E.1.

The flag is at the bottom of the question paper.

E.2.

I reversed the shift by modifying the given program to subtract k + i (mod 26) from the ciphertext. Since there are only 27 possible keys, it is trivial to brute force each key by enumerating from 0 to 26.

E.3.

I used openssl to decrypt the file.

```
openssl enc -d -aes-128-cbc -K [hexadecimal key] -iv [hexadecimal iv] -in encrypted.jpg -out decrypted
```

E.4.

I used openssl to generate the MAC of the file.

```
openssl dgst -hmac [key] -md5 test.txt
```

E.5.

I used an online RSA calculator to decrypt the ciphertext.

M.1.

Since every byte of the plaintext will NEVER map to itself in the ciphertext, then we can deduce the plaintext byte by elimination from a bunch of ciphertexts.

```
ciphertexts = open('ciphertexts.txt', 'r')
reference_line = bytes.fromhex(ciphertexts.readline())
num_bytes = len(reference_line)

plaintext_array = [b'0' for _ in range(num_bytes)]
possible_bytes = [{i.to_bytes(1, 'little') for i in range(256)} for _ in range(num_bytes)]
remaining_bytes = num_bytes

for ciphertext in ciphertexts:
    ciphertext_bytes = bytes.fromhex(ciphertext)
    for index, ciphertext_byte in enumerate(ciphertext_bytes):
        if len(possible_bytes[index]) < 1:
            continue
        possible_bytes[index].discard(ciphertext_byte.to_bytes(1, 'little'))
        if len(possible_bytes[index]) == 1:
            plaintext_array[index] = possible_bytes[index].pop()
            remaining_bytes -= 1

    if remaining_bytes < 1:
        break

print(b''.join(plaintext_array).decode())
```

M.2.

I wrote a python script to brute force every possible 8-digit key to decrypt the ciphertext.

```
from Crypto.Cipher import AES

ciphertext = 0x7d294f3c7a71e2808e8ce5afd9eb99c343460ca9f5f89ff062fed96b.to_bytes(28, byteorder='big')
NONCE = '0000000000'
PADDING = '00000000'

for k in range(100000000):
    num_leading_zeros = 8 - len(str(k))
    key_string = ''.join(['0' for _ in range(num_leading_zeros)]) + str(k) + PADDING
    cipher = AES.new(key_string.encode(), AES.MODE_CTR, initial_value=0, nonce=NONCE.decode())
    plaintext = cipher.decrypt(ciphertext).decode()
    try:
        print(plaintext.decode())
    except:
        continue
```

H.2.

First, I observed that by sending the server 'CS2107{' (translated to hexadecimal), the server returns a shorter ciphertext than if I send a sequence of random bytes of the same length. Upon inspection of the server side code, it made sense because the

plaintext contains the leading substring 'CS2107{', which will be compressed before encryption. Extending from this train of thought, I wrote a script that tries every ASCII character and observe the length of the returned ciphertext. For each position, I will select the character that returns the shortest ciphertext and append it to my proposed plaintext. By deduction, this character must appear in the plaintext in order to be compressed. This technique returned some words spelt in 7331 (leet), and I was able to construct a sentence from multiple runs of the script, adding the word that has been discovered to the pre-existing plaintext so that it will not be discovered again.

```
 import math
import pexpect

child = pexpect.spawn('nc cs2107-ctfd-i.comp.nus.edu.sg 5054')
prefix = 'CS2107{7his_I5_WHy_C0mpr35si0n_aR3_D!s@ble_by_d3f@ul7_'
plaintext = ''
prev_char = ''
postfix = ''

for i in range(7):
    child.readline()

for i in range(60):
    print('ciphertext of length ' + str(i + 1))
    min_ciphertext_len = math.inf
    candidate_char = ''
    d = {}
    for c in range(32, 127):
        # if chr(c) != prev_char:
        child.sendline((prefix + plaintext + chr(c) + postfix).encode().hex())
        child.readline()
        response = child.readline().decode()
        ciphertext = response[13:]
        d[chr(c)] = len(ciphertext)
        if len(ciphertext) <= min_ciphertext_len:
            candidate_char = chr(c)
            min_ciphertext_len = len(ciphertext)

    plaintext += candidate_char
    prev_char = candidate_char
    print(prefix + plaintext + postfix)
    print(d)

child.close()
```