

Assignment02

60847013S 資工碩一 蘇冠中

1. Modular Multiplicative Inverse

1.1 $135 \bmod 61 = 47$

$$\gcd(135, 61) = \gcd(61, 13) = \gcd(13, 9) = \gcd(9, 4) = \gcd(4, 1) =$$

$$\gcd(1, 0) = 1 \quad \text{有一解 從 } ax + by = 1 \text{ 代回去求 } x, y$$

$$\gcd(1, 0) ; \quad x=1; y=0$$

$$\gcd(4, 1) ; \quad x=0; y=1$$

$$\gcd(9, 4) ; \quad x=1, y=-2$$

$$\gcd(13, 9) ; \quad x=-2, y=3$$

$$\gcd(61, 13) ; \quad x=3, y=-14$$

$$\gcd(135, 61) ; \quad x=-14, y=31$$

最後 $x = -14$ 為其中一個模反元素，而可能的模反元素為

$-14 + (61 \cdot n)$ ， n 為整數，我們要求最小的正整數解，則為

$$-14 + 61 = 47$$

1.2 $7465 \bmod 2464 = 2329$

$$\gcd(7465, 2464) = \gcd(2464, 73) = \gcd(73, 55) = \gcd(55, 18) =$$

$$\gcd(18, 1) = \gcd(1, 0) = 1, \text{ 有一解}$$

從 $ax + by = 1$ 代回去求 x, y

$$\gcd(1, 0) ; x=1, y=0$$

$$\gcd(18, 1) ; x=0, y=1$$

$$\gcd(55, 18) ; x=1, y=-3$$

$$\gcd(73, 55) ; x=-3, y=4$$

$$\gcd(2464, 73) ; x=4, y=-135$$

$$\gcd(7465, 2464) ; x=-135, y=409$$

最後 $x=-135$ 為其中一個模反元素，而可能的模反元素為

$-135 + (2464*n)$ ， n 為整數，我們要求最小的正整數解，

則為 $-135+2464=2329$

1.3 $42828 \bmod 6407 = \text{no}$

$$\gcd(42828, 6407)=\gcd(6407, 4386)=\gcd(4386, 2021)=$$

$$\gcd(2021, 344)=\gcd(344, 301)=\gcd(301, 43)=\gcd(43, 0)=43$$

因此沒有解

2. Fermat' s Theorem

2.1 $4^{255} \bmod 13 =$

$$4^{12(21)+3} \bmod 13 = 4^3 \bmod 13 = 64 \bmod 13 = 12$$

2.2 $7^{1013} \bmod 93 =$

$$7^{92(11)+1} \bmod 93 = 7^1 \bmod 93 = 7$$

3. Chinese Remainder Theorem

3.1

(1) 設 $P = p_1 * p_2 * p_3 * \dots * p_k$, $P_i = P/p_i$, $\forall i \in \{1, 2, \dots, k\}$, 也就是說 P_i 是除了 p_i 以外的 $k-1$ 個整數的乘積

(2) 設 $t_i = P_i^{-1}$ 為 P_i 模 p_i 的數論倒數: $t_i P_i \equiv 1 \pmod{p_i}$, $\forall i \in \{1, 2, \dots, k\}$

(3) n 的通解為 $n = n_1 t_1 P_1 + n_2 t_2 P_2 + n_3 t_3 P_3 + \dots = \sum_{i=1}^k n_i t_i P_i$

(4) 從 $p_1 \sim p_k$ 都互質上可知, 當 $i \in \{1, 2, \dots, k\}$ 、 $j \in \{1, 2, \dots, k\}$, $j \neq i$ 時, $\gcd(p_i, p_j) = 1$, 所以 $\gcd(p_i, P_i) = 1$, 說明存在整數 t_i

使得 $t_i P_i \equiv 1 \pmod{p_i}$, 因此 $n_i t_i P_i \equiv n_i * 1 = n_i \pmod{p_i}$

, 且 $n_j t_j P_j \equiv 0 \pmod{p_i}$

(5) 所以 $n = \sum_{i=1}^k n_i t_i P_i$ 滿足:

$n = \sum_{j \neq i}^k n_j t_j P_j + n_i t_i P_i \equiv n_i + \sum_{j \neq i} 0 = n_i \pmod{p_i}$, 可說明

n 就是此方程組的一個解。

3.2

當 $p_1 \sim p_k$ 之間不互質的時候, 從上面(4)可知, $\gcd(p_i, p_j)$ 就不

會=1, 且 $n_j t_j P_j \pmod{p_i}$ 也不會=0, 範例:

假設 n_1, n_2 兩個方程組,

$n \equiv 3 \pmod{5}$ 和 $n \equiv 4 \pmod{10}$

$M = 5*10=50$, $M_1=10$, $t_1=0$, 當 $t=0$ 時, n_1 方程組算出來的會是

0，也就找不出 n 來。

4. Complement

4.1

從 Fiestel network 可知 $R_i = L_{i-1} \oplus F(R_{i-1}, K_i)$ ，所以可

以得到 $\bar{R}_i = \overline{L_{i-1}} \oplus F(\overline{R_{i-1}}, \bar{K}_i)$ 。

因為排列不會影響到 bit 的值，加上 $E \oplus K_i = \bar{E} \oplus \bar{K}_i$ ，所以

$$F(R_{i-1}, K_i) = F(\overline{R_{i-1}}, \bar{K}_i)。$$

所以 S-BOX 的輸出保持不變、加密結果也保持不變。

4.2

假設有個明文攻擊，若選擇的明文為 M 和 \bar{M} ，攻擊者得到

$$C_1 = E_k(M), C_2 = E_k(\bar{M})，並不會對暴力破解造成太多的影響。$$

5. Polynomial Ring

5.1 $(x^3 + 1)$ and $(x^2 + x + 1)$ with Z_2

$$(x^3 + 1) = (x^2 + x + 1) * (x - 1)$$

$$(x^2 + x + 1) = \mathbf{1} * (x^2 + x + 1)$$

$$\text{得 gcd} = x^2 + x + 1$$

5.2 $(x^3 + x + 1)$ and $(x^2 + 1)$ with Z_3

$$(x^3 + x + 1) = (x^2 + 1) * (x) + \mathbf{1} * 1$$

$$(x^2 + 1) = \mathbf{1} * (x^2 + 1)$$

$$\text{得 gcd} = 1$$

5.3 $(x^4 + 8x^3 + 7x + 8)$ and $(2x^3 + 9x^2 + 10x + 1)$ with Z_{11}

$$(x^4 + 8x^3 + 7x + 8) = (2x^3 + 9x^2 + 10x + 1) + (6x + 10) + (4x^2 + 9)$$

$$(2x^3 + 9x^2 + 10x + 1) = (4x^2 + 9)(6x + 5)$$

$$\text{得 gcd}=(4x^2 + 9)$$

6. Padding Oracle Attack

明文: My power flurries through the air into the ground.

My soul is spiraling in frozen fractals all around. And

one thought crystallizes like an icy blast. I'm never

going back, the past is in the past.

步驟 1: 先在網站上輸入正確的密文，網站會回傳 valid 正

確的回應，若傳送的字串長度不對，則是回傳 len invalid

將 IV 改為全部都為 0，而最後一個是我要嘗試的文字，如下

IV= 000000000000000000000000000000a1 （第一組改 0）

C = 462f0aec8f910e6b5daf6c47947de80c （第二組）

在此情況下，伺服器解密的文字會因填空的漏洞，解出來的明文

最後一段會是 0x01，因此我將 IV 最後一個字不斷改變從

00, 01, 02...ff 全部嘗試，而 IV 最後一個為 8b 的時候，伺服器

回傳了 valid，因此原始的值就為 $8b \text{ xor } 01 = 8a$

再來，解倒數第二個文字，密文填空情形，倒數兩個都會為

0x02，而我們已經知道最後一個為 8a，因此將 $8a \text{ xor } 02 = 88$ ，

將 IV 最後一位改成 88，倒數第二個一樣從 00, 01...ff 全部嘗

試，在 a6 的時候伺服器回 valid，則原始倒數第二個為 $a6 \text{ xor }$

$02 = a4$ ，依此反覆解，將原始值的每個位置都解出來，最後我

先得到第一組原始值 middle，將此 middle 與密文的第一組進行

xor 後可得到 My power flurrie

步驟 3:

```
client.py x oracle.py x attack.py x
67 tryasc = ["01","02","03","04","05","06","07",
68           "08","09","0a","0b","0c","0d","0e","0f","10"]
69
70 middle = []
71 # for j in range(len(IV)):
72 #     for i in asctable:
73 #         cipher = IV[j]+i+midstr(middle,tryasc[j])+text[2]
74 #         if(get(cipher)=="valid"):
75 #             print(i)
76 #             m = xor(i,tryasc[j])
77 #             middle.append(m)
78 #             break
79
80 for t in range(1,len(text),1):
81     middle = []
82     for j in range(len(IV)):
83         for i in asctable:
84             cipher = IV[j]+i+midstr(middle,tryasc[j])+text[t]
85             if(get(cipher)=="valid"):
86                 # print(i)
87                 m = xor(i,tryasc[j])
88                 middle.append(m)
89                 break
90     print(middle)
91
92     print(breakcipher(middle,text[t-1]))
93
[ '8a', 'a4', 'd9', 'fb', '12', '29', '45', '21', '9d', 'a8', 'dc', 'e6', '17', '65', '5a', '4c' ]
My power flurrie
```

同樣的，因為 aes cbc 的加密方式，第一組密文是第二組的 IV

，第二組加密過的則是第三組的 IV，所以可分為說第一組密文

塊用 1, 2 塊來解密，第二組密文快用 2, 3 塊來解，依此類推，

將每一組密文重複步驟 2 可依序解出所有的明文

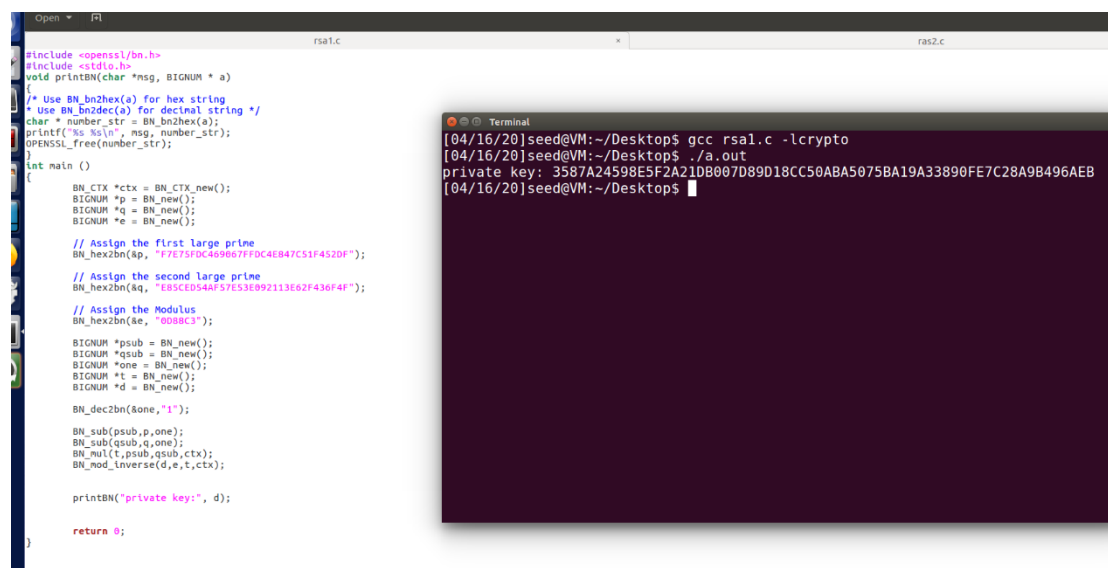
```
87         m = xor(i,tryasc[j])
88         middle.append(m)
89         break
90     print(middle)
91
92     print(breakcipher(middle,text[t-1]))
93
[ '8a', 'a4', 'd9', 'fb', '12', '29', '45', '21', '9d', 'a8', 'dc', 'e6', '17', '65', '5a', '4c' ]
My power flurrie
[ '65', '89', '5d', 'f1', '2f', '18', '8f', '35', '0c', '7b', 'fe', 'fd', '84', '7e', '0f', '35' ]
s through the ai
[ 'b0', '70', '03', 'a2', 'a1', 'b8', 'e7', '67', '0c', '78', 'f1', '8e', 'f9', 'e2', '6a', 'e7' ]
r into the groun
[ '63', '3c', '2a', '45', '6f', '83', 'e7', '09', 'a0', '70', '9c', 'd9', '9c', '94', '96', '6e' ]
d. My soul is sp
[ 'dc', '10', '33', '67', '2f', '73', '22', '33', 'a8', '53', '71', '6b', '8d', 'a0', '29', '1b' ]
iraling in froze
```


而最後一段解出來的填控則是 0b，也的確與密文填控規則一樣，因此明文及解出(跑最後一段伺服器好像快炸掉了，最後跑了 20 多分鐘)。

```
91     print(breakcipher(middle,text[t-1]))
92
93
['dc', '10', '33', '67', '2f', '73', '22', '33', 'a8', '53', '71', '6b', '8d', 'a0', '29', '1b']
iraling in froze
['f4', 'bb', 'dc', '82', '4b', '47', '23', '9d', 'a2', 'af', '64', '6c', '63', '44', 'ba', '72']
n fractals all a
['fc', 'd5', 'f4', '16', '17', 'e7', '77', '2d', '25', '24', 'b5', 'ef', '84', '82', 'cd', 'c8']
ound. And one t
['f1', '1d', '51', 'a7', 'fc', 'd7', '99', '43', '25', 'f8', '0c', 'fd', '2e', '13', '2c', '23']
hought crystalli
['ff', '93', '17', 'af', '6b', 'c8', '2c', 'b6', 'b3', '8d', '15', '1f', 'd8', 'a3', '80', 'c7']
zes like an icy
['50', '03', '5f', '2e', '56', '15', '85', '38', '31', '6d', '49', 'ca', '68', '68', '3a', 'f2']
blast. I'm never
['40', '73', 'bc', 'f5', '0a', '31', '7b', '67', '1d', '3c', '0a', 'f2', 'be', '14', '5d', '44']
going back, the
['ae', '6b', 'f2', '28', '9c', 'b1', '0c', '48', '8c', '10', '4f', '7f', 'f6', 'e9', '4d', 'a7']
past is in the
['e6', '84', 'ed', 'eb', '1a', 'f7', '0a', 'a3', '5f', 'f5', 'f3', 'f4', '52', 'cc', '61', 'b0']
past.<0x0b><0x0b><0x0b><0x0b><0x0b><0x0b><0x0b><0x0b><0x0b><0x0b>
Finished in 1317.2s]
```

7. Lab : RSA

task1



```
#include <openssl/bn.h>
#include <stdio.h>
void printBN(char *msg, BIGNUM * a)
{
    /* Use BN_bn2hex(a) for hex string
    * Use BN_dec2bn(a) for decimal string */
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

int main ()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *e = BN_new();

    // Assign the first large prime
    BN_hex2bn(&p, "F7E75FDC409067FFDC4E847C51F452DF");

    // Assign the second large prime
    BN_hex2bn(&q, "E85CE054AF5F53E092113E62F436F4F");

    // Assign the Modulus
    BN_hex2bn(&e, "0D88C3");

    BIGNUM *psub = BN_new();
    BIGNUM *qsub = BN_new();
    BIGNUM *one = BN_new();
    BIGNUM *t = BN_new();
    BIGNUM *d = BN_new();

    BN_dec2bn(&one, "1");

    BN_sub(psub, p, one);
    BN_sub(qsub, q, one);
    BN_mul(t, psub, qsub, ctx);
    BN_mod_inverse(d, e, t, ctx);

    printBN("private key:", d);

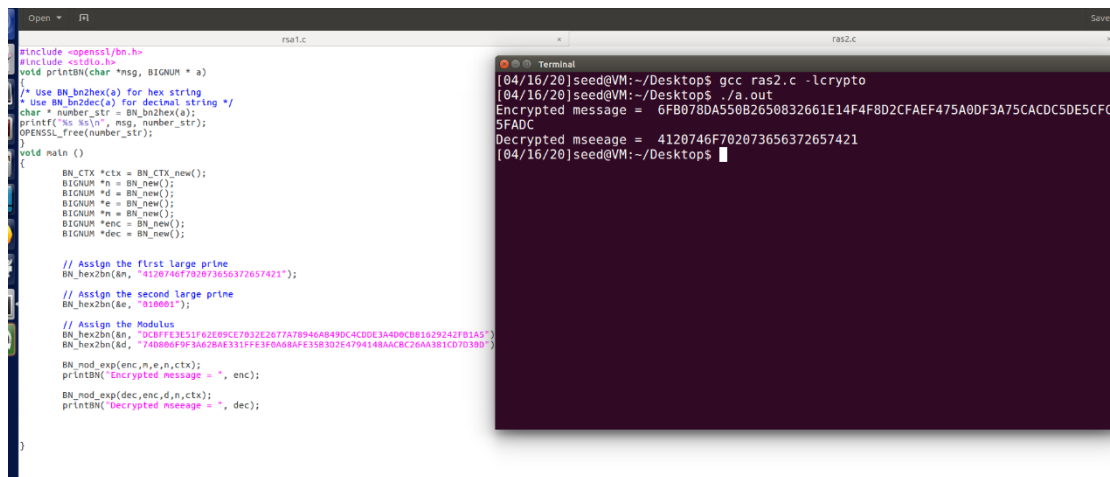
    return 0;
}
```

```
[04/16/20]seed@VM:~/Desktop$ gcc rsal.c -lcrypto
[04/16/20]seed@VM:~/Desktop$ ./a.out
private key: 3587A24598E5F2A210B007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[04/16/20]seed@VM:~/Desktop$
```

使用 BIGNUM、BN 來宣告 p, q, e，再用 BN_sub 將 p、q 做減一的動作，之後 BN_mul 算(p-1)*(q-1)，將此數值與 0D88c3 做

mod_inverse 算出 private key。

task2



The screenshot shows two windows. The left window is a code editor with the following C code:

```
#include <openssl/bn.h>
#include <stdio.h>
void printBN(char *msg, BIGNUM * a)
{
    /* Use BN_bn2hex(a) for hex string
    * use BN_bn2dec(a) for decimal string */
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

void main ()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    // Assign the first large prime
    BN_hex2bn(&n, "1120746F702073656372657421");
    // Assign the second large prime
    BN_hex2bn(&d, "010001");

    // Assign the Modulus
    BN_hex2bn(&n, "DC8FFE3E31F62E99CE7032E2677A78946A849DC4C0DE3A409C8B1629242FB1A5");
    BN_hex2bn(&d, "740B06F9F3A62BAE331FFE3F0A68AFE35B302E4794148AACBC26AA361CD7D300");

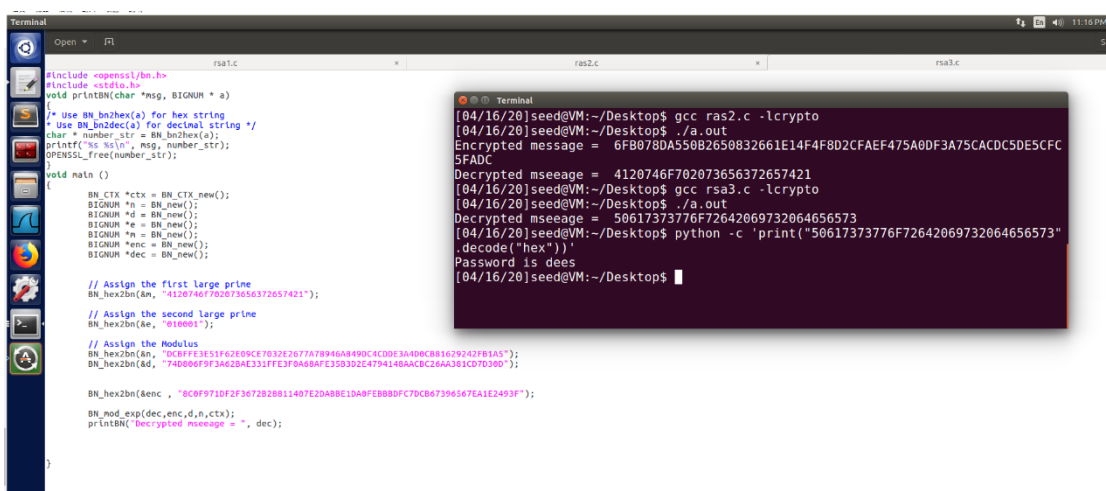
    BN_mod_exp(enc,n,e,n,ctx);
    printf("Encrypted message = ", enc);
    BN_mod_exp(dec,enc,d,n,ctx);
    printf("Decrypted message = ", dec);
}
```

The right window is a terminal showing the execution of the program:

```
[04/16/20]seed@VM:~/Desktop$ gcc ras2.c -lcrypto
[04/16/20]seed@VM:~/Desktop$ ./a.out
Encrypted message = 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
Decrypted message = 4120746F702073656372657421
[04/16/20]seed@VM:~/Desktop$
```

將 message，先轉為 hex 後，使用提示給的 e、n 去做加密，得到加密訊息 enc；之後再將 enc 去使用 private key d 作解密，驗證出加解密是正確的。

task3



The screenshot shows two windows. The left window is a code editor with the following C code:

```
#include <openssl/bn.h>
#include <stdio.h>
void printBN(char *msg, BIGNUM * a)
{
    /* Use BN_bn2hex(a) for hex string
    * use BN_bn2dec(a) for decimal string */
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

void main ()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    // Assign the first large prime
    BN_hex2bn(&n, "4120746F702073656372657421");
    // Assign the second large prime
    BN_hex2bn(&d, "010001");

    // Assign the Modulus
    BN_hex2bn(&n, "DC8FFE3E31F62E99CE7032E2677A78946A849DC4C0DE3A409C8B1629242FB1A5");
    BN_hex2bn(&d, "740B06F9F3A62BAE331FFE3F0A68AFE35B302E4794148AACBC26AA361CD7D300");

    BN_hex2bn(&enc, "0C0F971DF2F3672020811407E20A8BE10A0FEB880FC7DCB67396567EA1E2493F");
    BN_mod_exp(dec,enc,d,n,ctx);
    printf("Decrypted message = ", dec);
}
```

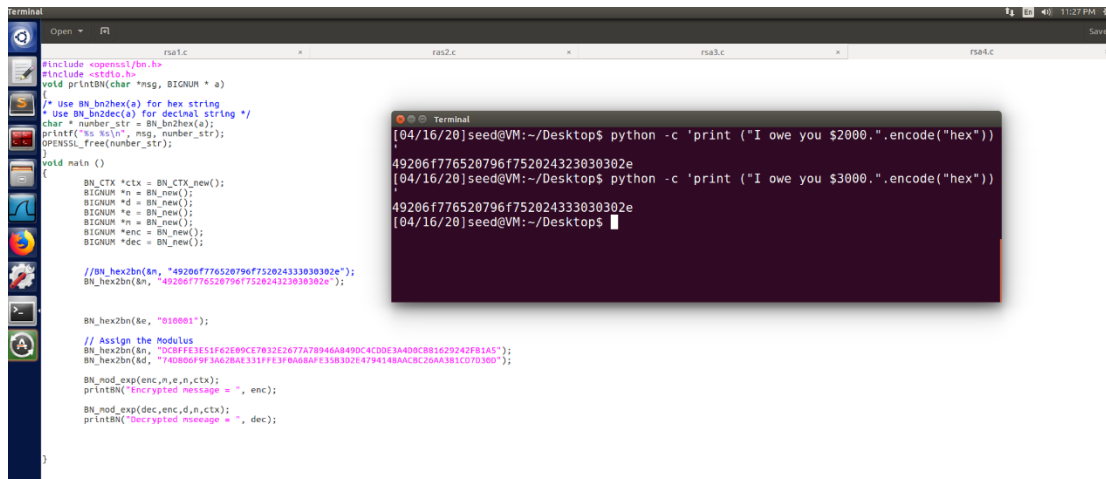
The right window is a terminal showing the execution of the program:

```
[04/16/20]seed@VM:~/Desktop$ gcc ras2.c -lcrypto
[04/16/20]seed@VM:~/Desktop$ ./a.out
Encrypted message = 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
Decrypted message = 4120746F702073656372657421
[04/16/20]seed@VM:~/Desktop$ gcc ras3.c -lcrypto
[04/16/20]seed@VM:~/Desktop$ ./a.out
Decrypted message = 50617373776F72642069732064656573
[04/16/20]seed@VM:~/Desktop$ python -c 'print("50617373776F72642069732064656573".decode("hex"))'
Password is dees
[04/16/20]seed@VM:~/Desktop$
```

使用提示給的密文、以及前一題的 private key、n 去作解密，可得到 50617373……的解密訊息，將此 hex 值轉為字串可得到明

文 Password is dees。

task4



```
#include <openssl/bn.h>
#include <stdio.h>
void printBN(char *msg, BIGNUM * a)
{
    /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

void main ()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    // BN_hex2bn(&n, "49206f776520796f752024333030302e");
    BN_hex2bn(&n, "49206f776520796f752024333030302e");

    BN_hex2bn(&e, "010001");

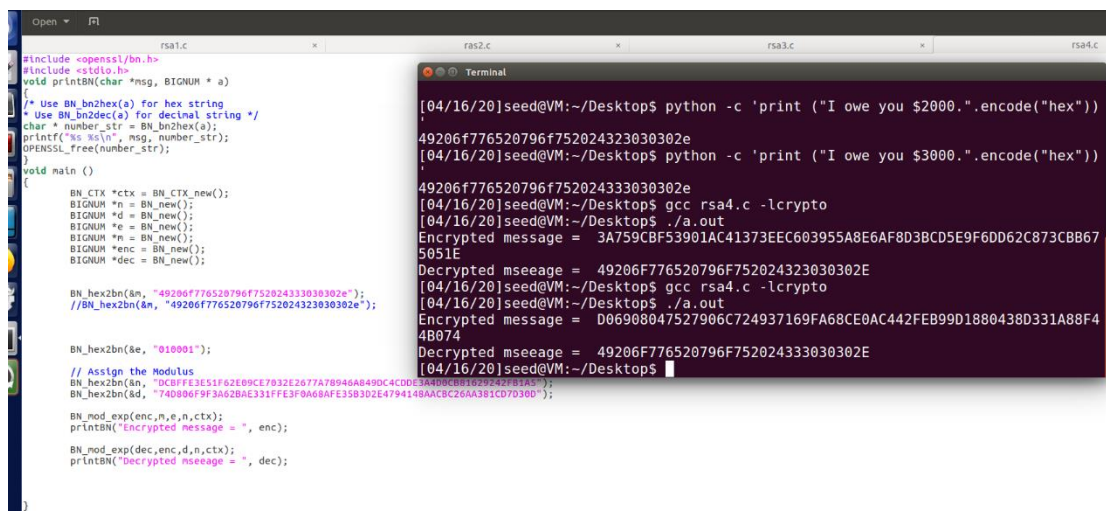
    // Assign the Modulus
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A7B946A849DC4CDD3A40DCB81629242F81A5");
    BN_hex2bn(&d, "740806F9F3A62BAE331FFE3F0A68AFE35B302E479414BAACBC26AA381CD70300");

    BN_mod_exp(enc,n,e,n,ctx);
    printBN("Encrypted message = ", enc);

    BN_mod_exp(dec,enc,d,n,ctx);
    printBN("Decrypted mmessage = ", dec);
}
```

```
[04/16/20]seed@VM:~/Desktop$ python -c 'print ("I owe you $2000.".encode("hex"))'
49206f776520796f752024333030302e
[04/16/20]seed@VM:~/Desktop$ python -c 'print ("I owe you $3000.".encode("hex"))'
49206f776520796f752024333030302e
[04/16/20]seed@VM:~/Desktop$
```

先將 I owe you \$2000. 和 \$3000 轉為 hex 值後可看出 hex 值只差一個值 3 和 2。



```
#include <openssl/bn.h>
#include <stdio.h>
void printBN(char *msg, BIGNUM * a)
{
    /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
    char * number_str = BN_bn2hex(a);
    printf("%s %s\n", msg, number_str);
    OPENSSL_free(number_str);
}

void main ()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    BN_hex2bn(&n, "49206f776520796f752024333030302e");
    // BN_hex2bn(&n, "49206f776520796f752024333030302e");

    BN_hex2bn(&e, "010001");

    // Assign the Modulus
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A7B946A849DC4CDD3A40DCB81629242F81A5");
    BN_hex2bn(&d, "740806F9F3A62BAE331FFE3F0A68AFE35B302E479414BAACBC26AA381CD70300");

    BN_mod_exp(enc,n,e,n,ctx);
    printBN("Encrypted message = ", enc);

    BN_mod_exp(dec,enc,d,n,ctx);
    printBN("Decrypted mmessage = ", dec);
}
```

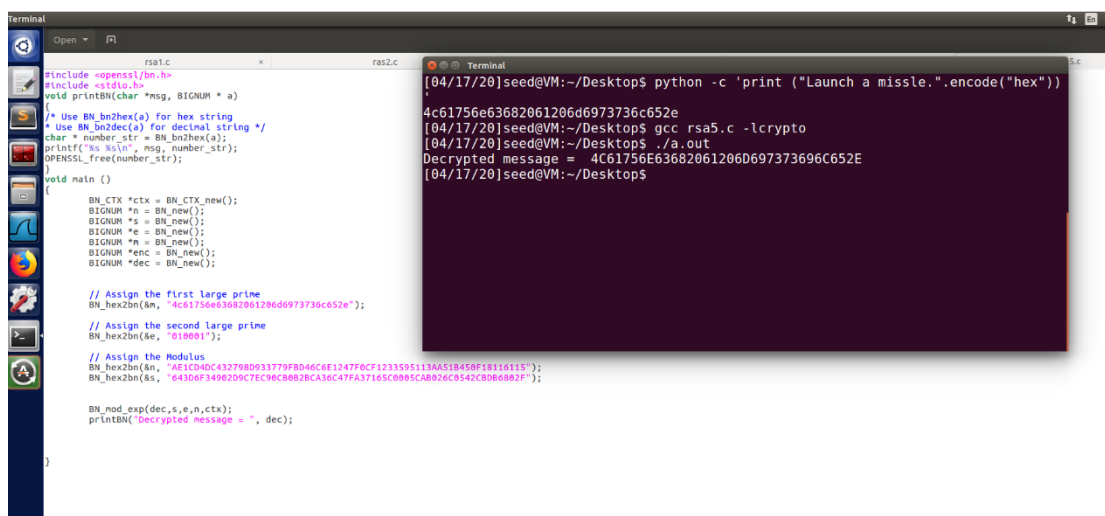
```
[04/16/20]seed@VM:~/Desktop$ python -c 'print ("I owe you $2000.".encode("hex"))'
49206f776520796f752024333030302e
[04/16/20]seed@VM:~/Desktop$ python -c 'print ("I owe you $3000.".encode("hex"))'
49206f776520796f752024333030302e
[04/16/20]seed@VM:~/Desktop$ gcc rsa4.c -lcrypto
[04/16/20]seed@VM:~/Desktop$ ./a.out
Encrypted message = 3A759CBF53901AC41373EEC60395A8E6AF8D3BCD5E9F6DD62C873CBB675051E
Decrypted mseeage = 49206f776520796f752024333030302e
[04/16/20]seed@VM:~/Desktop$ gcc rsa4.c -lcrypto
[04/16/20]seed@VM:~/Desktop$ ./a.out
Encrypted message = D06908047527906C724937169FA68CE0AC442FEB99D1880438D331A88F44B074
Decrypted mseeage = 49206f776520796f752024333030302e
[04/16/20]seed@VM:~/Desktop$
```

使用同樣金鑰做兩次加密可看出，即便只差一個值，加密出來的會完全不一樣。

I owe you \$2000 -> 3A759CBF539.....。

I owe you \$3000 -> D0690804752.....。

Task5



```
rsal.c
#include <openssl/bn.h>
#include <stdio.h>
void printBN(char *msg, BIGNUM *a)
{
    /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
    char *number_str = BN_bn2hex(a);
    printf("%s\n", msg, number_str);
    OPENSSL_free(number_str);
}

void main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *s = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *m = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    // Assign the first large prime
    BN_hex2bn(&n, "4c61756e63682061206d6973736c652e");

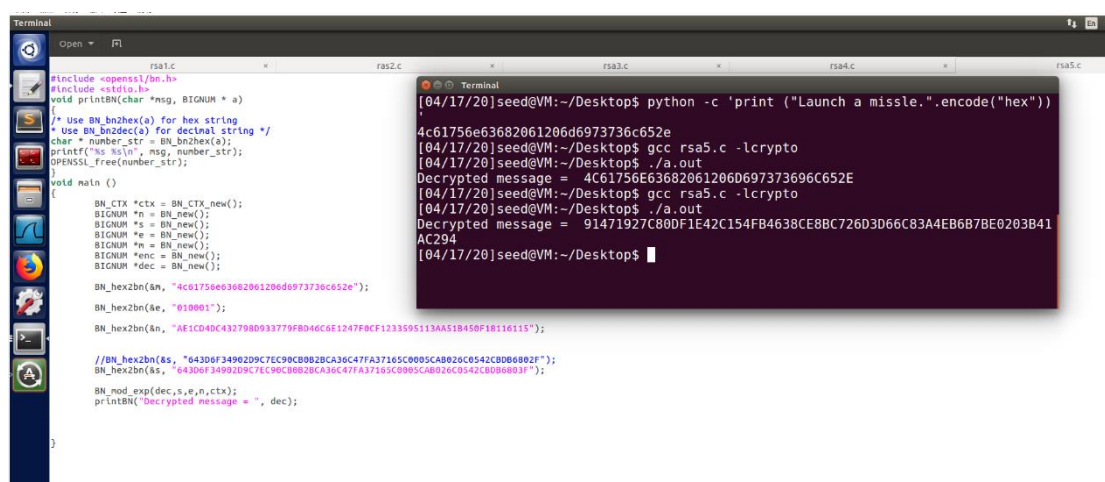
    // Assign the second large prime
    BN_hex2bn(&e, "010001");

    // Assign the Modulus
    BN_hex2bn(&m, "AE1CD40C4327980933779FB046C6E1247F0CF133595113A51B450F18110115");
    BN_hex2bn(&s, "6430F3490209C7EC96C8B82BCA36C47FA37165C0805CA8026C6542CB060802F");

    BN_mod_exp(dec, s, e, n, ctx);
    printBN("Decrypted message = ", dec);
}
```

```
[04/17/20]seed@VM:~/Desktop$ python -c 'print ("Launch a missile.".encode("hex"))'
4c61756e63682061206d6973736c652e
[04/17/20]seed@VM:~/Desktop$ gcc rsa5.c -lcrypto
[04/17/20]seed@VM:~/Desktop$ ./a.out
Decrypted message = 4C61756E63682061206D697373696C652E
[04/17/20]seed@VM:~/Desktop$
```

先將 launch a missile. 轉為 hex 後，將 Alice 的 signature S 以及 e,n 做運算，得到訊息為 4C61756E……，與 launch a missile 的 hex 一樣，代表驗證正確。



```
rsal.c
#include <openssl/bn.h>
#include <stdio.h>
void printBN(char *msg, BIGNUM *a)
{
    /* Use BN_bn2hex(a) for hex string
    * Use BN_bn2dec(a) for decimal string */
    char *number_str = BN_bn2hex(a);
    printf("%s\n", msg, number_str);
    OPENSSL_free(number_str);
}

void main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *s = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *m = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    BN_hex2bn(&n, "4c61756e63682061206d6973736c652e");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&m, "AE1CD40C4327980933779FB046C6E1247F0CF133595113A51B450F18110115");

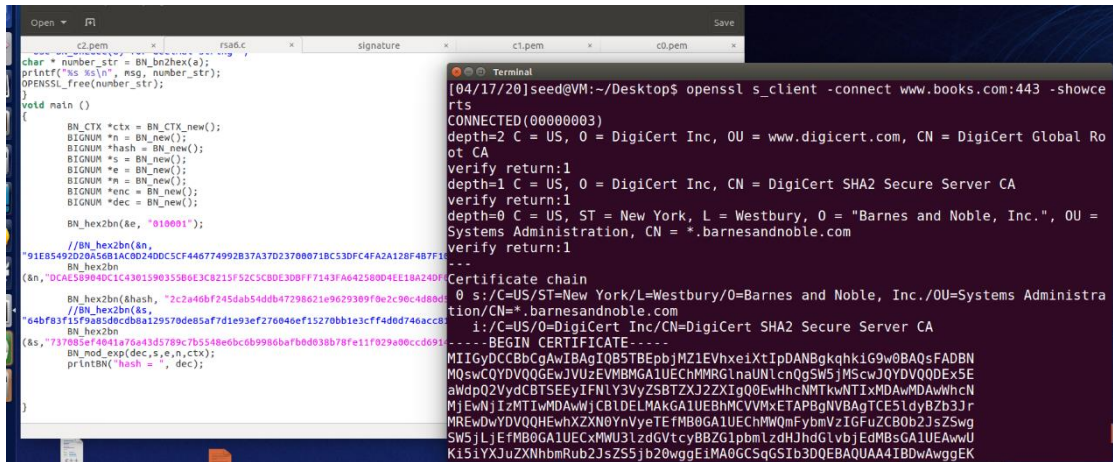
    //BN_hex2bn(&s, "6430F3490209C7EC96C8B82BCA36C47FA37165C0805CA8026C6542CB060802F");
    BN_hex2bn(&s, "6430F3490209C7EC96C8B82BCA36C47FA37165C0805CA8026C6542CB060802F");

    BN_mod_exp(dec, s, e, n, ctx);
    printBN("Decrypted message = ", dec);
}
```

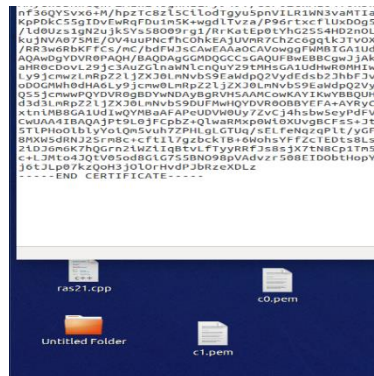
```
[04/17/20]seed@VM:~/Desktop$ python -c 'print ("Launch a missile.".encode("hex"))'
4c61756e63682061206d6973736c652e
[04/17/20]seed@VM:~/Desktop$ gcc rsa5.c -lcrypto
[04/17/20]seed@VM:~/Desktop$ ./a.out
Decrypted message = 4C61756E63682061206D697373696C652E
[04/17/20]seed@VM:~/Desktop$ gcc rsa5.c -lcrypto
[04/17/20]seed@VM:~/Desktop$ ./a.out
Decrypted message = 91471927C80DF1E42C154FB4638CEB8C726D3D66C83A4EB6B7BE0203B41AC294
[04/17/20]seed@VM:~/Desktop$
```

再來修改 S 的最後的值 2F -> 3F，再做一次運算後的結果為 91471927C80D……，與訊息 launch a missile. 的值不一樣；可得知 signature 只差一個值，加解密出來的值也是會完全不一樣。

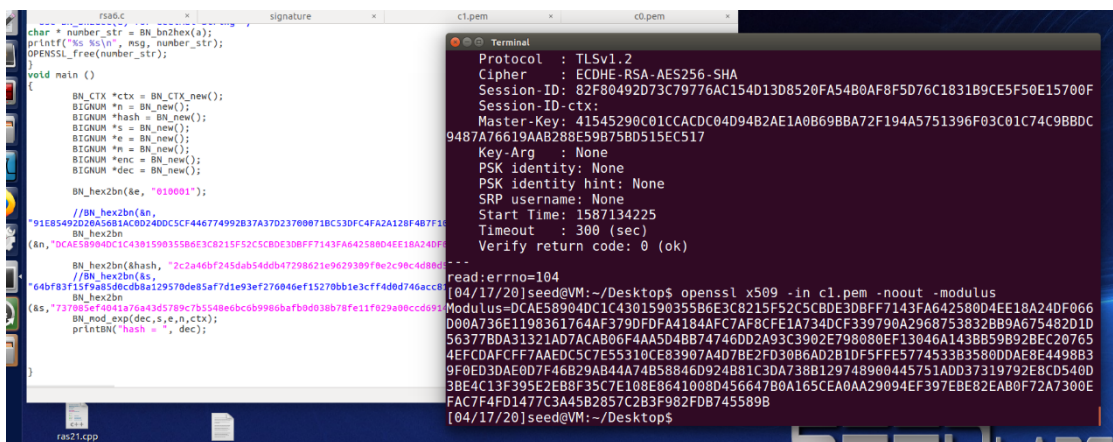
Task6



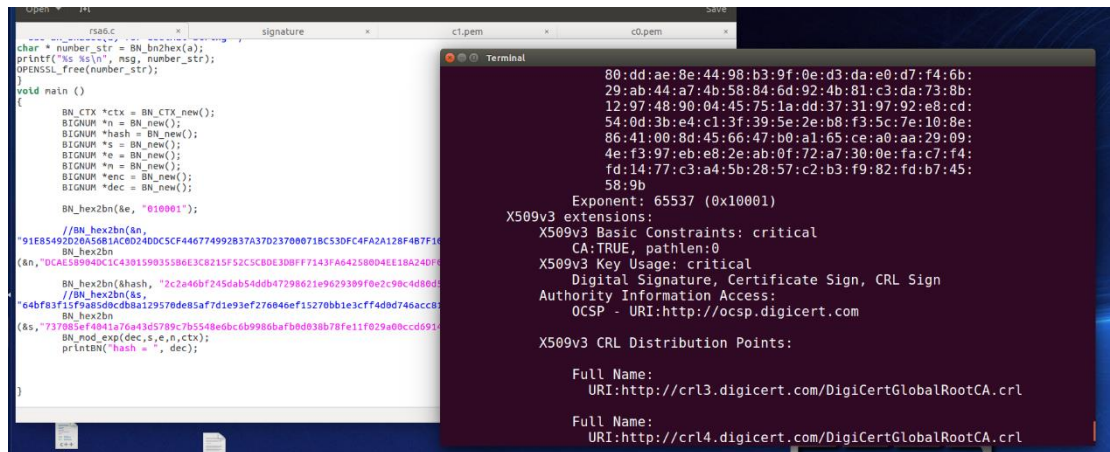
我使用 books.com 來做實驗，而這個網站總共有 2 個 certificate，將 2 個 certificate 存成 c0、c1。



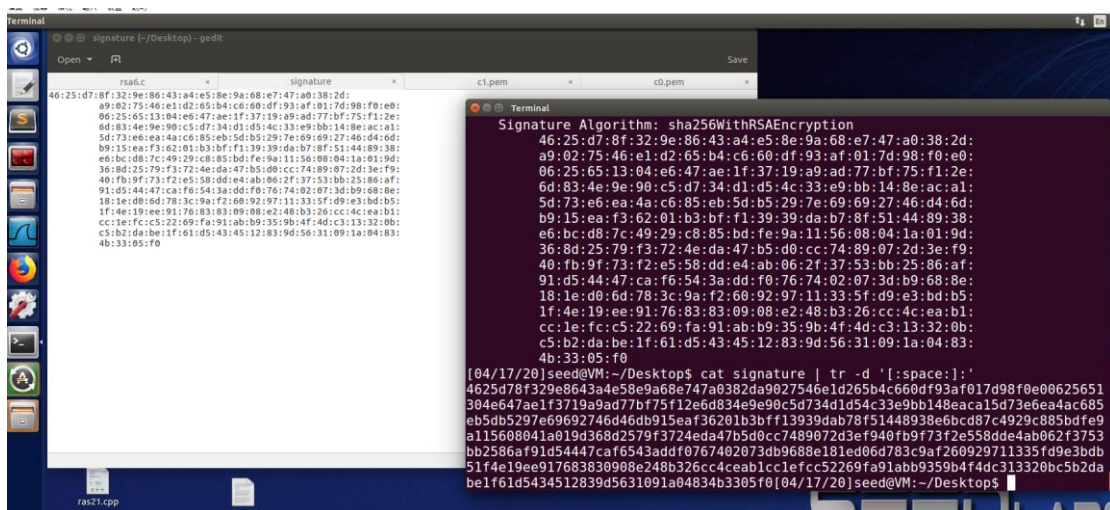
使用 c1 來進行 signature 驗證，先找尋 n，用 openssl x509 -



noout -modulus 對 cl.pem 可直接取得 n。



再來是 e，用 `-text -noout` 可印出 `c1.pem` 的資訊找到 exponent 為 10001。



再來是 signature，從 `c0.pem` 資訊可以找到，存在檔案中後，用 `tr -d '[:space:]:'` 把：和空白去除。

```
Open ▾ ▾ rsa.c x signature x c1.pem x c0.pem x Save
```

```
char * number_str = BN_bn2hex(a);
printf("%s\n", msg, number_str);
OPENSSL_free(number_str);
}

void main ()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *hash = BN_new();
    BIGNUM *s = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *m = BN_new();
    BIGNUM *enc = BN_new();
    BIGNUM *dec = BN_new();

    BN_hex2bn(&n, "010001");

    /BN_hex2bn(&n,
"91E854920A0581AC0240DC5CF446774992837A370237000718C53DFCAFA2A128F4871F",
    BN_hex2bn
(&n, "D0C8E95404C143B9155F863C0215F52C8D03DFB7143FA6A42580A4E18A240F",
    BN_hex2bn(&hash, "2c2a4b7f245dab54dbd4729621e962930f90c2c96c48bd",
    /BN_hex2bn
("64bf83f15f9a85dcdb8a129570de85a7f7d1e93ef276046ef15270b1e3cf4dd476aac8",
    BN_hex2bn
(&s, "40c5e787329e6434de58b9a68677a0382d049275661d265b4c66d9f3afe171d9ff",
    BN_mod_exp(dec, s, e, n, ctx);
    printf("hash = ", dec);
}
```

```
Terminal
181e:40:d6:78:3c:9a:2:f60:92:97:11:33:5f:d9:e3:bd:b5:
1f:4e:19:ee:91:76:83:03:09:08:e2:48:b3:26:cc:4c:ea:b1:
cc:1b:fc:55:22:09:fa:91:ab:99:35:9b:4f:4d:c3:13:32:0b:
c5:b2:da:be:1f:61:d5:43:45:12:83:9d:56:31:09:1a:04:83:
4b:33:05:f0
[04/17/20]seed@VM~/.Desktop$ cat signature | tr -d '\xspace:'!
4625d781329e8643a4e589a68674740382d9a927546e1d265b4c66bd933af17d98f0e0b625651
304e647ae1f7319a94d77bf75f126e6d83a9e90c5d734d1d54c39ebb148eaca15f736ea4ac685
ebb5d52969962746d46db915eaf36201b3bf13939d4b7f514489386ebcd87c492c885bdfef9
a11560804191d04368d2579f3727ada4b75d0c3748907d4b7514489386ebcd87c492c885bdfef9
bb2586a191d54447ca16543ad0767402073d936b88181e06d783c9af260929711335f9e3bd3b
5174e19e517683830998e248b326cc4ceab1c1efcc52269fa91abb9359b4f4dc313320bcb52dda
bf161d54345128394563109a04834ab3305f0[04/17/20]seed@VM~/.Desktop$ openssl asn
lparse -i -in c0.pem
0:d=0 h1=4 l=1736 cons: SEQUENCE
4:d=1 h1=4 l=1456 cons: SEQUENCE
8:d=2 h1=2 l= 3 cons: cont [ 0 ]
10:d=3 h1=2 l= 1 prim: INTEGER :02
13:d=2 h1=2 l= 16 prim: INTEGER :0794c11296c3319044561c5e89784
8A4
31:d=2 h1=2 l= 13 cons: SEQUENCE
33:d=3 h1=2 l= 9 prim: OBJECT :sha256WithRSASignature
44:d=3 h1=2 l= 0 prim: NULL
46:d=2 h1=2 l= 77 cons: SEQUENCE
```

[illegible]

蘇冠中 60847013S，有問題可以 mail 我

Email: deathkabuto@gmail.com