Section 1

Rong Jun Leong 301052623
Assignment 4 CSCi157

1. (5 Points) A message of 59 bytes is encrypted using AES with the CBC mode. The padding scheme is PKCS#7. Describe what the padding data will be. What if this message has 64 Bytes?

Since the AES block size is 16 bytes, the 59 bytes message will occupy 3 blocks of 16 bytes. Therefore, 5 bytes of padding data which is equivalent to the padding data will need to fill up the last 11 byte block. If the message is 64 bytes, then the padding data will be 16 bytes because 64mod16=0 and the full 16 byte block need to be added

2. (5 Points) Alice encrypts a message using AES, and sends the ciphertext to Bob.Unfortunately, during the transmission, the 2nd bit of the third block in the ciphertext is corrupted. How much of the plaintext can Bob still recover if the mode of encryption is one of the followings: OFB, or CTR?

If the encryption mode is OFB, Bob can recover all plaintext except for the 2nd bit of the third block. Same scenario will happen to Bob if CTR encryption mode is used.

3. (5 Points) Why is the hash function f(x) = x mod 10000 not a good one-way hash function?

It has small output range, collision will happen and lack of preimage resistance which makes it less secure and effective for making a hash fuhnnction

4. (5 Points) Currently, the salt used in the shadow file is saved in the file in plaintext. Isn't it better not to save the salt in plaintext? Please explain.

The purpose of salt is to add randomness and uniqueness to password hashing and it prevents attackers from quickly guessing the password using a computed table. So it doesn't really matter where you place the salt in and it doesen't weaken the security signifinactly.

5. (5 Points) A developer writes the following in a post: "I am writing a login for a forum, and I would like to hash the password at the client side in JavaScript before sending it to the server. If the hash matches with the one stored on the server, the user will be allowed to log in." The developer believes that by sending the hash of the password, instead of sending the password directly, can improve the security. Do you agree or not, why?

I don't think it will improve security in a significant way. If the server only checks the bash or attacker gets the hashed password, the attacker still can authenticate themselves. This means that the client side never gets the original password, it will be difficult for account recovery. Other security measures should come in with client side hashing to ensure more security.

6. (5 Points) In Linux, the password hash is produced by applying a hash function for many rounds (e.g., 5000 rounds for SHA-512). This seems to waste time, why does Linux do this?

It does this because it can increase the computation time to execute the hash which makes it harder for an attacker to breach the password using brute force.

7. (5 Points) Given h = Sha256(K || M), where K is a secret and "||" means concatenation (no padding is involved in calculating h). Please describe how one can calculate Sha256(K ||X) for a different message X without knowing K.

We cannot calculate Sha256(K ||X) without knowing the value of K. The Sha256(K ||X) is a one way function which means that it is not reversible to extract K.

8. (5 Points) In the length extension attack, do we need to know the length of the key?

Yes, the attacker needs to know the length of the key to calculate the hash. The padding between the original data and additional data is crucial in order to perform the attack.

9. (10 Points) The following message K:M is fed into SHA256. (1) What will be used as the padding? (2) Given hash(K:M), we need to calculate hash(K:M:N) without knowing the value K. The string N should contain the following message "extra content". Please describe the actual content of N. K = abcd9313x
M=12345678901234567890123456789012345678901234567890
K:M = abcd9313x:12345678901234567890123456789012345678901234567890

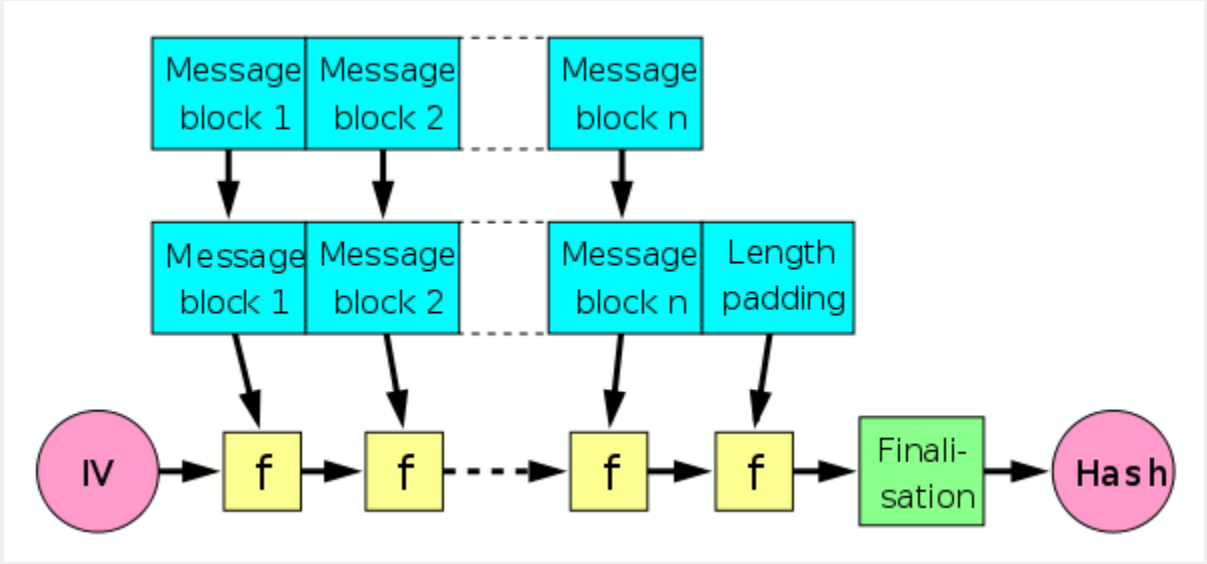
In SHA-256, data is processed in 512-bit blocks and padded if it doesn't fit. Padding involves appending a '1' bit, enough '0' bits, and the original data length. For a length extension attack, an attacker can calculate Hash(K:M:N) from Hash(K:M) and the length of K:M, without knowing K. N contains the padding for Hash(K:M), the length of K:M, and "extra content". This is a vulnerability of SHA-256 and other hash functions using the Merkle–Damgård construction. It's recommended to use a hash function not susceptible to length extension attacks, like SHA-3, or HMAC.

10. (10 Points) Charlie has arranged a blind date for Alice and Bob, who are both cryptographers, and they do not know each other before.

Charlie also gave Alice and Bob a secret number K (nobody else knows K). Bob wants to make sure that the person he is dating is Alice, not somebody else. Please describe how Bob can ask Alice to securely prove that she is Alice (Alice will not reveal the secret number K to anybody). (HINT:HMAC)

Bob can verify Alice's identity securely using the HMAC (Hash-based Message Authentication Code) protocol without Alice revealing the secret number K. Bob starts by generating a random message M and sending it to Alice. Alice then computes HMAC(K, M), where K is the secret number and M is the message received from Bob, and sends the result back to Bob. Bob also computes HMAC(K, M) on his side using the same secret number K and the message M. He then compares the HMAC value he computed with the one received from Alice. If they match, Bob can be assured that he is communicating with Alice, as only Alice and Bob know the secret number K and can generate the correct HMAC value. This method ensures that Alice does not reveal the secret number K to anybody. However, it's important to note that this method assumes that the communication channel is secure and no man-in-the-middle attack can occur. If the channel is not secure, additional measures like using a secure protocol (e.g., SSL/TLS) should be taken

11. (5 Points) Explain the Merkle-Damgard construction method for hash algorithms, with the help of a simple diagram.



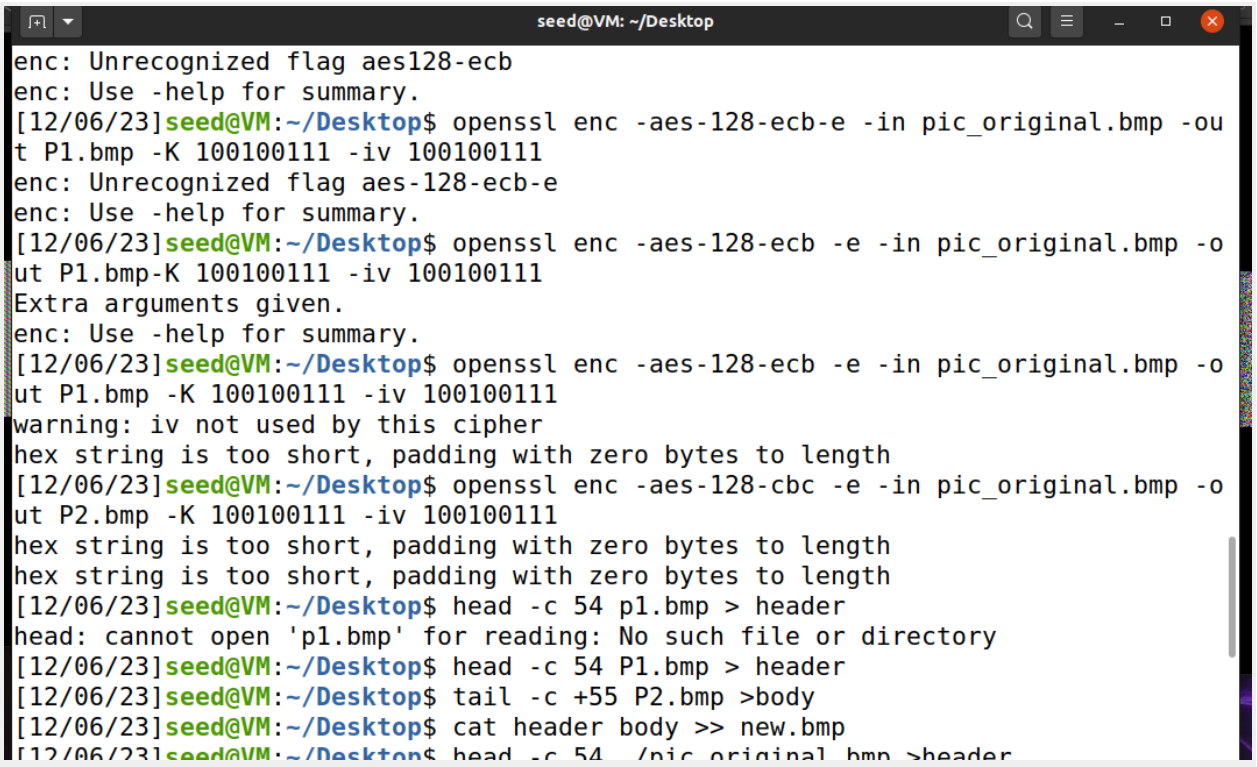
Merkle-Damgard construction method is great for creating a collision resistant hash function. The input message is padded to a length of multiple block size of comrepssion function and it is divided into different blocks that are equal fixed size. The hash algorithm is started with a value

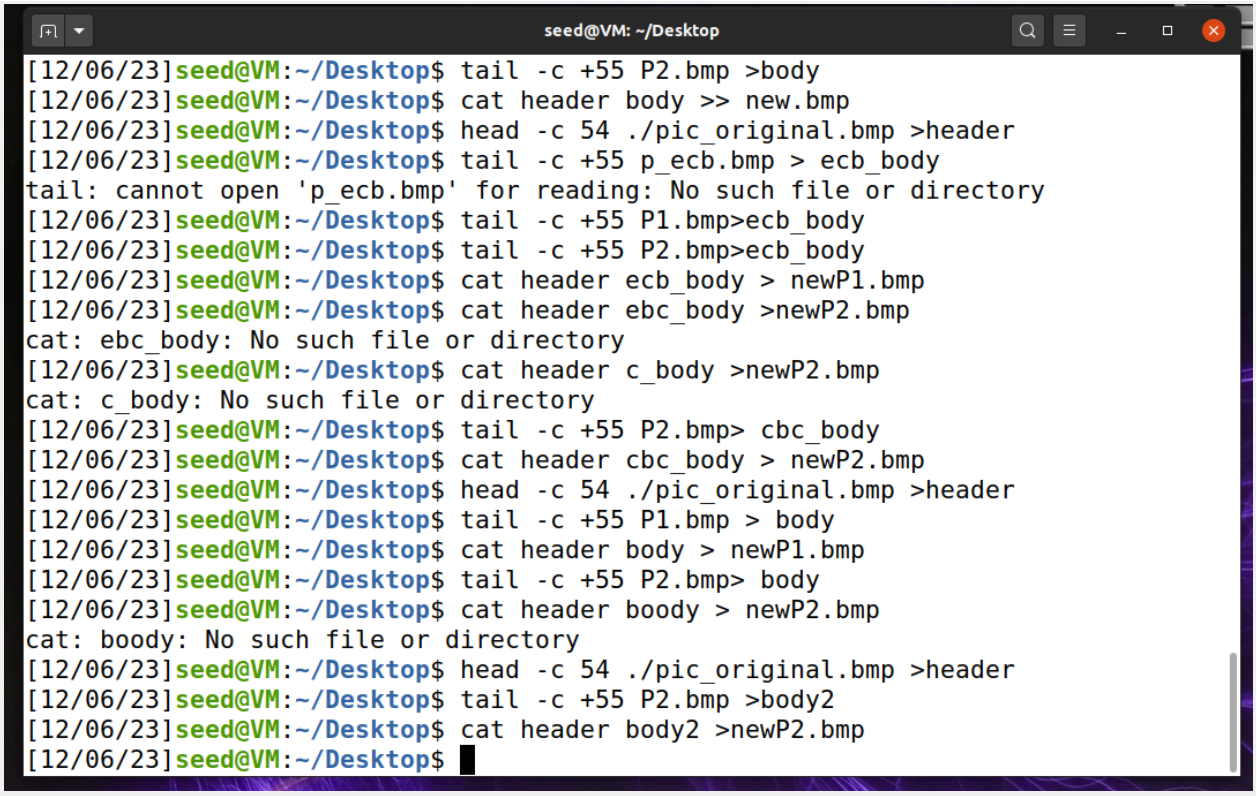
IV which is a fixed value. The finalization function is to final hash the last block of message. The method is efficient and secure for building a collision resistan hash function.

12. (10 Points) If Bob happens to find two different messages M1 and M2 (each has 64 bytes),such that SHA256(M1) = SHA256(M2). Can you find another pair M3 and M4, such that SHA256(M3) = SHA256(M4)?

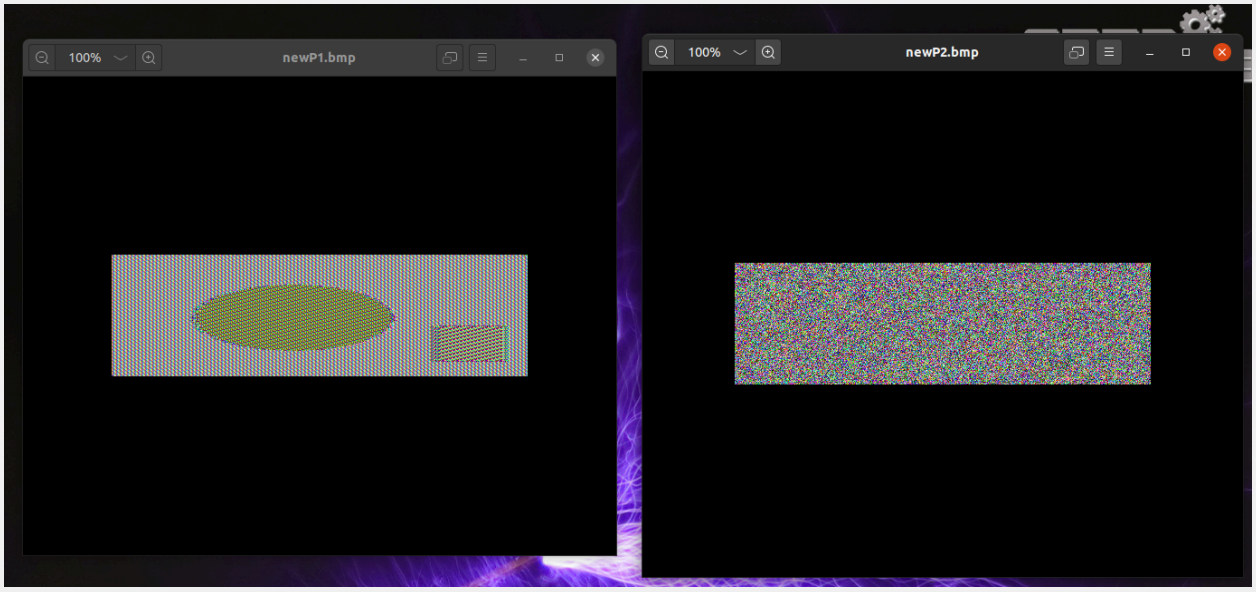
Yes, it is possible to find another pair of messages with the same SHA-256 hash value. This is because SHA-256 is a collision-resistant hash function, but not a preimage-resistant hash function. This means that it is very difficult to find two different messages with the same hash value, but it is not impossible. One way to find such a pair of messages is to use a technique called a second preimage attack. This involves finding a message M3 that has the same hash value as M2. Once M3 is found, it is easy to find a message M4 that has the same hash value as M1. For example, to find a collision pair using the method I mentioned. First, Find a message M2 with a known hash value.Modify M2 to create a new message M3. Calculate the hash value of M3. If the hash value of M3 is equal to the hash value of M2, then you have found a collision pair. If the hash value of M3 is not equal to the hash value of M2, then repeat steps 2-4 until you find a collision pair.

Section 2
1.a)


seed@VM: ~/Desktop
enc: Unrecognized flag aes128-ecb
enc: Use -help for summary.
[12/06/23]seed@VM:~/Desktop$ openssl enc -aes-128-ecb-e -in pic_original.bmp -ou
t P1.bmp -K 100100111 -iv 100100111
enc: Unrecognized flag aes-128-ecb-e
enc: Use -help for summary.
[12/06/23]seed@VM:~/Desktop$ openssl enc -aes-128-ecb -e -in pic_original.bmp -o
ut P1.bmp-K 100100111 -iv 100100111
Extra arguments given.
enc: Use -help for summary.
[12/06/23]seed@VM:~/Desktop$ openssl enc -aes-128-ecb -e -in pic_original.bmp -o
ut P1.bmp -K 100100111 -iv 100100111
warning: iv not used by this cipher
hex string is too short, padding with zero bytes to length
[12/06/23]seed@VM:~/Desktop$ openssl enc -aes-128-cbc -e -in pic_original.bmp -o
ut P2.bmp -K 100100111 -iv 100100111
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
[12/06/23]seed@VM:~/Desktop$ head -c 54 p1.bmp > header
head: cannot open 'p1.bmp' for reading: No such file or directory
[12/06/23]seed@VM:~/Desktop$ head -c 54 P1.bmp > header
[12/06/23]seed@VM:~/Desktop$ tail -c +55 P2.bmp >body
[12/06/23]seed@VM:~/Desktop$ cat header body >> new.bmp


seed@VM: ~/Desktop
[12/06/23]seed@VM:~/Desktop$ tail -c +55 P2.bmp >body
[12/06/23]seed@VM:~/Desktop$ cat header body >> new.bmp
[12/06/23]seed@VM:~/Desktop$ head -c 54 ./pic_original.bmp >header
[12/06/23]seed@VM:~/Desktop$ tail -c +55 p_ecb.bmp > ecb_body
tail: cannot open 'p_ecb.bmp' for reading: No such file or directory
[12/06/23]seed@VM:~/Desktop$ tail -c +55 P1.bmp>ecb_body
[12/06/23]seed@VM:~/Desktop$ tail -c +55 P2.bmp>ecb_body
[12/06/23]seed@VM:~/Desktop$ cat header ecb_body > newP1.bmp
[12/06/23]seed@VM:~/Desktop$ cat header ebc_body >newP2.bmp
cat: ebc_body: No such file or directory
[12/06/23]seed@VM:~/Desktop$ cat header c_body >newP2.bmp
cat: c_body: No such file or directory
[12/06/23]seed@VM:~/Desktop$ tail -c +55 P2.bmp> cbc_body
[12/06/23]seed@VM:~/Desktop$ cat header cbc_body > newP2.bmp
[12/06/23]seed@VM:~/Desktop$ head -c 54 ./pic_original.bmp >header
[12/06/23]seed@VM:~/Desktop$ tail -c +55 P1.bmp > body
[12/06/23]seed@VM:~/Desktop$ cat header body > newP1.bmp
[12/06/23]seed@VM:~/Desktop$ tail -c +55 P2.bmp> body
[12/06/23]seed@VM:~/Desktop$ cat header boody > newP2.bmp
cat: boody: No such file or directory
[12/06/23]seed@VM:~/Desktop$ head -c 54 ./pic_original.bmp >header
[12/06/23]seed@VM:~/Desktop$ tail -c +55 P2.bmp >body2
[12/06/23]seed@VM:~/Desktop$ cat header body2 >newP2.bmp
[12/06/23]seed@VM:~/Desktop$

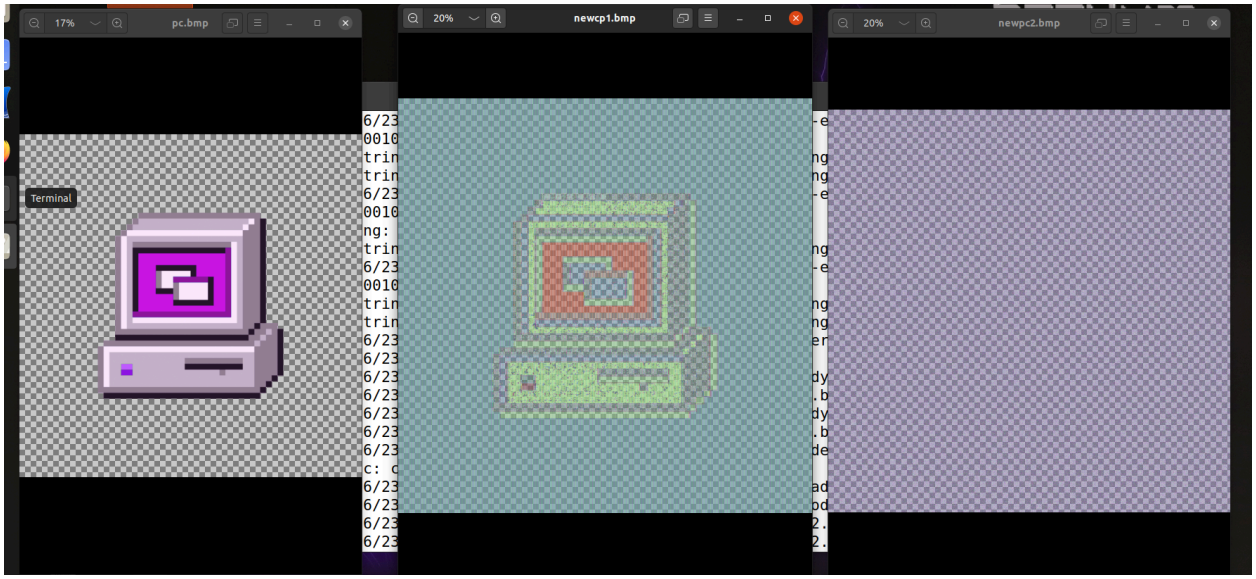
b)



When you use ECB mode to encrypt a picture and then view it, you might notice that some parts of the original image are still visible, but it's not clear enough to identify what the picture is. This happens because ECB encrypts data in blocks independently, so if there are identical blocks in the original picture, they'll look the same in the encrypted version. This can reveal patterns from the original picture, potentially giving away information.

On the other hand, if you use CBC (Cipher Block Chaining) mode and view the encrypted picture, it looks completely jumbled up, and you can't make out any details from the original image. In CBC mode, each block of data is XORed with the previous encrypted block before encryption, and an initialization vector is used. This prevents identical blocks in the original picture from producing identical blocks in the encrypted version. So, CBC mode is considered more secure than ECB because it better protects against certain types of attacks.

2.



The first is the original bmp picture. Second one is ecb mode encryption and the last one is CBC mode encryption. Below is the step i use to encrypt the picture using the same step as question 1.

```
[12/06/23]seed@VM:~/Desktop$ head -c 54 pc1.bmp> header
[12/06/23]seed@VM:~/Desktop$ head -c 54 ./pc.bmp> header
[12/06/23]seed@VM:~/Desktop$ openssl enc -aes-128-ecb -e -in pc.bmp -out pc1.bmp -K 1001011 -iv 0010011
warning: iv not used by this cipher
hex string is too short, padding with zero bytes to length
[12/06/23]seed@VM:~/Desktop$ penssl enc -aes-128-ecb -e -in pc.bmp -out pc1.bmp -K 1001011 -iv 0010011

Command 'penssl' not found, did you mean:

  command 'openssl' from deb openssl (1.1.1f-1ubuntu2)

Try: sudo apt install <deb name>

[12/06/23]seed@VM:~/Desktop$ openssl enc -aes-128-cbc -e -in pc.bmp -out pc2.bmp -K 1001011 -iv 0010011
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
[12/06/23]seed@VM:~/Desktop$ openssl enc -aes-128-ecb -e -in pc.bmp -out pc1.bmp -K 1001011 -iv 0010011
warning: iv not used by this cipher
hex string is too short, padding with zero bytes to length
[12/06/23]seed@VM:~/Desktop$ openssl enc -aes-128-cbc -e -in pc.bmp -out pc2.bmp -K 1001011 -iv 0010011
hex string is too short, padding with zero bytes to length
hex string is too short, padding with zero bytes to length
[12/06/23]seed@VM:~/Desktop$ head -c 54 ./pc.bmp> header
[12/06/23]seed@VM:~/Desktop$ head -c 54 pc.bmp>header
[12/06/23]seed@VM:~/Desktop$ tail -c +55 pc1.bmp>pc1body
[12/06/23]seed@VM:~/Desktop$ cat header pc1body>newcp1.bmp
[12/06/23]seed@VM:~/Desktop$ head -c +55 pc2.bmp>pc2body
[12/06/23]seed@VM:~/Desktop$ cat header pc2body>newpc2.bmp
[12/06/23]seed@VM:~/Desktop$ head-c 54 pc.bmp > pc2header
head-c: command not found
[12/06/23]seed@VM:~/Desktop$ head -c 54 pc.bmp > pc2header
[12/06/23]seed@VM:~/Desktop$ tail -c +55 pc2.bmp> pc2body
[12/06/23]seed@VM:~/Desktop$ cat header pc2body> newpc2.bmp
[12/06/23]seed@VM:~/Desktop$ '/home/seed/Desktop/newpc2.bmp'
```