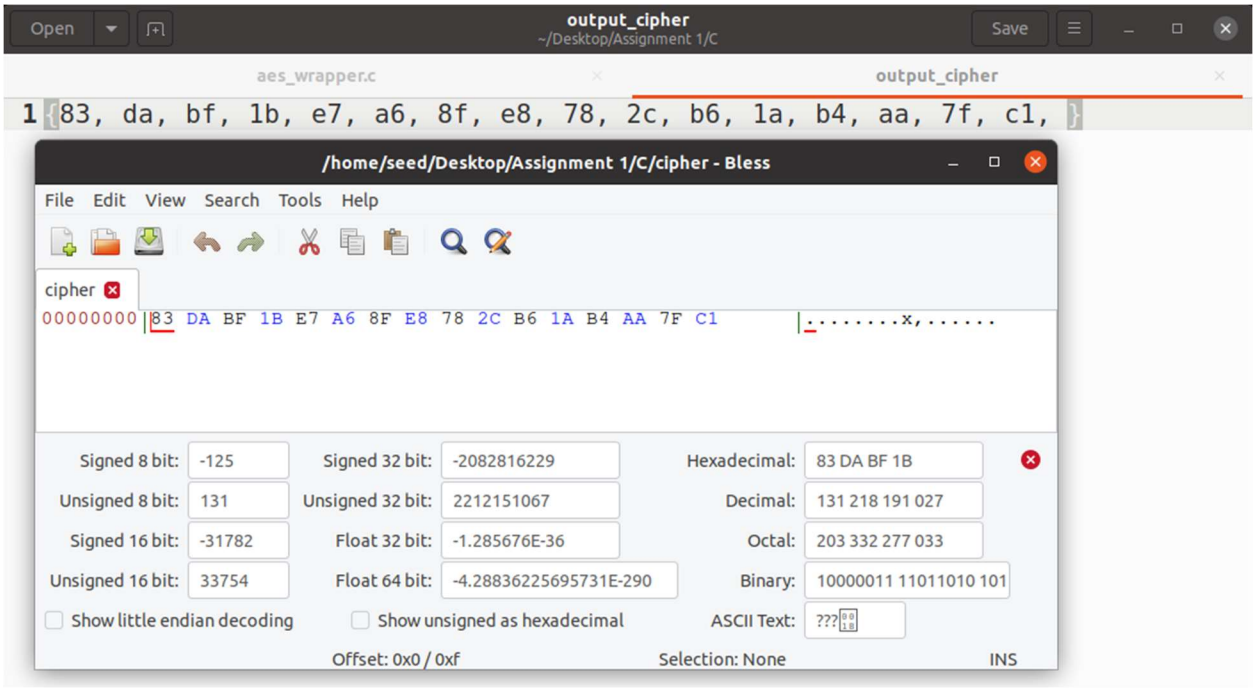

Figure 4: Student Number Output

Cipher text from student number: 83 DA BF 1B E7 A6 8F E8 78 2C B6 1A B4 AA 7F C1

Problem 3:

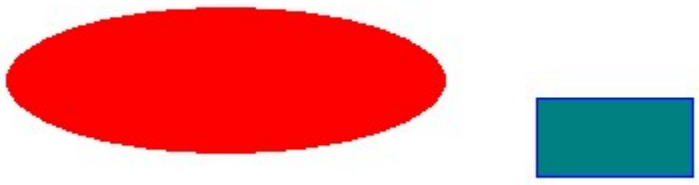

Figure 5: pic_original input picture

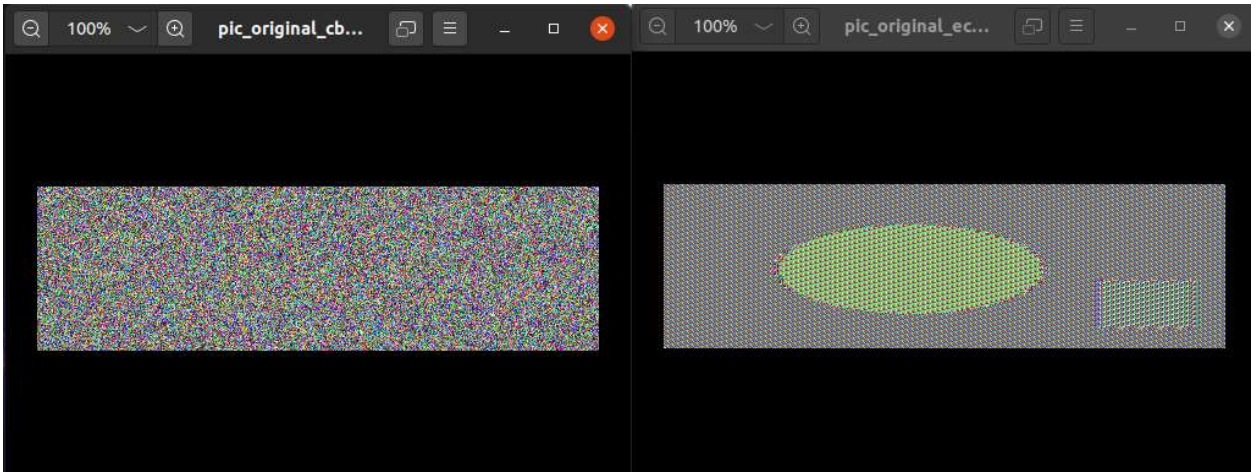

Figure 6: AES-CBC output (left) and AES-ECB output (right)

In the ECB output, the original picture is clearly visible and can be easily recreated with minimal effort. The CBC output looks completely random and doesn't look like the original picture can be recreated. This makes CBC far more secure and makes data less prone to falling prey to an attack. The other item of note is that the output is the same size as the input, which means we have no data loss and decryption with a key should return the same input.

Problem 4:

Figure 7: Personal Image Input

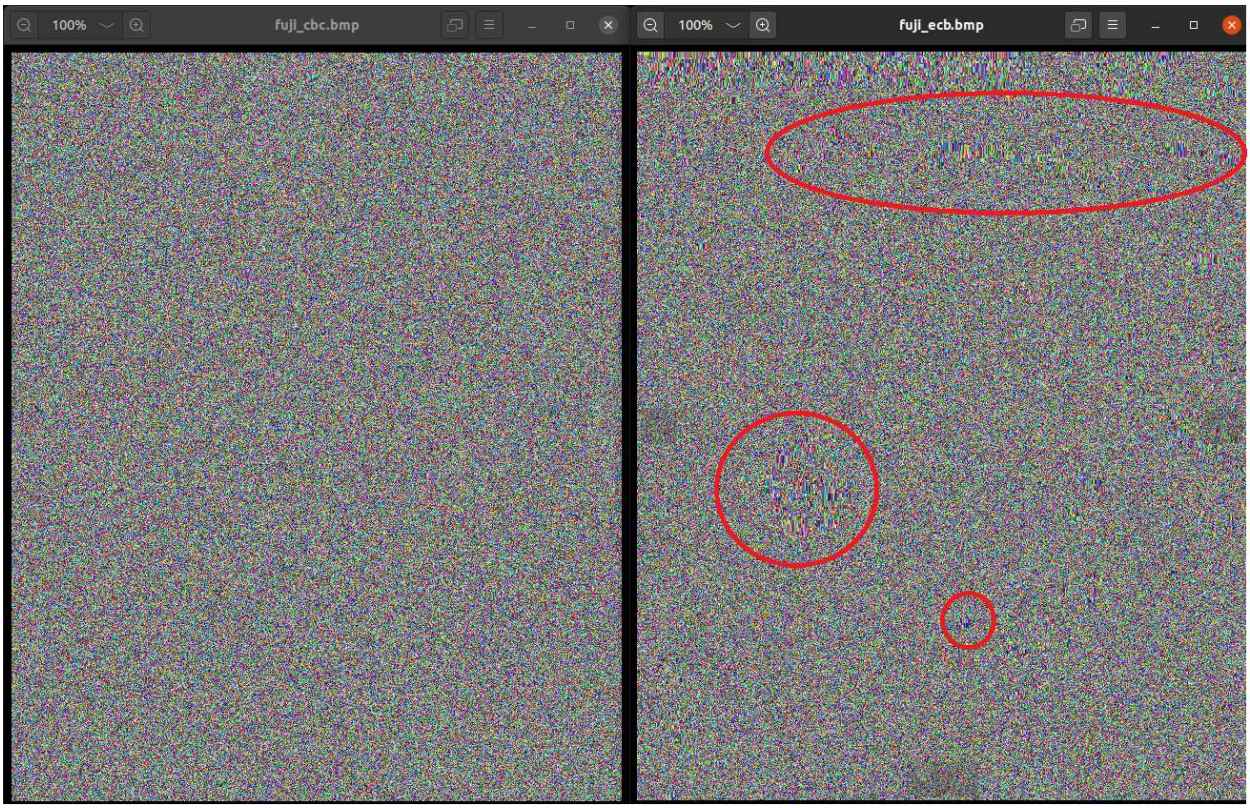

Figure 8: Personal Image AES-CBC Output (left) and Personal Image AES-ECB Output (right)

As can be seen from the above figure, even when using large amounts of data and a complicated, non-repeating input, the AES-ECB output(left) is still distinguishable from a random sequence, as there are artifacts, highlighted in red, that can be seen in the output. This makes the encryption insecure since being indistinguishable from a random output is a defining trait of a secure encrypted output. An attacker could identify this as encrypted or corrupted data and may try to recover the original information. The AEC-CBC output achieves this clearly as the output looks like static or noise, i.e. random, and not like data. Both identification and decryption without the key would be next to impossible.

Task 2

Problem 1

```
[10/10/22]seed@VM:~/.../Task 2$ openssl dgst -sha3-224 -hex Message
SHA3-224(Message)= 6b4e03423667dbb73b6e15454f0eb1abd4597f9a1b078e3f5b5a6bc7
[10/10/22]seed@VM:~/.../Task 2$ python3 hashing.py
Hex digest: 6b4e03423667dbb73b6e15454f0eb1abd4597f9a1b078e3f5b5a6bc7
[10/10/22]seed@VM:~/.../Task 2$
```

Figure 9: SHA3 Empty Output

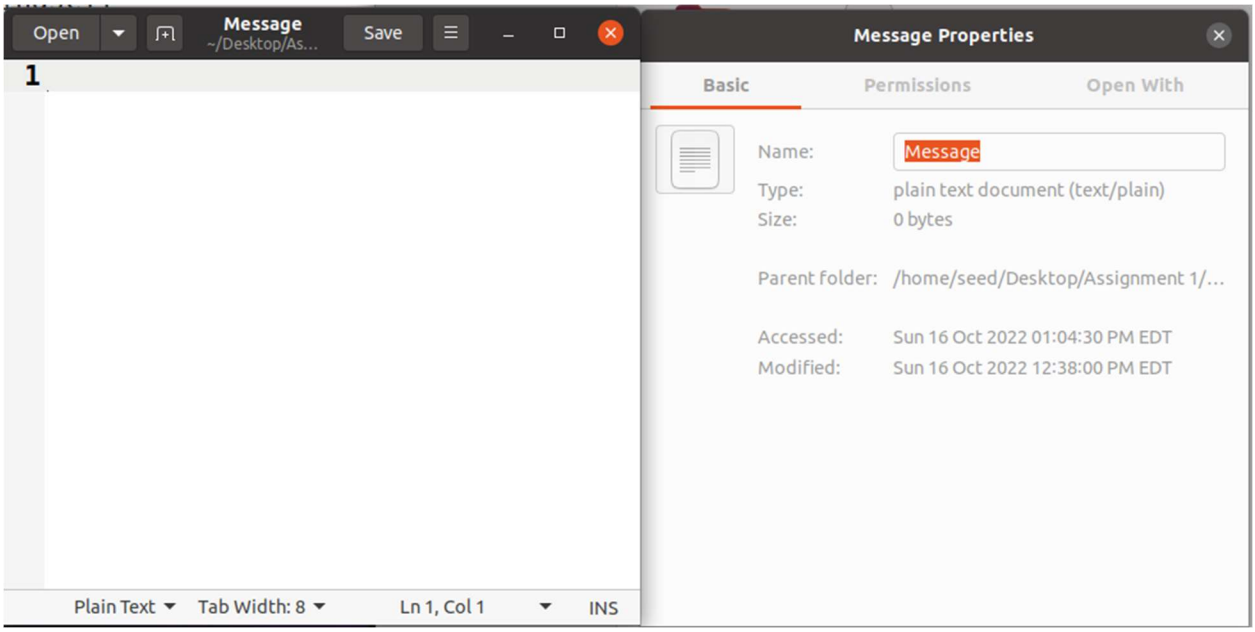

Figure 10:SHA3 Empty Input

It's clear from the above 2 sources that if the same SHA3 algorithm is used for generating an output digest, regardless of where it's called from, the same input will give the same output. The following code shows how the output was generated using python.

```
1 import os
2 import sys
3 import CompactFIPS202
4 import binascii
5
6 if __name__ == '__main__':
7         # SHA3 224 bit digest
8         cipher = CompactFIPS202.SHA3_224(b"Aksh")
9         print("Hex digest: ")
10        print("".join('{:02x}'.format(x) for x in cipher))
11
```

Figure 11: Python code for generating 224-bit digest

Problem 2

```
[10/10/22]seed@VM:~/.../Task 2$ python3 hashing.py
SHA3-224 digest: 23b89532516c314a66028371d0543cabcb8bd1c1ce32d11ba8135fb0
SHA3-256 digest: 16c56aa6c5a4367636a8e8f599952d584d5e77cb22b9168d8c4444622f9721bb
SHA3-384 digest: d9da5f1871746204b73dbe8da8391b9af9f82d88b2a0e20abe726934fc698a06ae85efce43
f917487cc84bccf0246592
SHA3-512 digest: b1ecd9a09039bcec38e8429666c35626d4239feedca01d09d067ccd3539a0a6116b58e6d2b
76eb4358d83b4aefae149405d5cfc56f4134b3e47850a8158716e6
[10/10/22]seed@VM:~/.../Task 2$ openssl dgst -sha3-224 -hex message
SHA3-224(message)= 23b89532516c314a66028371d0543cabcb8bd1c1ce32d11ba8135fb0
[10/10/22]seed@VM:~/.../Task 2$ openssl dgst -sha3-256 -hex message
SHA3-256(message)= 16c56aa6c5a4367636a8e8f599952d584d5e77cb22b9168d8c4444622f9721bb
[10/10/22]seed@VM:~/.../Task 2$ openssl dgst -sha3-384 -hex message
SHA3-384(message)= d9da5f1871746204b73dbe8da8391b9af9f82d88b2a0e20abe726934fc698a06ae85efce
43f917487cc84bccf0246592
[10/10/22]seed@VM:~/.../Task 2$ openssl dgst -sha3-512 -hex message
SHA3-512(message)= b1ecd9a09039bcec38e8429666c35626d4239feedca01d09d067ccd3539a0a6116b58e6d
2b76eb4358d83b4aefae149405d5cfc56f4134b3e47850a8158716e6
[10/10/22]seed@VM:~/.../Task 2$
```

Figure 12: SHA3 Name Output for 4 digest sizes

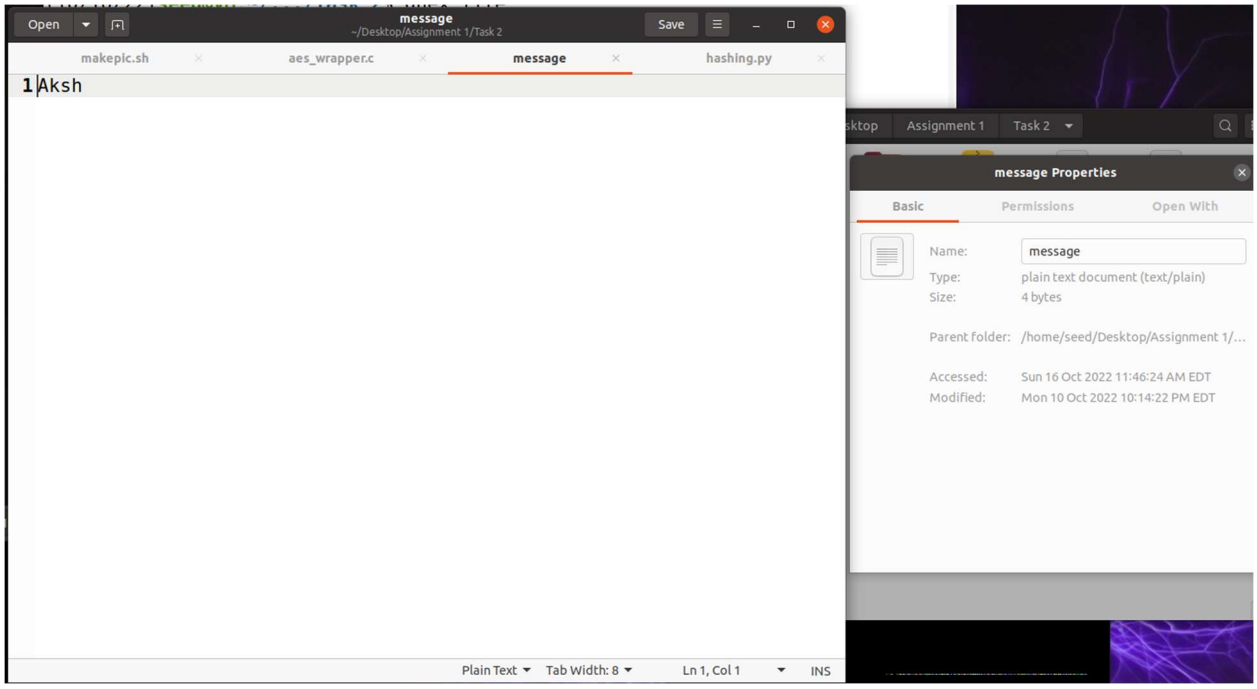

Figure 13: Hashing name input

The above output shows that changing the size of the input (from the previous example) does not change the size of the output, as the output size is defined when the hashing algorithm is called. It also once again shows that the same input will lead to the same output, regardless of where's called from.

```
# SHA3 224 bit digest
cipher = CompactFIPS202.SHA3_224(b"Aksh")
print("SHA3-224 digest: ")
print("".join('{:02x}'.format(x) for x in cipher))
```

Figure 14: Python code for generating 224-bit digest