

3DES: k1加密 -> k2解密 -> k3加密

DES 加密算法使用64位元的區塊大小和56位元的金鑰。在一個金鑰空間中，有256個可能的金鑰，3DES用三個金鑰，所以是256\*3 = 2168種組合

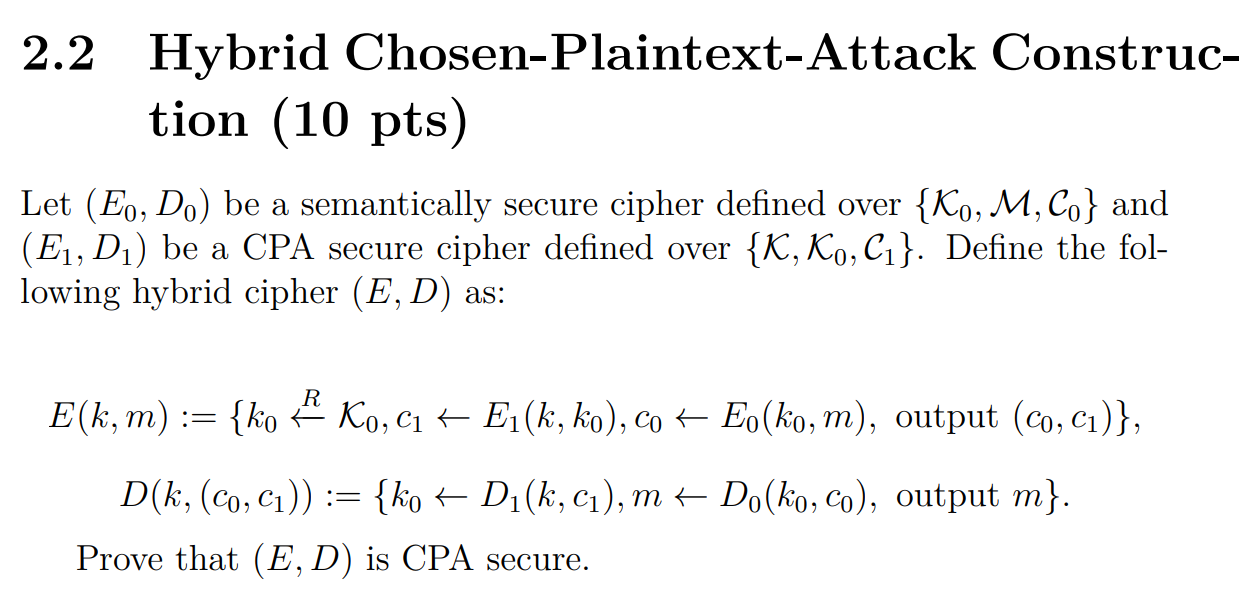
讓3DES等同於只用一把金鑰的可能情況:

情況1. 如果k1=k2，相當於只用k3加密

情況2. 如果k2=k3，相當於只用k1加密

如果k1, k2, k3是各自獨立選擇，情況1和情況2的機率就都是1/256\*2 = 1/2112

總體發生的機率是2/2112，小於1/288，所以可以忽略



CPA安全性要求對於任何給定的兩個m0和m1，如果攻擊者只能進行加密和解密操作，那麼他在區分兩個c0和c1的能力應該是可忽略的

用反證法證明(E,D)是CPA安全：

設(E,D)不安全(存在一個CPA攻擊者可以在可忽略的時間內區分兩個密文)

建構攻擊者A’，它是兩個子加密方案E0和E1的攻擊者

用A的結果區分(E,D)的密文 -> 違反(E0,D0)和(E1,D1)的安全性假設

由於A’和A的時間複雜度相同，且(E0,D0)和(E1,D1)是安全的

所以A’也無法區分(E,D)的密文

因此，如果(E0,D0)和(E1,D1)抗CPA攻擊，那(E,D)也是CPA安全

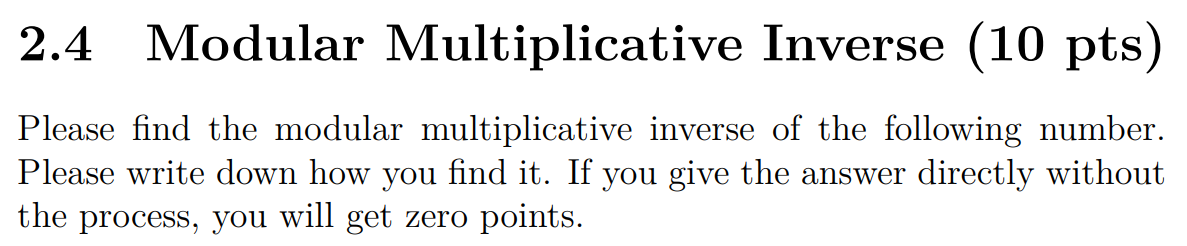


1. 解密c，得到m0, m1, m2,…,mn

2. 修改m0的位元，使m0[0] = m[0]⊕Δ

3. 將修改後的m0和原本的m1, m2,…,mn重新組合成新的m’

4. 將m’進行CBC加密，得到c’





**找到b使得 400b≡1(mod997)**

997 = 2\*400 + 197

400 = 2\*197 + 6

197 = 32\*6 + 5

6 = 1\*5 + 1

​**反回去**

1 = 6 − 1\*5

= 6 – 1\*(197 − 32\*6)

= 33\*6 – 1\*197

= 33\*(400 – 2\*197) – 1\*197

= 33\*400 – 67\*197

= 33\*400 – 67\*(997 − 2\*400)

= 167\*400 – 67\*997

​b = 167



**找到b使得 472b≡1(mod16651)**

16651 =35\*472+431

472 = 1\*431 + 41

431 = 10\*41 + 21

41 = 1\*21 + 20

21 = 1\*20 + 1

**反回去**

​1 = 21 – 1\*20

= 21 – 1\*(41 – 1\*21)

= 2\*21 – 1\*41

= 2\*(431 – 10\*41) – 1\*41

= 2\*431 – 21\*41

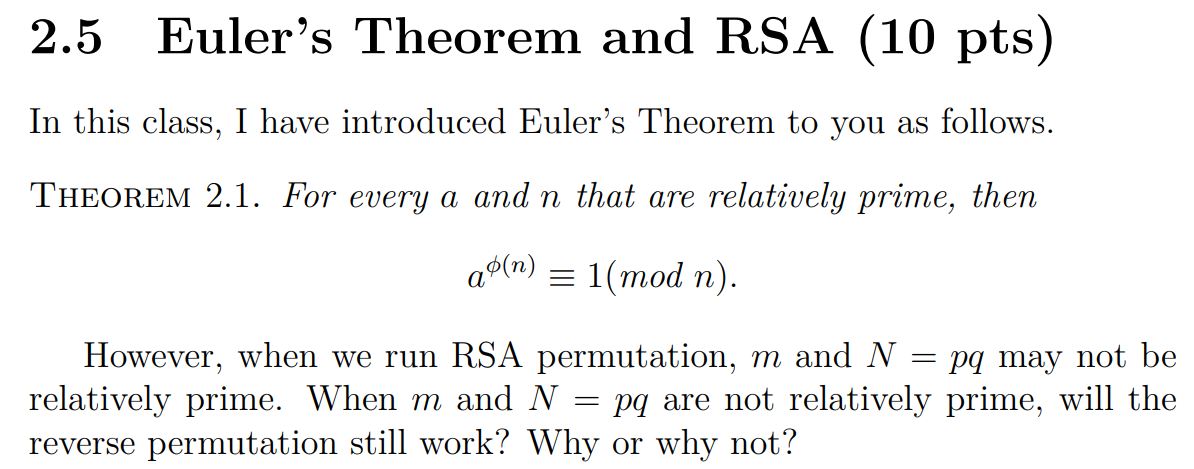
= 2\*431 – 21\*(472 – 1\*431)

= 23\*431 – 21\*472

= 23\*(16651 – 35\*472) – 21\*472

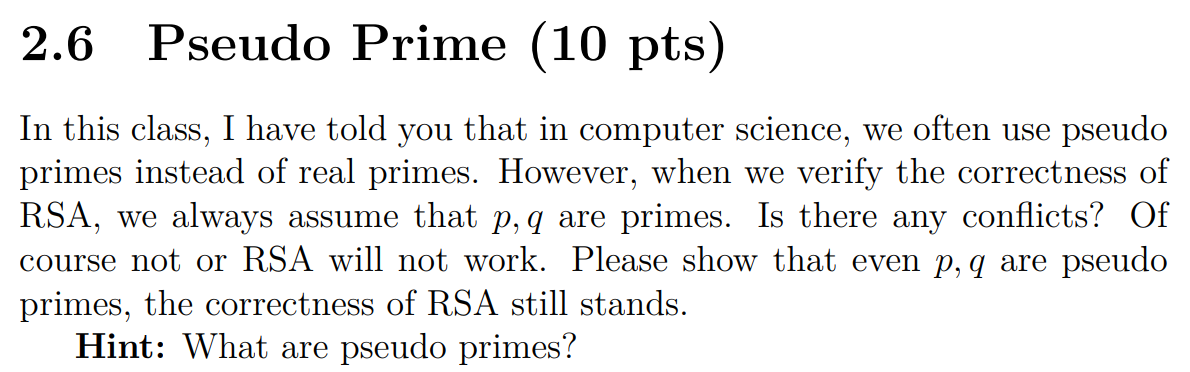
= −803\*472 + 23\*16651

​b = −803 但要找正的，所以是b = 16651−803 = 15848

​

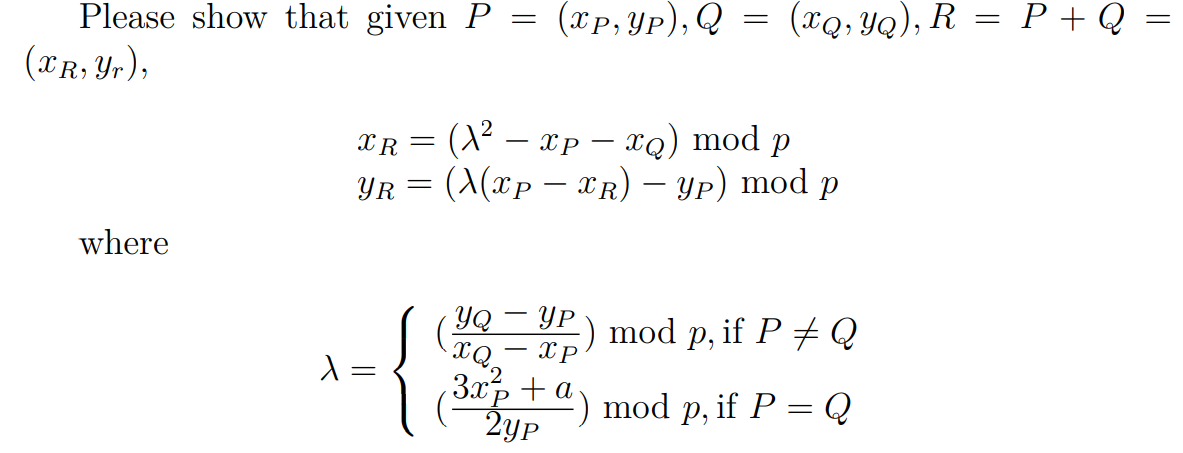
在RSA加密中，我們選擇兩個大質數p和q，然後計算它們的乘積N = pq作為模數，當m和N不互質(m不是N的倍數)，Euler's Theorem不再適用

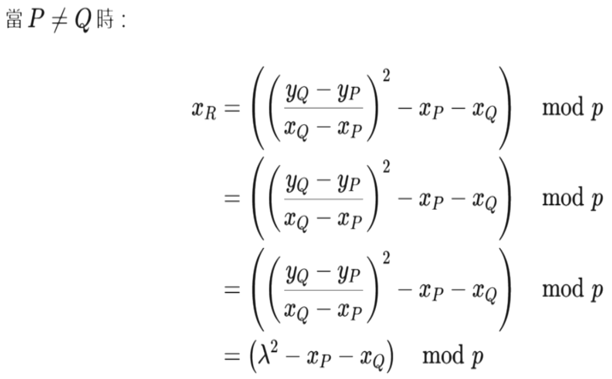
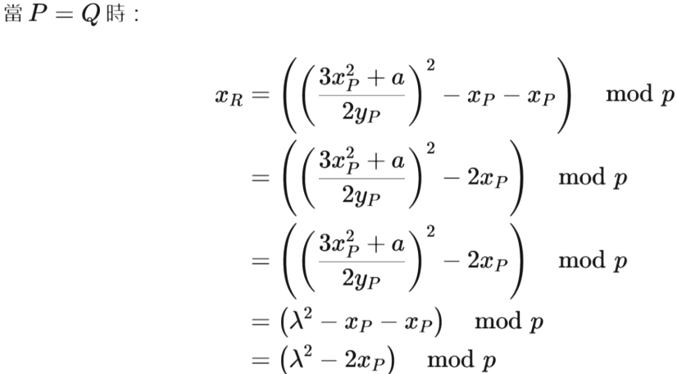
因為Euler's Theorem僅適用於互質的a和n，這樣才能確保 aϕ(n) ≡ 1(mod n)，當m和N不互質時，m就不會滿足Euler's Theorem的條件，因此反向置換不正確

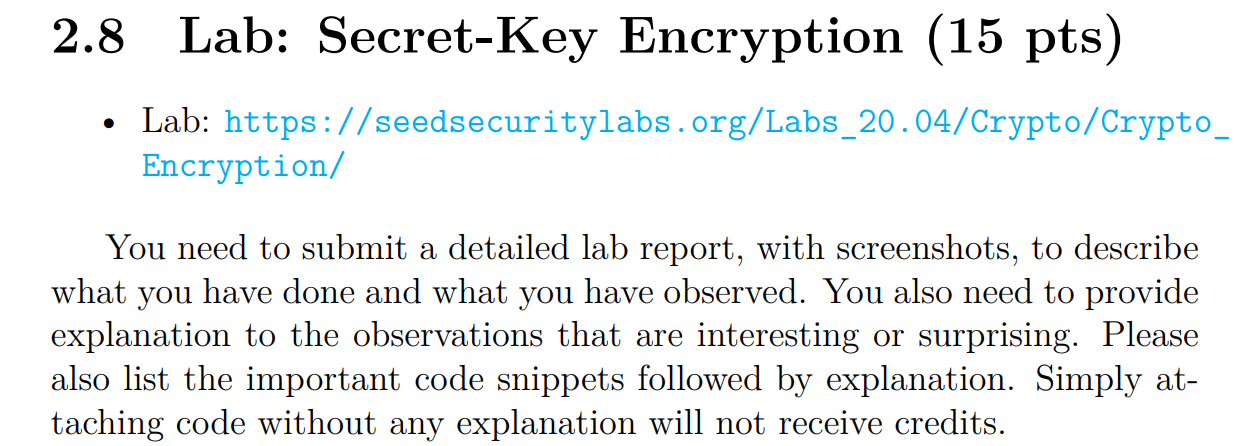


因為RSA難解的原因是N = pq的p和q不好找，不是因為其他特殊的質數性質，所以只要p和q夠大，加上N的因數分解夠困難，就會讓RSA達到安全的效果。





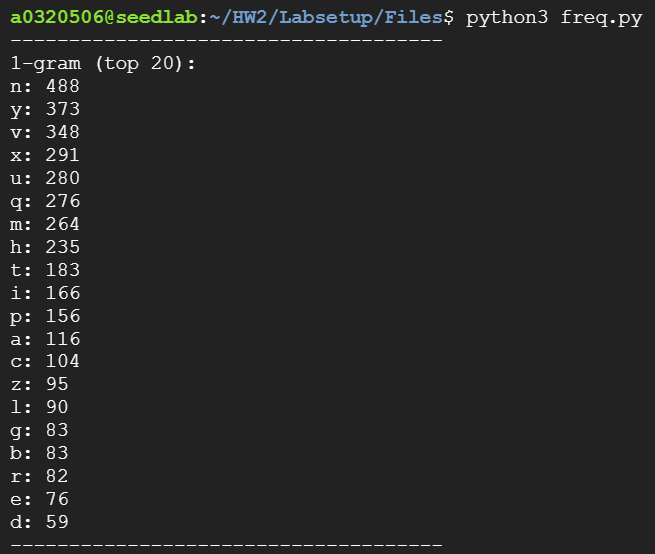
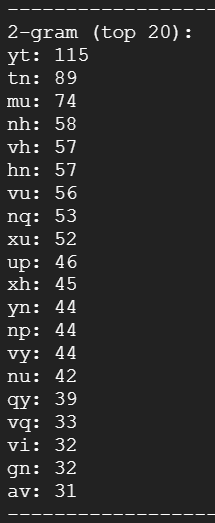
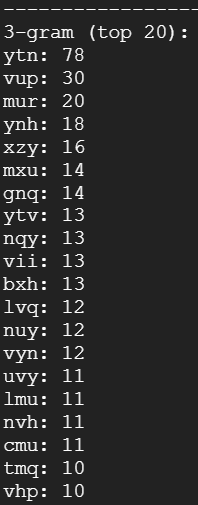


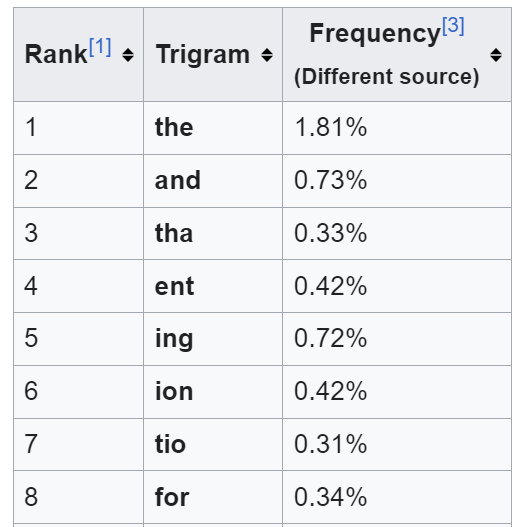
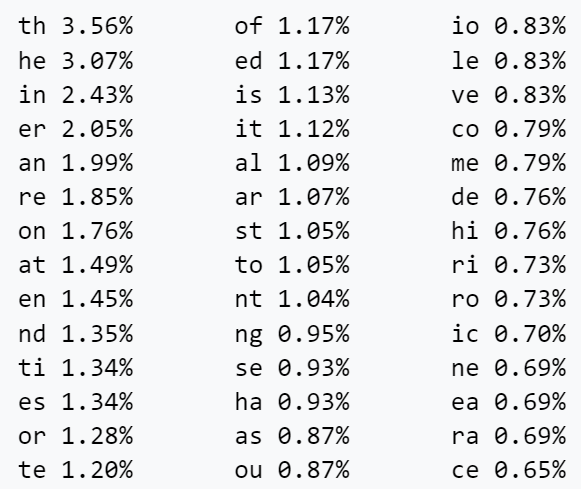
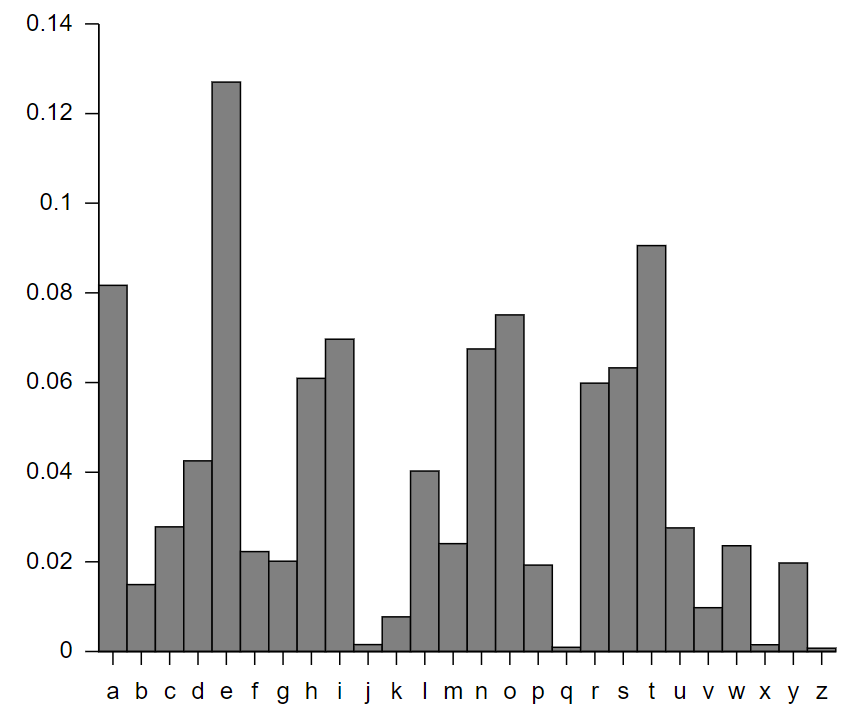










解密過程

sudo tr ’mnrtuvy’ ’IEGHNAT’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvybp’ ’IEGHNATSD’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvybpx’ ’IEGHNATSDO’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvybpxh’ ’IEGHNATSDOL’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxh’ ’IEGHNATSDOL’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhb’ ’IEGHNATSDOLF’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhb’ ’IEGHNATSDORF’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbi’ ’IEGHNATSDORFL’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbiz’ ’IEGHNATSDORFLU’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizg’ ’IEGHNATSDORFLUB’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgd’ ’IEGHNATSDORFLUBY’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgds’ ’IEGHNATSDORFLUBYK’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgdsl’ ’IEGHNATSDORFLUBYKW’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgdslc’ ’IEGHNATSDORFLUBYKWC’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgdsla’ ’IEGHNATSDORFLUBYKWC’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgdslac’ ’IEGHNATSDORFLUBYKWCM’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgdslace’ ’IEGHNATSDORFLUBYKWCMP’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgdslacef’ ’IEGHNATSDORFLUBYKWCMPV’ <ciphertext.txt> message.txt

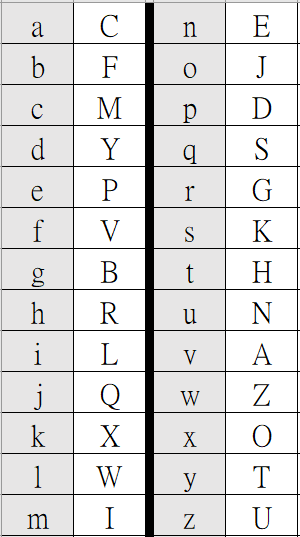
sudo tr ’mnrtuvyqpxhbizgdslacefj’ ’IEGHNATSDORFLUBYKWCMPVQ’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgdslacefjw’ ’IEGHNATSDORFLUBYKWCMPVQM’ <ciphertext.txt> message.txt

sudo tr ’mnrtuvyqpxhbizgdslacefjwo’ ’IEGHNATSDORFLUBYKWCMPVQMJ’ <ciphertext.txt> message.txt

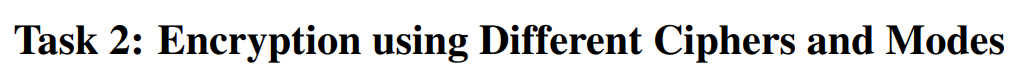
sudo tr ’mnrtuvyqpxhbizgdslacefjwok’ ’IEGHNATSDORFLUBYKWCMPVQMJX’ <ciphertext.txt> message.txt

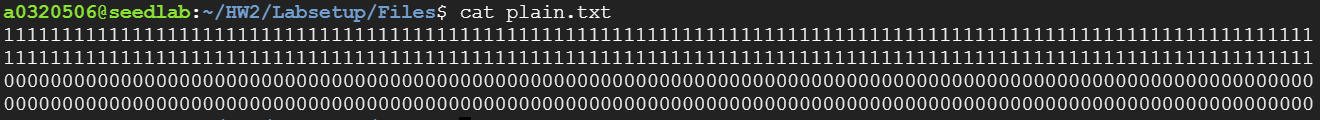
sudo tr ’mnrtuvyqpxhbizgdslacefjwok’ ’IEGHNATSDORFLUBYKWCMPVQZJX’ <ciphertext.txt> message.txt

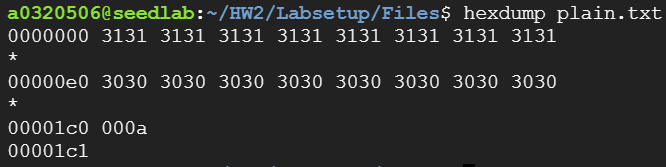
 整理後的對照表

abcdefghijklmnopqrstuvwxyz -> cfmypvbrlqxwiejdsgkhnazotu

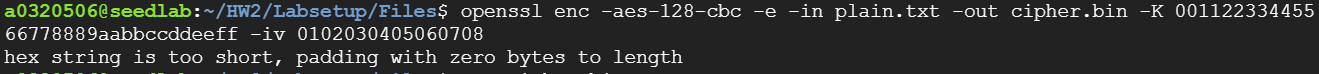
解密結果在message.txt

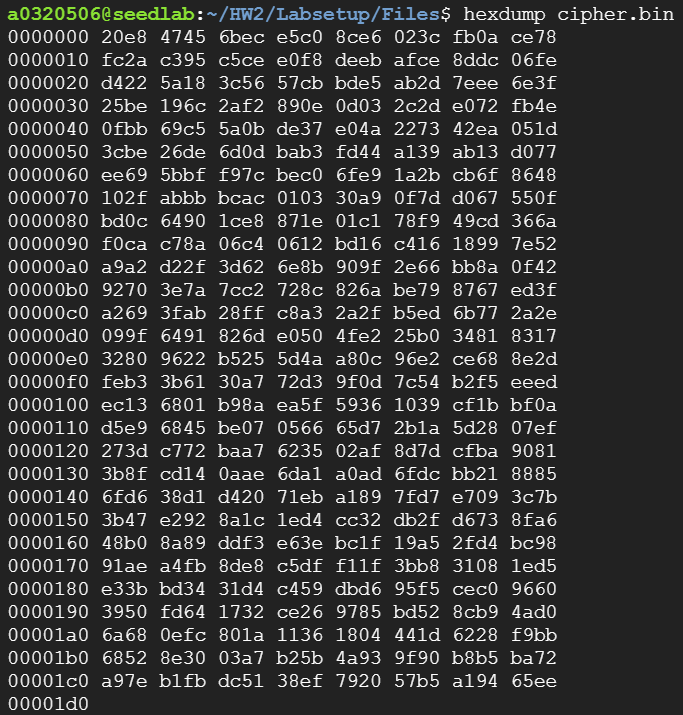






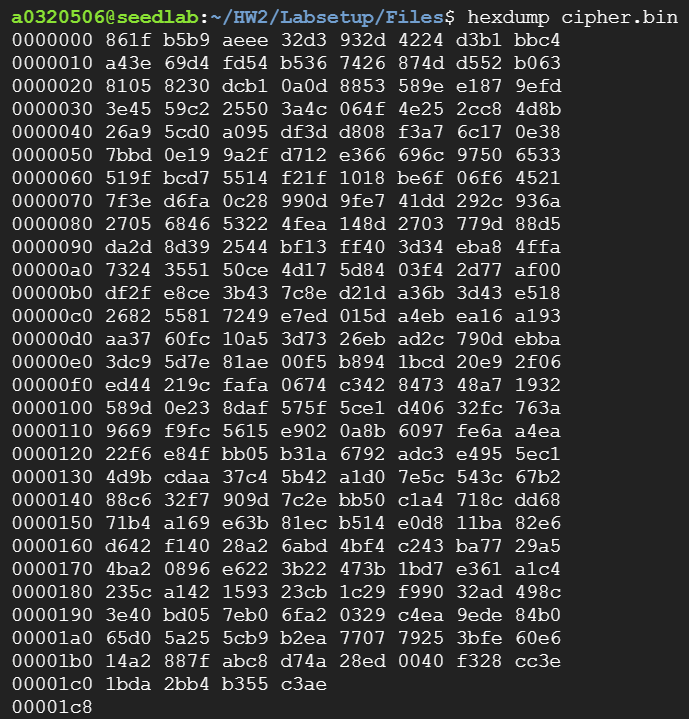
openssl enc -aes-128-cbc -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

****

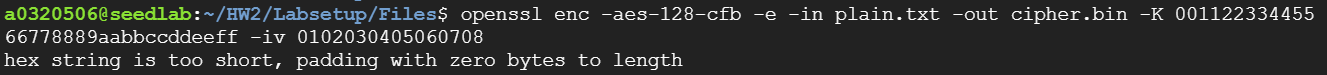
 aes-128-cbc加密結果

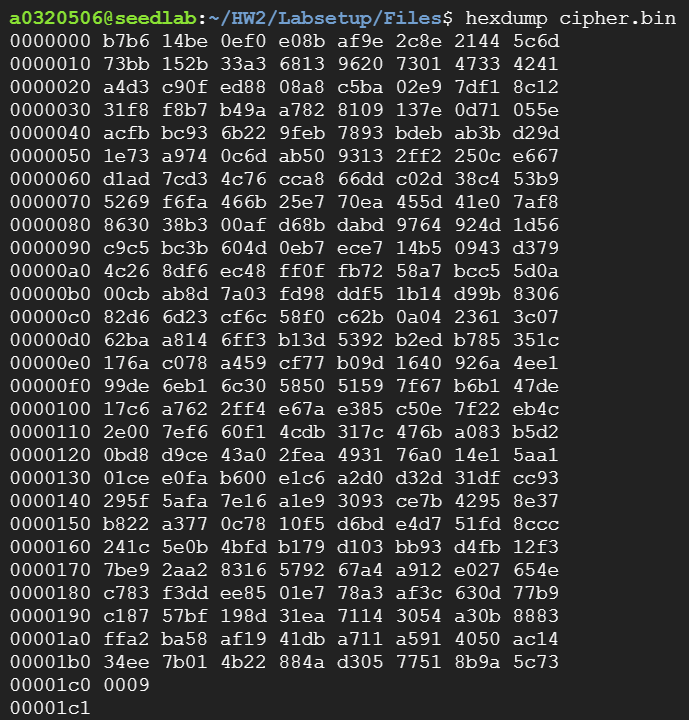
openssl enc -bf-cbc -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

****

**** bf-cbc加密結果

openssl enc -aes-128-cfb -e -in plain.txt -out cipher.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708



 aes-128-cfb加密結果



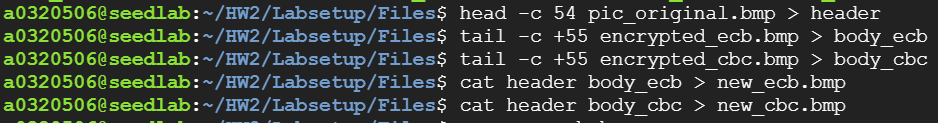
openssl enc -aes-128-ecb -e -in pic\_original.bmp -out encrypted\_ecb.bmp -K 00112233445566778889aabbccddeeff

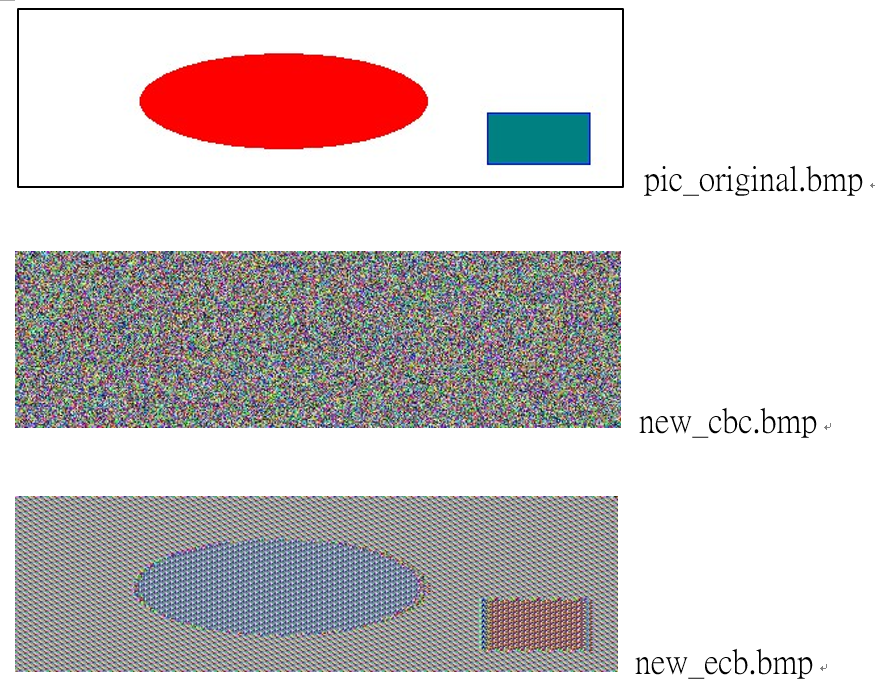


openssl enc -aes-128-cbc -e -in pic\_original.bmp -out encrypted\_cbc.bmp -K 00112233445566778889aabbccddeeff -iv 0102030405060708



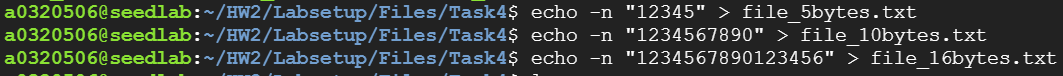
修改標頭



 結果



創建三種長度的原文檔案



**aes-128-cbc**

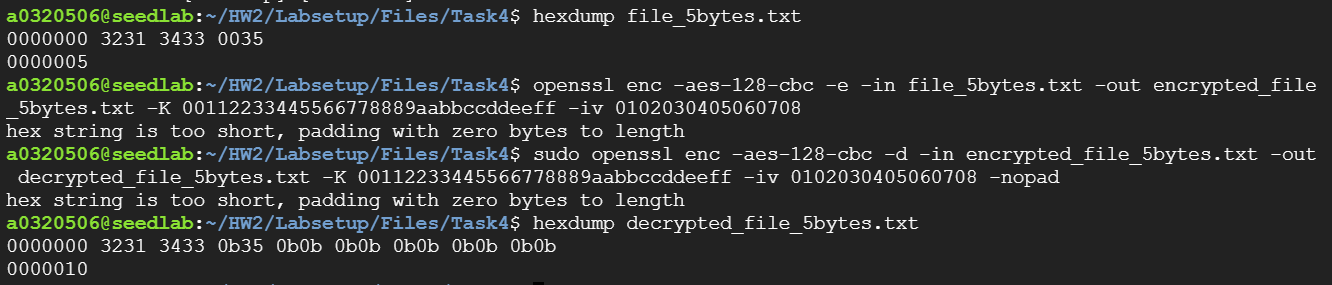
**5 bytes**

hexdump file\_5bytes.txt

openssl enc -aes-128-cbc -e -in file\_5bytes.txt -out encrypted\_file\_5bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-cbc -d -in encrypted\_file\_5bytes.txt -out decrypted\_file\_5bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_5bytes.txt



**aes-128-cbc**

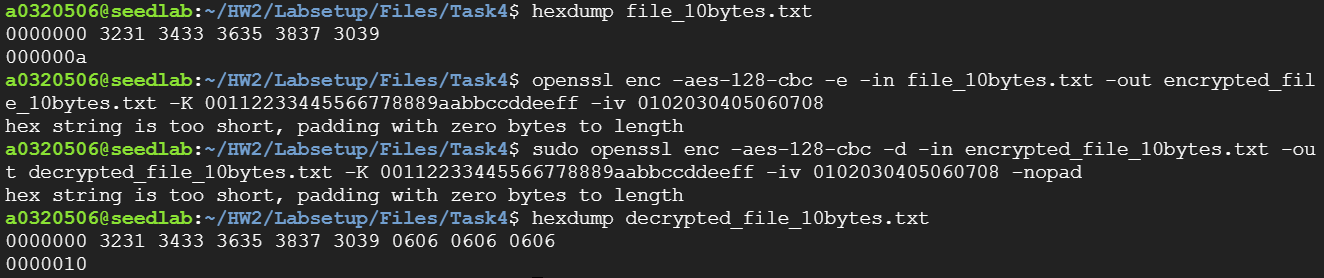
**10 bytes**

hexdump file\_10bytes.txt

openssl enc -aes-128-cbc -e -in file\_10bytes.txt -out encrypted\_file\_10bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-cbc -d -in encrypted\_file\_10bytes.txt -out decrypted\_file\_10bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_10bytes.txt

****

**aes-128-cbc**

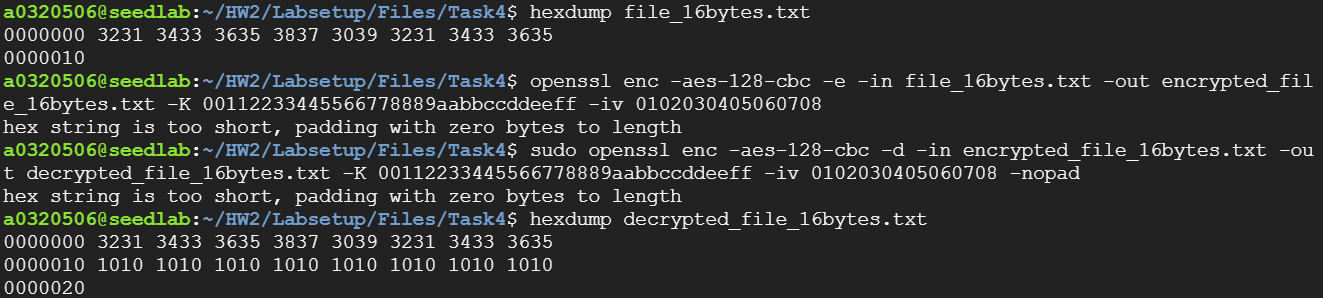
**16 bytes**

hexdump file\_16bytes.txt

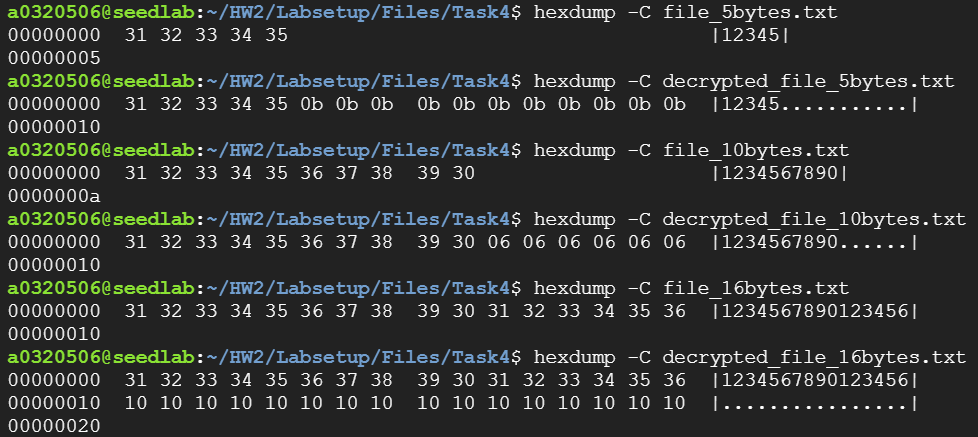
openssl enc -aes-128-cbc -e -in file\_16bytes.txt -out encrypted\_file\_16bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-cbc -d -in encrypted\_file\_16bytes.txt -out decrypted\_file\_16bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_16bytes.txt



**aes-128-cbc 比較**



要填充

**aes-128-ecb**

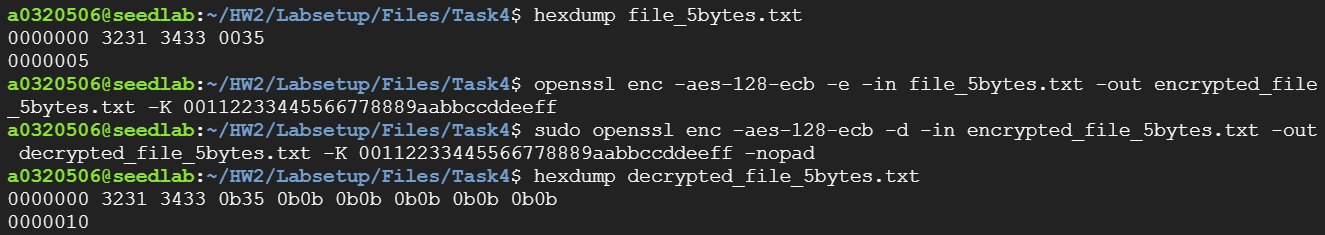
**5 bytes**

hexdump file\_5bytes.txt

openssl enc -aes-128-ecb -e -in file\_5bytes.txt -out encrypted\_file\_5bytes.txt -K 00112233445566778889aabbccddeeff

sudo openssl enc -aes-128-ecb -d -in encrypted\_file\_5bytes.txt -out decrypted\_file\_5bytes.txt -K 00112233445566778889aabbccddeeff -nopad

hexdump decrypted\_file\_5bytes.txt



**aes-128-ecb**

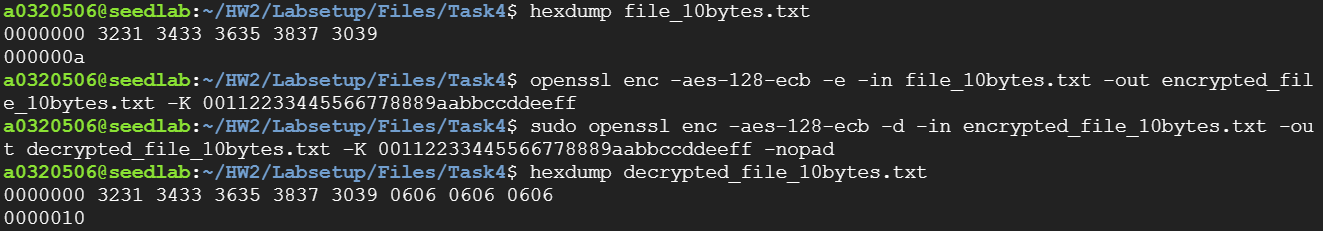
**10 bytes**

hexdump file\_10bytes.txt

openssl enc -aes-128-ecb -e -in file\_10bytes.txt -out encrypted\_file\_10bytes.txt -K 00112233445566778889aabbccddeeff

sudo openssl enc -aes-128-ecb -d -in encrypted\_file\_10bytes.txt -out decrypted\_file\_10bytes.txt -K 00112233445566778889aabbccddeeff -nopad

hexdump decrypted\_file\_10bytes.txt



**aes-128-ecb**

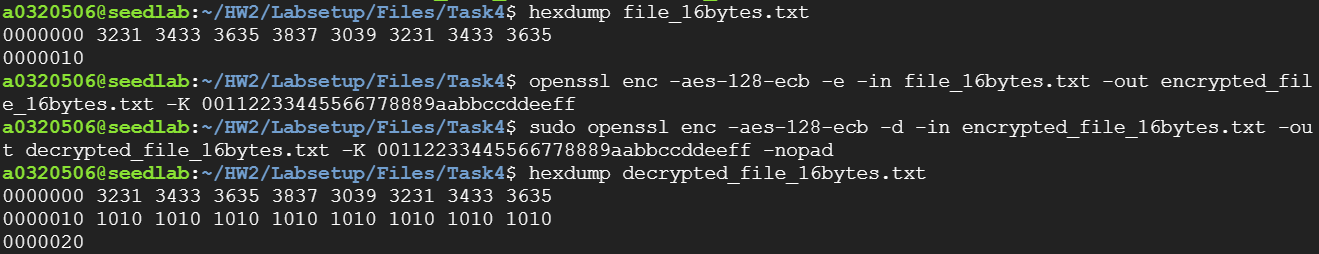
**16 bytes**

hexdump file\_16bytes.txt

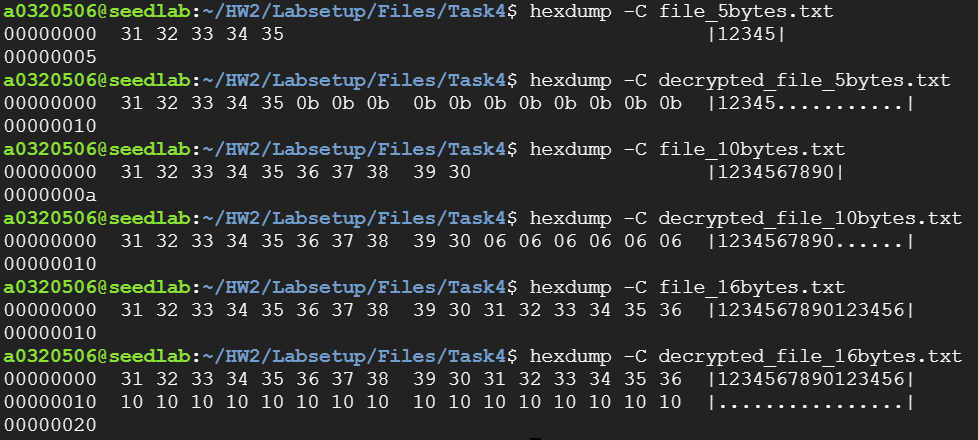
openssl enc -aes-128-ecb -e -in file\_16bytes.txt -out encrypted\_file\_16bytes.txt -K 00112233445566778889aabbccddeeff

sudo openssl enc -aes-128-ecb -d -in encrypted\_file\_16bytes.txt -out decrypted\_file\_16bytes.txt -K 00112233445566778889aabbccddeeff -nopad

hexdump decrypted\_file\_16bytes.txt



**aes-128-ecb 比較**



要填充

**aes-128-cfb**

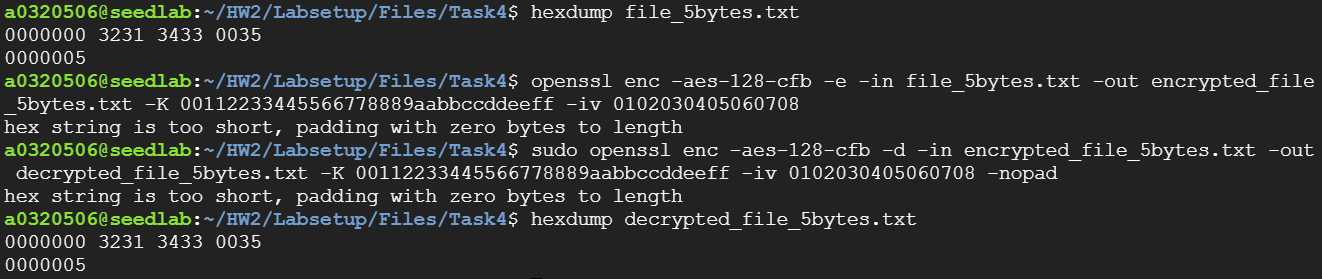
**5 bytes**

hexdump file\_5bytes.txt

openssl enc -aes-128-cfb -e -in file\_5bytes.txt -out encrypted\_file\_5bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-cfb -d -in encrypted\_file\_5bytes.txt -out decrypted\_file\_5bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_5bytes.txt



**aes-128-cfb**

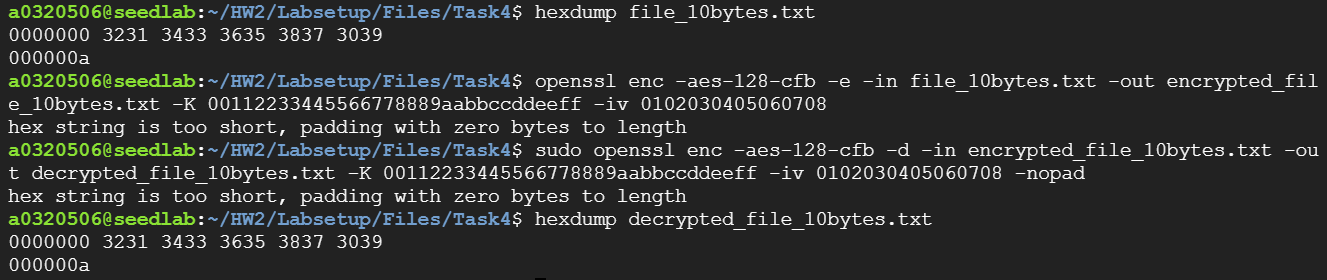
**10 bytes**

hexdump file\_10bytes.txt

openssl enc -aes-128-cfb -e -in file\_10bytes.txt -out encrypted\_file\_10bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-cfb -d -in encrypted\_file\_10bytes.txt -out decrypted\_file\_10bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_10bytes.txt



**aes-128-cfb**

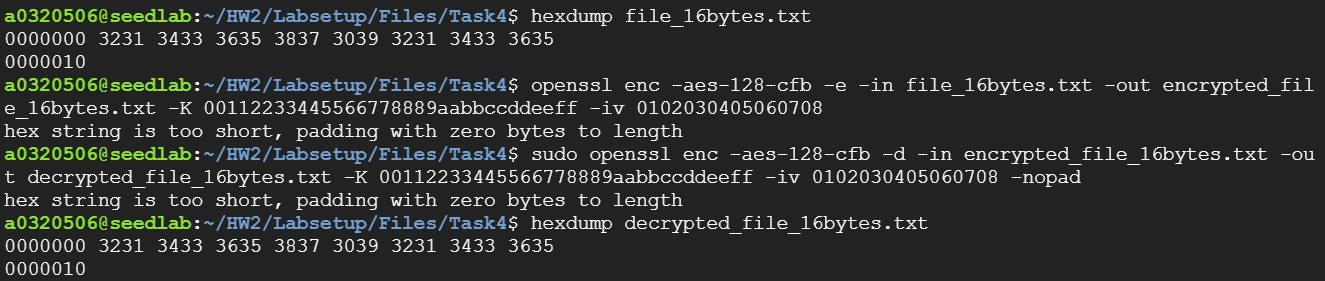
**16 bytes**

hexdump file\_16bytes.txt

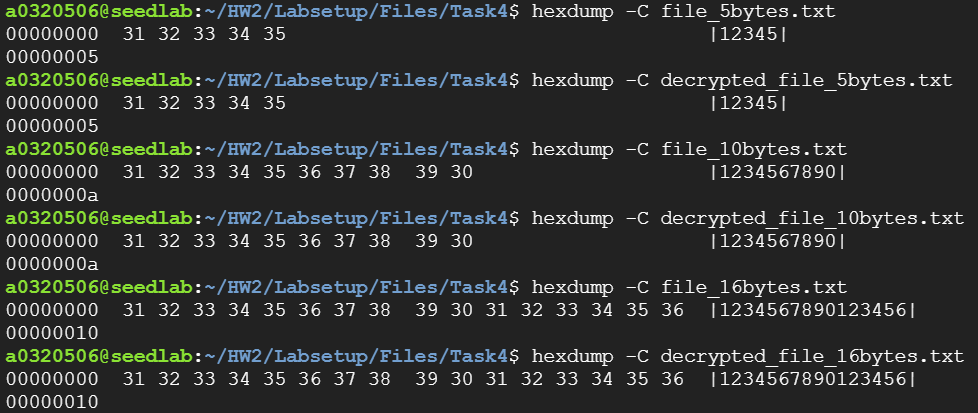
openssl enc -aes-128-cfb -e -in file\_16bytes.txt -out encrypted\_file\_16bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-cfb -d -in encrypted\_file\_16bytes.txt -out decrypted\_file\_16bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_16bytes.txt



**aes-128-cfb 比較**



不填充

**aes-128-ofb**

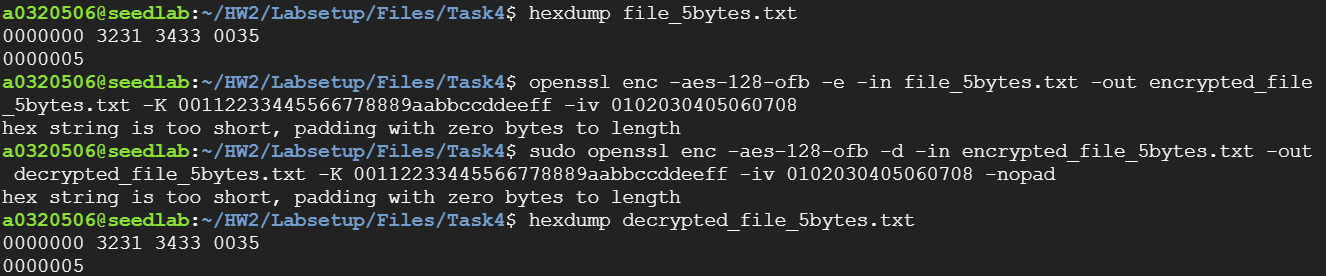
**5 bytes**

hexdump file\_5bytes.txt

openssl enc -aes-128-ofb -e -in file\_5bytes.txt -out encrypted\_file\_5bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-ofb -d -in encrypted\_file\_5bytes.txt -out decrypted\_file\_5bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_5bytes.txt



**aes-128-ofb**

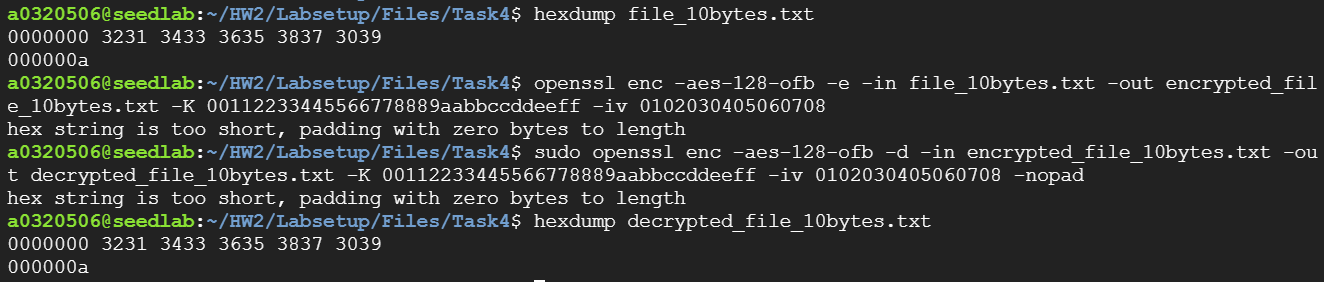
**10 bytes**

hexdump file\_10bytes.txt

openssl enc -aes-128-ofb -e -in file\_10bytes.txt -out encrypted\_file\_10bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-ofb -d -in encrypted\_file\_10bytes.txt -out decrypted\_file\_10bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_10bytes.txt



**aes-128-ofb**

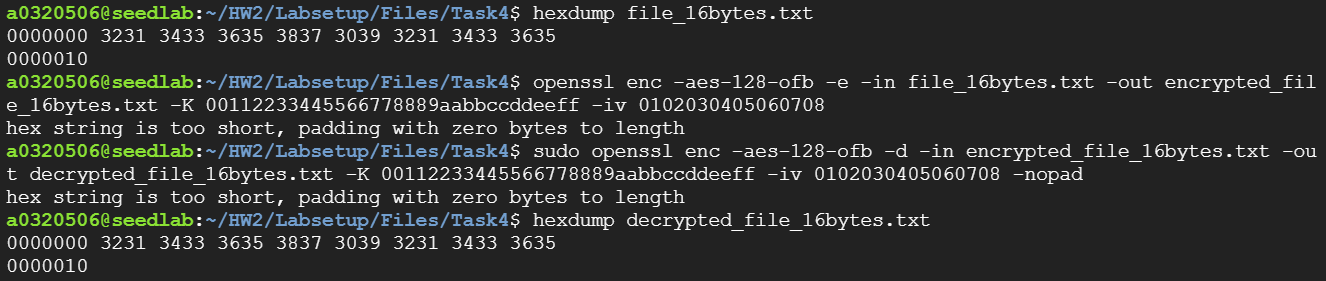
**16 bytes**

hexdump file\_16bytes.txt

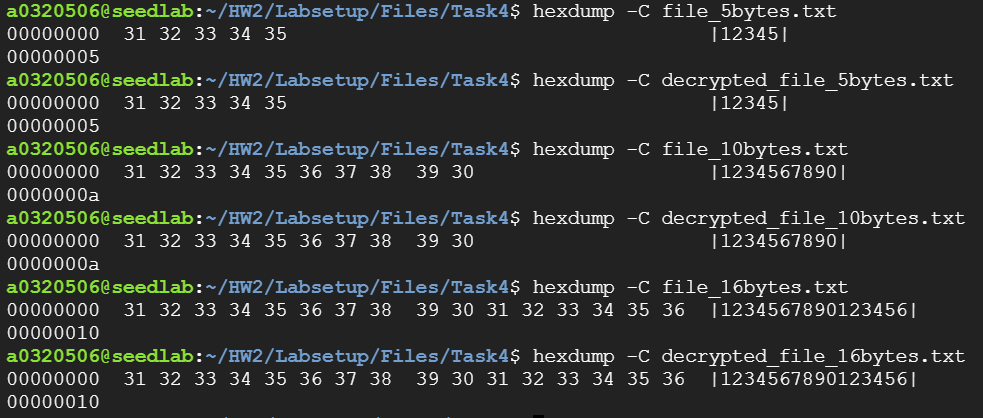
openssl enc -aes-128-ofb -e -in file\_16bytes.txt -out encrypted\_file\_16bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

sudo openssl enc -aes-128-ofb -d -in encrypted\_file\_16bytes.txt -out decrypted\_file\_16bytes.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad

hexdump decrypted\_file\_16bytes.txt



**aes-128-ofb 比較**



不填充

ECB、CBC要填充，因為在他們把明文分成固定大小的塊進行加密。如果明文的大小不是加密演算法要求的固定塊大小的倍數，就需要對明文進行填充，讓每塊的大小相同

CFB、OFB不填充，因為他們的加密是以byte為單位進行，所以即使明文的大小不是固定塊大小的倍數，也不需要填充



文本檔案在1000-bytes.txt

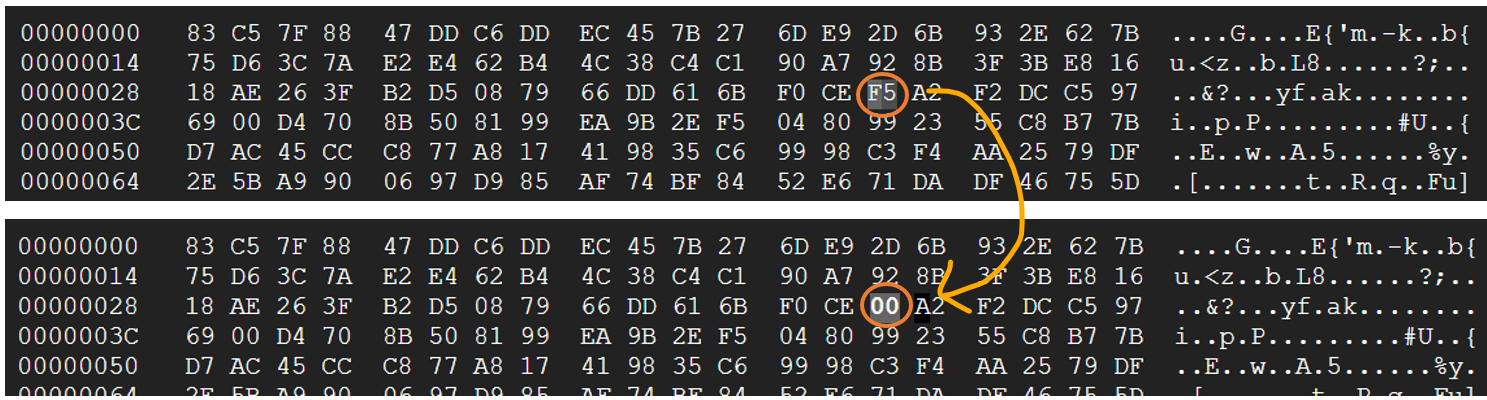
-iv 0102030405060708

**aes-128-ecb**

openssl enc -aes-128-ecb -e -in 1000-bytes.txt -out encrypted\_file.bin -K 00112233445566778889aabbccddeeff

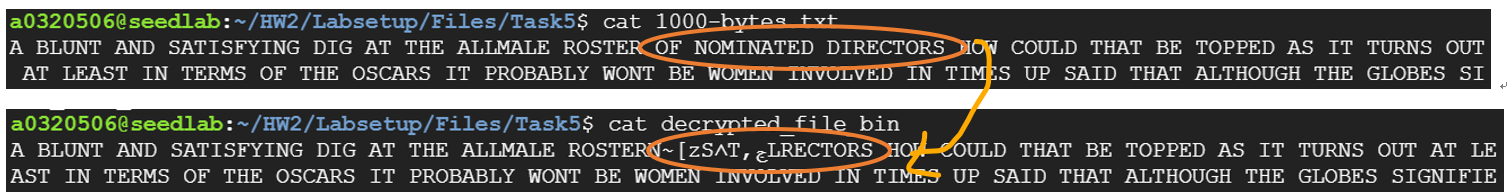






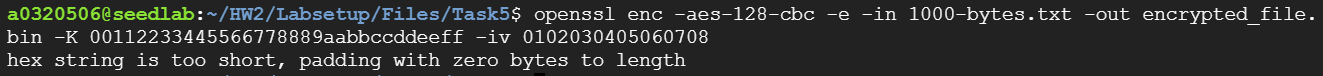
sudo openssl enc -aes-128-ecb -d -in encrypted\_file.bin -out decrypted\_file\_bin -K 00112233445566778889aabbccddeeff -nopad





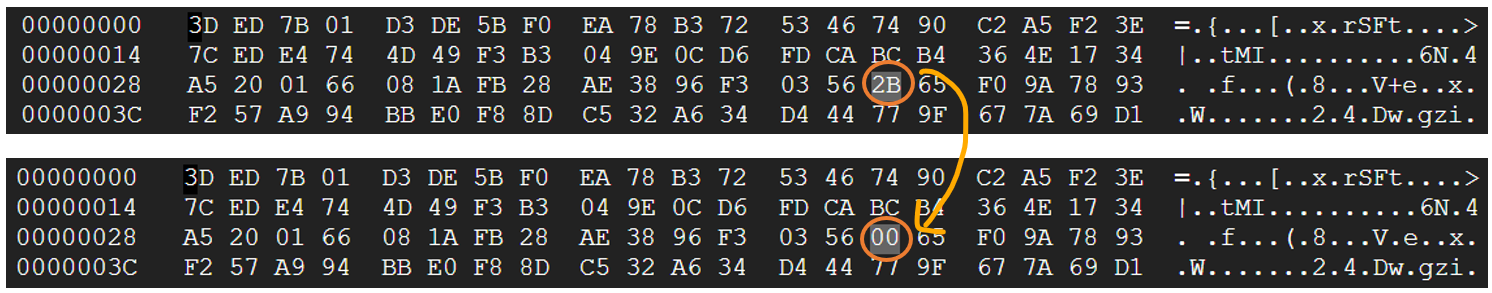
**aes-128-cbc**

openssl enc -aes-128-cbc -e -in 1000-bytes.txt -out encrypted\_file.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

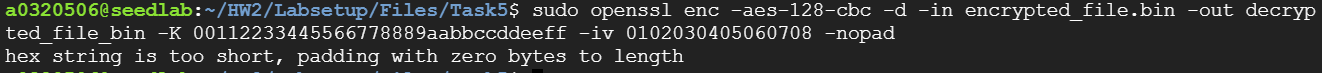


hexedit encrypted\_file.bin



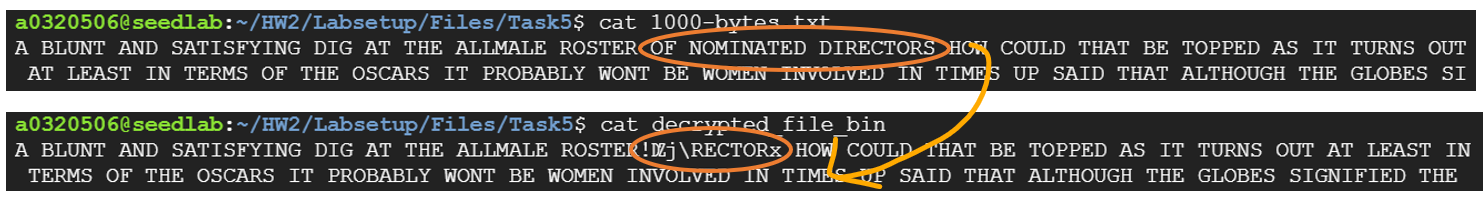


sudo openssl enc -aes-128-cbc -d -in encrypted\_file.bin -out decrypted\_file\_bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad



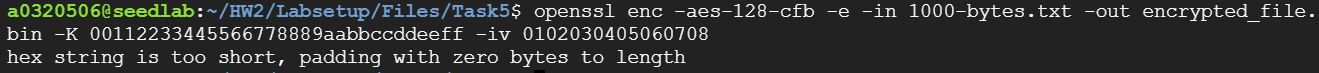
cat decrypted\_file\_bin

cat 1000-bytes.txt



**aes-128-cfb**

openssl enc -aes-128-cfb -e -in 1000-bytes.txt -out encrypted\_file.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

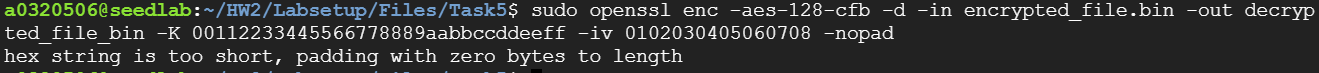


hexedit encrypted\_file.bin



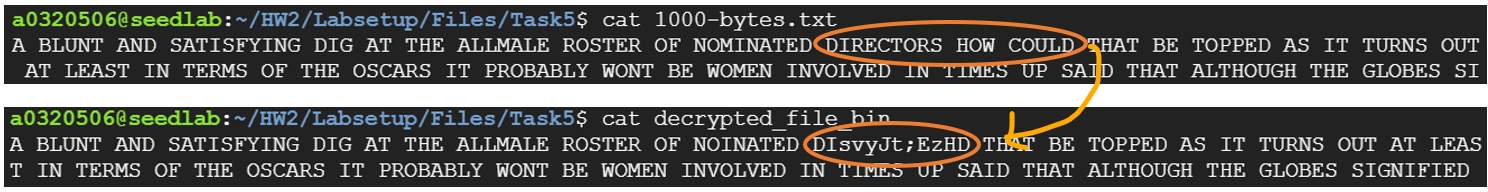


sudo openssl enc -aes-128-cfb -d -in encrypted\_file.bin -out decrypted\_file\_bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad



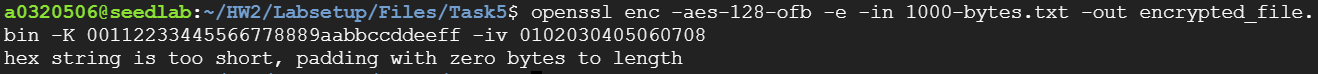
cat decrypted\_file\_bin

cat 1000-bytes.txt



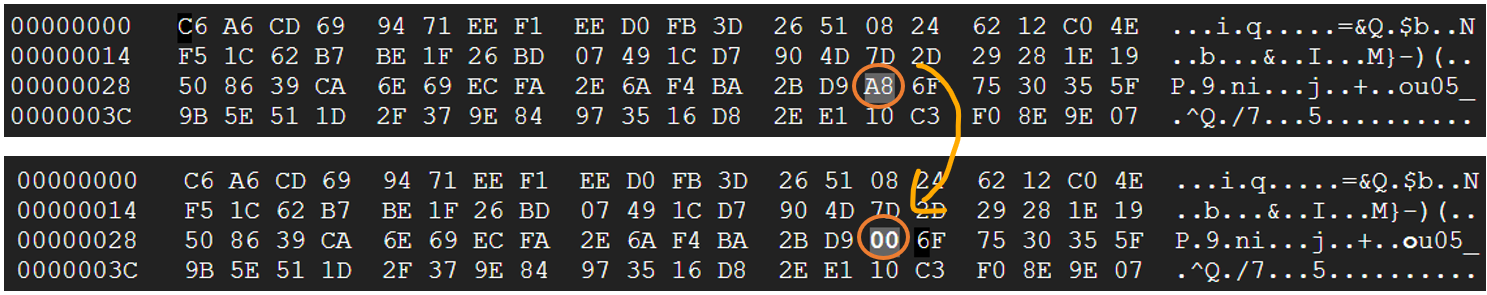
**aes-128-ofb**

openssl enc -aes-128-ofb -e -in 1000-bytes.txt -out encrypted\_file.bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708

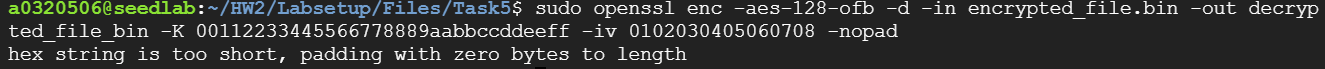


hexedit encrypted\_file.bin



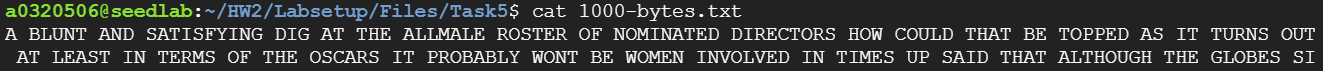


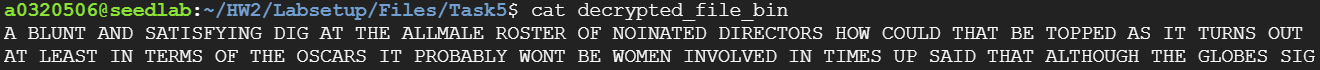
sudo openssl enc -aes-128-ofb -d -in encrypted\_file.bin -out decrypted\_file\_bin -K 00112233445566778889aabbccddeeff -iv 0102030405060708 -nopad



cat decrypted\_file\_bin

cat 1000-bytes.txt





不受影響

**實驗前猜測:** ECB和CBC會受較大影響，因為他們是用區塊為單位做加密，CFB和OFB影響則較小，因為他們是用byte為單位做加密

**實驗後觀察:**

ECB和CBC被影響到的部分: OF NOMINATED DIRECTORS，理論上ECB只影響到單個塊的解密結果，CBC會影響到錯誤塊他的下一塊的解密結果，但在這裡剛好相同

CFB被影響到的部分: DIRECTORS HOW COULD，影響到錯誤位元及其後續位元的解密結果

因為ECB、CBC以塊為單位，CFB以byte為單位，所以他們受影響的起始位置不同

OFB不受影響，因為他的加密過程中不需要使用到明文，因此即使在加密過程中某一位元發生了錯誤，也不會對解密結果產生影響



hexdump P.txt

openssl enc -aes-128-ofb -e -in P.txt -out C.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

hexdump C.txt

openssl enc -aes-128-ofb -e -in P.txt -out C.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

hexdump C.txt

IV相同，都是0102030405060708



加密結果完全相同

hexdump P.txt

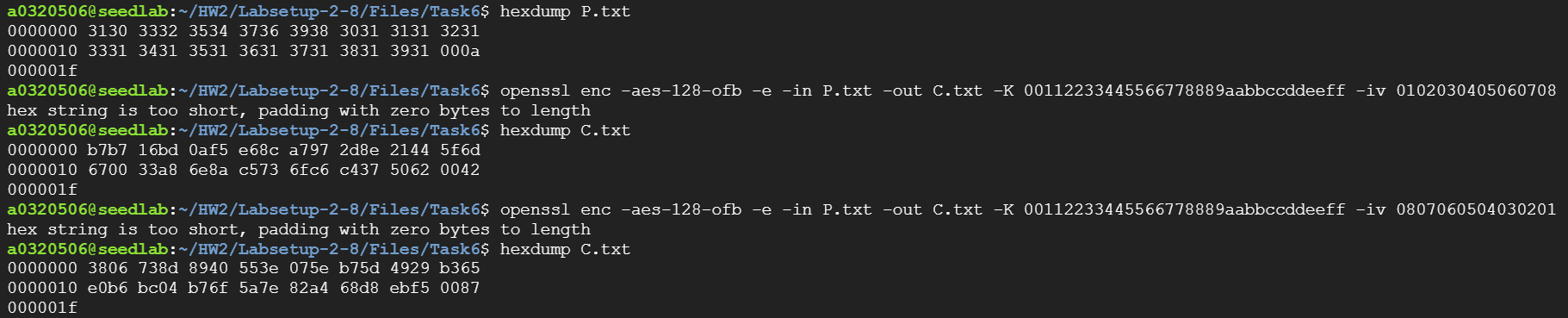
openssl enc -aes-128-ofb -e -in P.txt -out C.txt -K 00112233445566778889aabbccddeeff -iv 0102030405060708

hexdump C.txt

openssl enc -aes-128-ofb -e -in P.txt -out C.txt -K 00112233445566778889aabbccddeeff -iv 0807060504030201

hexdump C.txt

IV不同，0102030405060708和0807060504030201

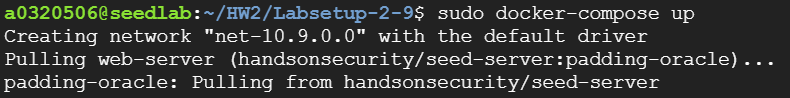


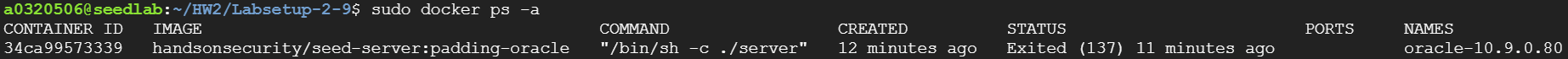
加密結果不同

 不然可能會發生加密結果一樣的情況

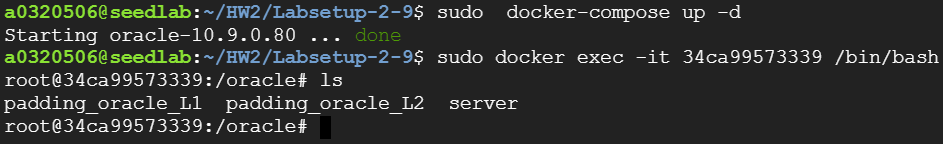


要在docker下執行





ID: 34ca99573339



sudo docker-compose up -d

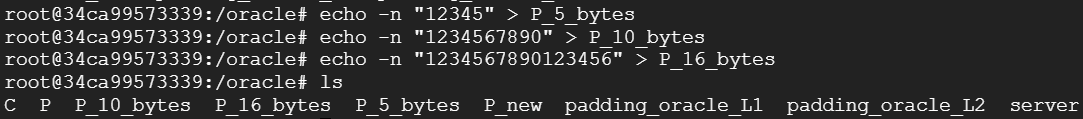
docker exec -it 34ca99573339 /bin/bash



echo -n "12345" > P\_5\_bytes

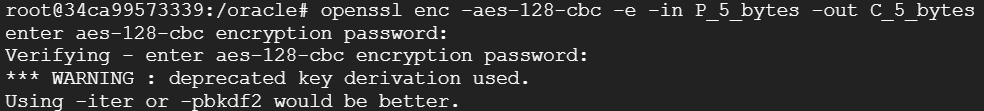
echo -n "1234567890" > P\_10\_bytes

echo -n "1234567890123456" > P\_16\_bytes



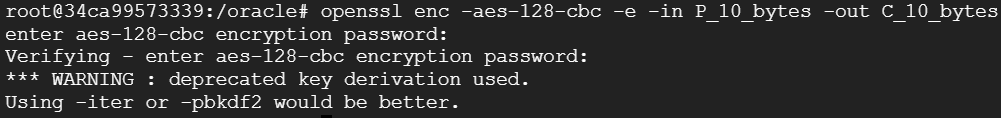
openssl enc -aes-128-cbc -e -in P\_5\_bytes -out C\_5\_bytes

密碼: 00112233445566778889aabbccddeeff

****

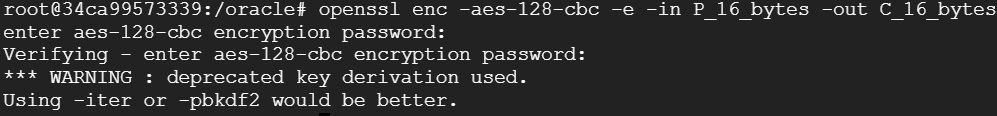
openssl enc -aes-128-cbc -e -in P\_10\_bytes -out C\_10\_bytes

密碼: 00112233445566778889aabbccddeeff

****

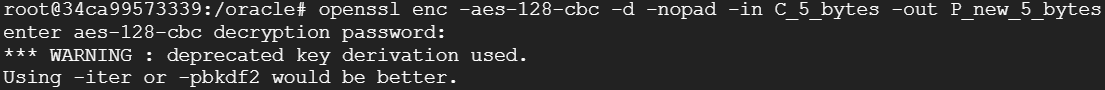
openssl enc -aes-128-cbc -e -in P\_16\_bytes -out C\_16\_bytes

密碼: 00112233445566778889aabbccddeeff

****

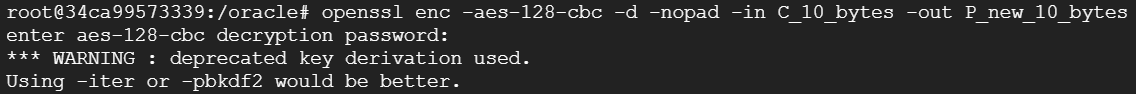
openssl enc -aes-128-cbc -d -nopad -in C\_5\_bytes -out P\_new\_5\_bytes

密碼: 00112233445566778889aabbccddeeff

****

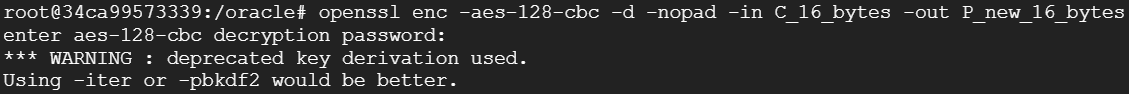
openssl enc -aes-128-cbc -d -nopad -in C\_10\_bytes -out P\_new\_10\_bytes

密碼: 00112233445566778889aabbccddeeff

****

openssl enc -aes-128-cbc -d -nopad -in C\_16\_bytes -out P\_new\_16\_bytes

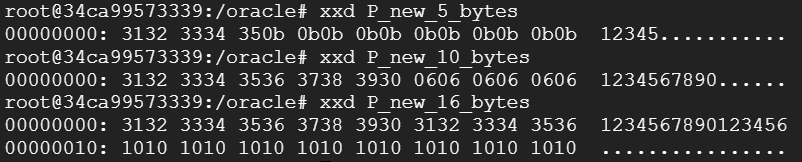
密碼: 00112233445566778889aabbccddeeff

****

xxd P\_new\_5\_bytes

xxd P\_new\_10\_bytes

xxd P\_new\_16\_bytes





用AES-128-CBC模式對一個16 bytes的文件加密時，即使原始文件的大小已經是密鑰塊大小的倍數(在這裡是16 bytes)，仍會看到一個完整的填充塊

這是因為在CBC模式下，加密演算法要求明文的大小必須是密鑰塊大小的倍數(即使原始文件已經是密鑰塊大小的倍數也一樣)

填充是為了確保明文能夠完全填滿一個密鑰塊，避免在加密過程中出現明文被分塊的情況，在CBC 模式下一定要填充，因為每個明文塊都需要與前一個密文塊進行XOR，這需要每個明文塊都有一個完整的密鑰塊大小

所以就算原始文件的大小已經是密鑰塊大小的倍數，還是需要填充來確保加密操作的正確



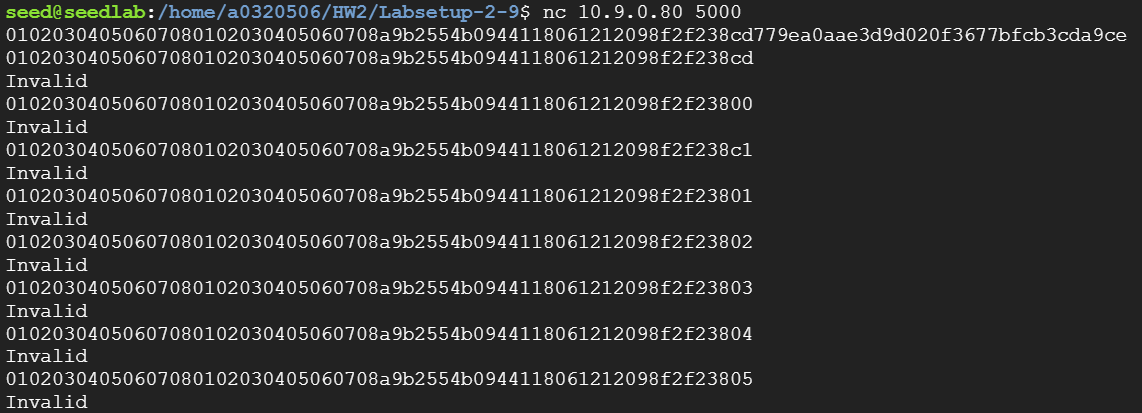




01020304050607080102030405060708a9b2554b0944118061212098f2f238cd779ea0aae3d9d020f3677bfcb3cda9ce

IV: 0102030405 0607080102 0304050607 08

C: a9b2554b09 4411806121 2098f2f238 cd779ea0aa e3d9d020f3 677bfcb3cd a9 ce



理論上往下找就會找到正確的，但我目前試不出來



K = 1

