CISC870 Assignment

October 11, 2022

CISC870 Assignment2

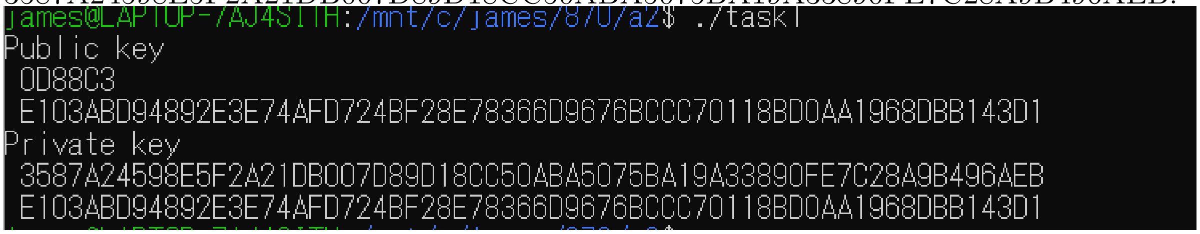
Name: Donghao Wang
Student Number: 20119632

Partner:
Student Number:

Task1

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

int main()
{
    // init
    BN_CTX *ctx = BN_CTX_new();
    //prime number p q
    BIGNUM *p = BN_new();
    BIGNUM *q = BN_new();
    //n=pq
    BIGNUM *n = BN_new();
    //phi(n)=(p-1)(q-1)
    BIGNUM *phi = BN_new();
    BIGNUM *p_decrement = BN_new();
    BIGNUM *q_decrement = BN_new();

    //public key pair(e,n )
    BIGNUM *e = BN_new();
    BIGNUM *d = BN_new();
    BIGNUM *res = BN_new();

    // assign value
    BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e, "0D88C3");

    //n=pq
    BN_mul(n, p, q, ctx);
    //printBN("public key", e, n);
    printf("%s\n %s\n %s\n", "Public key",BN_bn2hex(e), BN_bn2hex(n));

    // phi(n) = (p-1)*(q-1)
    BN_sub(p_decrement, p, BN_value_one());
    BN_sub(q_decrement, q, BN_value_one());
    BN_mul(phi, p_decrement, q_decrement, ctx);

    // e & phi(n) shoud be relatively prime
    BN_gcd(res, phi, e, ctx);
    if (!BN_is_one(res))
    {
        printf("Error: e and phi(n) is not relatively prime \n ");
        exit(0);
    }

    BN_mod_inverse(d, e, phi, ctx);
    printf("%s\n %s\n %s\n", "Private key",BN_bn2hex(d), BN_bn2hex(n));

    //I know I need to free the pointers
    //but in this system the program automatically frees them after executing the program
    //cheers

    return 0;
}

    return 0;
}
```

Step2:calculate $Phi(n) = (p-1)(q-1)$
Step3:make sure public exponent(e)and phi(n) are relatively prime gcd(e,phi(n))=1
Step4 compute privatekey d*e = 1 mod phi(n), in this case the inverse mod method should be used.
Here is the result, the final key is
3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB.



```
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/a2$ ./task1
Public key
 0D88C3
 E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
Private key
 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
 E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
```

Task2

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
int main()
{
    // init
    BN_CTX *ctx = BN_CTX_new();
    //public key
    BIGNUM *n = BN_new();
    //exponent
    BIGNUM *e = BN_new();
    //Plain text
    BIGNUM *P = BN_new();
    //Cipher text
    BIGNUM *C = BN_new();
    //private key
    BIGNUM *d = BN_new();
    //decrypted plaintext
    BIGNUM *P_d = BN_new();
    // initialize
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&P, "4120746f702073656372657421"); //" A top secret!" in hexadecimal derived by python
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    // encrypt
    BN_mod_exp(C, P, e, n, ctx);
    printf("%s\n %s\n", "Cipher text",BN_bn2hex(C));

    //Decrypt
    BN_mod_exp(P_d, C, d, n, ctx);
    //print decrpted message in hex
    printf("%s\n", BN_bn2hex(P_d));
    //Verify if the decrpted plain text is the same as before encryption
    if (BN_cmp(P,P_d)==0)
    {
        printf("%s\n", "Successful decrpytion");
    };

    return 0;
}
```

Encryption= $x^e mod n$
Decryption = $y^d mod n$

```
Cipher text
 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
4120746F702073656372657421
Successful decrpytion
```

The cipher text is
6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC

Task3

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
int main()
{
    // init
    BN_CTX *ctx = BN_CTX_new();
    //public key
    BIGNUM *n = BN_new();
    //Plain text
    BIGNUM *P = BN_new();
    //Cipher text
    BIGNUM *C = BN_new();
    //private key
    BIGNUM *d = BN_new();
    // assign values
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F"); //" A top
secret!" in hexadecimal derived by python
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    //Decrypt:P=C^d mod n
    BN_mod_exp(P, C, d, n, ctx);
    //print decrpted message in hex
    printf("%s\n", BN_bn2hex(P));
    return 0;
}
```

Decryption $= y^d modn$

```
[10/09/22]seed@VM:~$ python -c 'print("50617373776F726420697320646565 73".decode
"hex"))'
Password is dees
```

Plain text is : Password is dees

Task4

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
int main()
{
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM *n = BN_new();
    BIGNUM *d = BN_new();
    //two plain text
    BIGNUM *P_1 = BN_new();
    BIGNUM *P_2 = BN_new();
    //two signature
    BIGNUM *S_1 = BN_new();
    BIGNUM *S_2 = BN_new();

    // assign values
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&P_1, "49206f776520796f75202432303030"); //I owe you $2000
    BN_hex2bn(&P_2, "49206f776520796f75202433303030"); //I owe you $3000
    // encrypt
    BN_mod_exp(S_1, P_1, d, n, ctx);
    BN_mod_exp(S_2, P_2, d, n, ctx);
    //print the signature
    printf("%s\n%s\n", "I owe you $2000",BN_bn2hex(S_1));
    printf("%s\n%s\n", "I owe you $3000",BN_bn2hex(S_2));

    return 0;
}
```

Signature generation: $S = P^d mod n$

d is the private key and n is the public key

```
james@LAPTOP-7AJ4STTH:/mnt/c/james/870/a2$ ./task4
I owe you $2000
80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
I owe you $3000
04FC9C53ED7BBE4ED4BE2C24B0BDF7184B96290B4ED4E3959F58E94B1ECEA2EB
```

Even there is only a slight change in plain text, it would result in totally different signature.

Task5

```
[10/10/22]seed@VM:~$ python -c 'print("Launch a missile.".encode("hex"))'
4c61756e63682061206d697373696c652e
```

Use the signature $S^e mod n$ should generate the exact same plain text value

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
int main()
{
    // ctx
    BN_CTX *ctx = BN_CTX_new();
    //public key
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    //plain text
    BIGNUM *P = BN_new();
    //cipher text
    BIGNUM *C = BN_new();
    BIGNUM *C_broke = BN_new();
    //signature
    BIGNUM *S = BN_new();
    BIGNUM *S_broke = BN_new();

    // assign values
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_dec2bn(&e, "65537");
    BN_hex2bn(&P, "4c61756e63682061206d697373696c652e"); //" Launch a missile."
    BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    BN_hex2bn(&S_broke, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");

    // check P == S^e mod n
    BN_mod_exp(C, S, e, n, ctx);
    BN_mod_exp(C_broke, S_broke, e, n, ctx);
    printf("%s %s\n", "use unbroken signature:",BN_bn2hex(C));

    if (BN_cmp(C, P) == 0)
    {
        printf("Success \n");
    }
    else
    {
        printf("Fail \n");

    }
    printf("%s %s\n", "use broken signature:",BN_bn2hex(C_broke));
    if (BN_cmp(C_broke, P) == 0)
    {
        printf("Success \n");
    }
    else
    {
        printf("Fail \n");
    }

    return 0;
}
```

```
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/a2$ ./task5
use unbroken signature: 4C61756E63682061206D697373696C652E
Success
use broken signature: 91471927C80DF1E42C154FB4638CE8BC726D3D66C83A4EB6B7BE0203B41AC294
Fail
```

Even if there is a small change in the signature value, the verification process would fail.

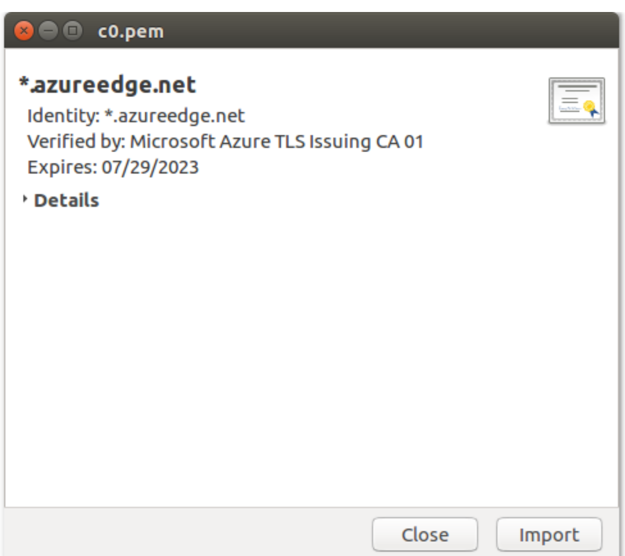
Task6

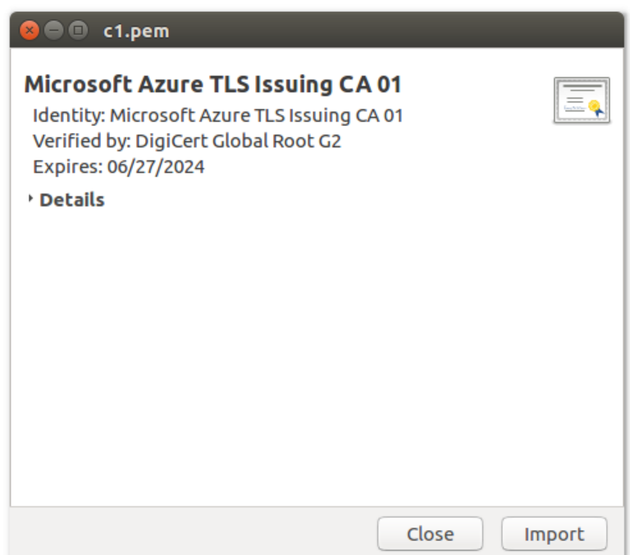
In this case, the certificate of queensu.ca is used

Step1

```
[10/10/22]seed@VM:.../task6$ sudo openssl s_client -connect queensu.ca:443 -show
certs>task6.txt
depth=2 C = US, O = DigiCert Inc, OU = www.digicert.com, CN = DigiCert Global Ro
ot G2
verify return:1
depth=1 C = US, O = Microsoft Corporation, CN = Microsoft Azure TLS Issuing CA 0
1
verify return:1
depth=0 C = US, ST = WA, L = Redmond, O = Microsoft Corporation, CN = *.azureedg
e.net
verify return:1
```

Save the two certificate in the two files c0 and c1





Step2

```
[10/10/22]seed@VM:.../task6$ openssl x509 -in c1.pem -noout -modulus
Modulus=C79D703AE45EE7CFEE9C559B51472C3340F488F01DA328698F1AAEAA99E067DFE31CAB47
28276AF5CBEE0D7630F031F10E8377D85F073ADF5B1905D01AF6C08BE5BE4CD6D359DA3554A41E9F
8E4B784BDD6EA797F8D8BD99D57895E3B6C29871721FFFA5ADB29705E5AC2543D2D925C8F0687ECA
A19B8AF931E95C232DCC3F17355766196FC92340BFED4FC4EDEBD79FD5C38A8F1262419233ADDA2C
EE41B4B0AFD3DDBED5DE06181490628951ABEECF775911124581 3E6D32C19D956FC1884B90CBAE37
A5124067282CB2DC1D3242E77D76956C894784EC5E6C63C31EF7A595C5071B3B261D118FE3FFD529
0E5386FAF03E6203AF06190A1238AACA05693FC119A0C3228DEA61CE0E376F9422F6BE54AEF628A3
5F6847FF297F81C93433C5D3F149C0554C5BC3E60812C6B2D8A7DCD135F8D7964E41ADD13CE91E38
0AC9F6833F6B4EE1B0A4D46EE93F6903153ED261F73CC5B802905436502117B1A1FF7F9626A4B9F8
60F88D14279E22275AEF19DF4EF738EC8C7255C7071EAD45CC6CA03AA7A6AF11044CA87C86F0D1E9
E05DAC1C2608756066B9C0AA79FDDC9B46C8F2E2EBE723586546B4C647C3356F5F51EF4851BBEF5B
2C91C123271890350 0D04561C4877371BCD6FB9059405E755D46492F863A5119C845853033AA6C25
B5D5A158313208C282DB1FE6499BD9B6564663E837ED8EF6A87CE2770B725BBAC17AD649
[10/10/22]seed@VM:.../task6$ openssl x509 -in c1.pem -text -noout| grep Exponent
                Exponent: 65537 (0x10001)
```

Derive the public key (n,e)

Step3

```
  Signature Algorithm: sha384WithRSAEncryption
       b4:e8:78:a7:66:dc:d1:f1:14:14:1b:9a:e9:19:bc:f1:59:b6:
       ce:6a:53:88:ff:70:47:1a:8d:00:4f:77:00:0e:eb:d7:10:d1:
       48:e1:b4:de:32:6f:57:0c:60:e5:0e:9e:63:18:31:94:f1:be:
       0a:03:9f:1d:07:1e:35:7e:a4:74:c4:dc:c2:f8:d8:e2:d0:50:
       56:89:62:40:9b:e3:ba:17:67:00:a4:59:ca:de:8d:70:f1:ed:
       19:83:bc:e0:2a:4d:d2:c0:e7:2f:f5:17:64:db:e6:da:76:43:
       5b:51:94:10:1d:15:e4:f1:e9:c8:cf:07:0e:ef:82:85:bd:ae:
       96:c2:55:29:47:5f:e4:cf:61:3b:9a:87:51:0d:f5:d6:7f:10:
       f8:f6:b7:5b:bf:5e:86:8f:74:b5:a0:2c:1e:b6:74:8b:cb:7b:
       6d:a9:6f:ea:9a:23:71:5b:f4:4c:0a:e5:6f:d4:89:f4:ec:c8:
       76:e1:ae:1b:ad:73:04:1b:79:ea:e7:cc:38:72:1a:0a:e4:ff:
       2c:f5:e0:2e:83:0b:5a:65:16:1d:78:0b:12:41:cb:9a:1f:47:
       de:ee:79:63:ba:ea:96:c0:d7:e2:de:59:89:b7:54:c8:1b:53:
       fe:bb:55:5b:a6:11:c9:8b:50:99:e3:f6:8d:7d:1a:50:54:51:
       a2:ab:f2:98:69:6b:55:01:be:05:09:c0:77:d6:1c:9c:56:56:
       70:b5:6e:a0:e6:3d:4f:b6:16:74:e0:4e:fb:5c:86:28:2e:c3:
       59:6d:06:48:e4:a9:18:cc:83:b0:c7:ac:b4:09:2c:77:34:0b:
       6c:e8:14:5d:ad:61:6e:fa:c6:3b:d0:25:c7:dc:b4:08:c0:59:
       41:b2:5a:c0:23:18:29:a6:bb:3c:a6:53:4b:46:1b:c5:b7:80:
       8c:83:1c:5d:d3:eb:5b:81:93:c8:0a:ea:6c:cf:fa:7c:f8:79:
       ab:3a:d0:2c:23:63:b1:3b:27:7e:5f:a8:36:a8:20:db:df:d1:
       e5:8e:9b:af:fd:4a:34:44:3e:7f:92:22:b6:41:47:25:6c:01:
       68:fe:8c:5c:e7:b8:eb:d7:4e:3c:57:d4:45:71:0a:35:df:23:
       ab:e4:b1:f9:69:a0:60:20:5b:ef:ac:25:b7:91:f7:fd:89:07:
       0b:d4:19:54:83:16:50:58:20:d5:93:5b:77:7d:04:63:2e:e2:
       23:45:18:b9:5e:30:71:89:f9:d5:d9:b5:32:78:bf:81:88:23:
       81:1d:28:50:0c:9e:3c:13:df:e6:f0:79:ed:67:b3:fc:5d:36:
       56:d0:23:d1:ed:61:ae:1d:32:d1:3f:c6:ea:de:0f:0b:c1:c7:
       9f:03:f0:60:8e:f1:28:ca
```

Derive the exponent and modulus

Step4

```
[10/10/22]seed@VM:.../task6$  openssl asn1parse -i -in c0.pem
    0:d=0  hl=4 l=2234 cons: SEQUENCE
    4:d=1  hl=4 l=1698 cons:  SEQUENCE
    8:d=2  hl=2 l=   3 cons:   cont [ 0 ]
   10:d=3  hl=2 l=   1 prim:    INTEGER           :02
   13:d=2  hl=2 l=  19 prim:   INTEGER           :33004FF63A45418DC82D798D350000004FF63A
   34:d=2  hl=2 l=  13 cons:   SEQUENCE
   36:d=3  hl=2 l=   9 prim:    OBJECT            :sha384WithRSAEncryption
   47:d=3  hl=2 l=   0 prim:    NULL
   49:d=2  hl=2 l=  89 cons:   SEQUENCE
   51:d=3  hl=2 l=  11 cons:    SET
   53:d=4  hl=2 l=   9 cons:     SEQUENCE
   55:d=5  hl=2 l=   3 prim:      OBJECT            :countryName
   60:d=5  hl=2 l=   2 prim:      PRINTABLESTRING   :US
   64:d=3  hl=2 l=  30 cons:    SET
   66:d=4  hl=2 l=  28 cons:     SEQUENCE
   68:d=5  hl=2 l=   3 prim:      OBJECT            :organizationName
```

We can see in the first sign ends at offset 4
Thus we get the content starts at offset4

```
[10/10/22]seed@VM:.../task6$  openssl asn1parse -i -in c0.pem -strparse 4 -out c0_body.bin -noout
```

Since the encryption type is sha384rsa, the hash value will be derived through the sha384sum function

```
[10/10/22]seed@VM:.../task6$ sha384sum c0_body.bin
f1b49da16d7e46698607236395b19e1d67fdd647f6ac42a19b86f087172e69724d19fb1492076750ed018d92e670a4fa  c0_body.bin
```

Derived the hash value of the certificate

Step5

```
[10/11/22]seed@VM:.../task6$ python
Python 2.7.12 (default, Nov 19 2016, 06:48:10)
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> prefix="01"
>>> hash=""
KeyboardInterrupt
>>> hash="f1b49da16d7e46698607236395b19e1d67fdd647f6ac42a19b86f087172e69724d19fb1492076750ed018d92e670a4fa"#drop the last digit that doesn't represent anything
>>> hex="3041300D060960864801650304020205000430"
>>> padding=384-8-(len(prefix)-len(hash)-len(hex))//2
>>>
KeyboardInterrupt
>>> prefix+"FF"*padding+"00"+hex+hash
'01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003041300D060960864801650304020205000430f1b49da16d7e46698607236395b19e1d67fdd647f6ac42a19b86f087172e69724d19fb149207
d018d92e670a4fa'
```

Padding the hashed certificate with respect to sha384rsa.

```c
#include <stdio.h>
#include <openssl/bn.h>


int main()
{
    // ctx
    BN_CTX *ctx = BN_CTX_new();
    //public key
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    //certificate content
    BIGNUM *P = BN_new();
    //verify with signature
    BIGNUM *C = BN_new();
    //signature
    BIGNUM *S = BN_new();

    // pulic key(Modulus)
    BN_hex2bn(&n,
"c79d703ae45ee7cfee9c559b51472c3340f488f01da328698f1aaeaa99e067dfe31cab4728276af5cbee0d7630f031f10e83
77d85f073adf5b1905d01af6c08be5be4cd6d359da3554a41e9f8e4b784bdd6ea797f8d8bd99d57895e3b6c29871721fffa5a
db29705e5ac2543d2d925c8f0687ecaa19b8af931e95c232dcc3f17355766196fc92340bfed4fc4edebd79fd5c38a8f126241
9233adda2cee41b4b0afd3ddbed5de06181490628951abeecf7759111245813e6d32c19d956fc1884b90cbae37a5124067282
cb2dc1d3242e77d76956c894784ec5e6c63c31ef7a595c5071b3b261d118fe3ffd5290e5386faf03e6203af06190a1238aaca
05693fc119a0c3228dea61ce0e376f9422f6be54aef628a35f6847ff297f81c93433c5d3f149c0554c5bc3e60812c6b2d8a7d
cd135f8d7964e41add13ce91e380ac9f6833f6b4ee1b0a4d46ee93f6903153ed261f73cc5b802905436502117b1a1ff7f9626
a4b9f860f88d14279e22275aef19df4ef738ec8c7255c7071ead45cc6ca03aa7a6af11044ca87c86f0d1e9e05dac1c2608756
066b9c0aa79fddc9b46c8f2e2ebe723586546b4c647c3356f5f51ef4851bbef5b2c91c1232718903500d04561c4877371bcd6
fb9059405e755d46492f863a5119c845853033aa6c25b5d5a158313208c282db1fe6499bd9b6564663e837ed8ef6a87ce2770
b725bbac17ad649");
    BN_dec2bn(&e, "65537");

BN_hex2bn(&P,"01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF003041300
D060960864801650304020205000430f1b49da16d7e46698607236395b19e1d67fdd647f6ac42a19b86f087172e69724d19fb
1492076750ed018d92e670a4fa");//padded hash
```

```
    BN_hex2bn(&S,
"b4e878a766dcd1f114141b9ae919bcf159b6ce6a5388ff70471a8d004f77000eebd710d148e1b4de326f570c60e50e9e6318
3194f1be0a039f1d071e357ea474c4dcc2f8d8e2d050568962409be3ba176700a459cade8d70f1ed1983bce02a4dd2c0e72ff
51764dbe6da76435b5194101d15e4f1e9c8cf070eef8285bdae96c25529475fe4cf613b9a87510df5d67f10f8f6b75bbf5e86
8f74b5a02c1eb6748bcb7b6da96fea9a23715bf44c0ae56fd489f4ecc876e1ae1bad73041b79eae7cc38721a0ae4ff2cf5e02
e830b5a65161d780b1241cb9a1f47deee7963baea96c0d7e2de5989b754c81b53febb555ba611c98b5099e3f68d7d1a505451
a2abf298696b5501be0509c077d61c9c565670b56ea0e63d4fb61674e04efb5c86282ec3596d0648e4a918cc83b0c7acb4092
c77340b6ce8145dad616efac63bd025c7dcb408c05941b25ac0231829a6bb3ca6534b461bc5b7808c831c5dd3eb5b8193c80a
ea6ccffa7cf879ab3ad02c2363b13b277e5fa836a820dbdfd1e58e9baffd4a34443e7f9222b64147256c0168fe8c5ce7b8ebd
74e3c57d445710a35df23abe4b1f969a060205befac25b791f7fd89070bd419548316505820d5935b777d04632ee2234518b9
5e307189f9d5d9b53278bf818823811d28500c9e3c13dfe6f079ed67b3fc5d3656d023d1ed61ae1d32d13fc6eade0f0bc1c79
f03f0608ef128ca");

    //S: S^e mod n = P
    BN_mod_exp(C, S, e, n, ctx);

    printf("%s %s\n", "now is ",BN_bn2hex(C));
    //compare the generated text with the original padded plain text
    if (BN_cmp(C, P) == 0)
    {
        printf("The signature is valid");
    }
    else
    {
        printf("Fail");
    }

    return 0;
}
```

result shown as below

```
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/a2$ gcc -o task6 task6.c -lcrypto
james@LAPTOP-7AJ4SITH:/mnt/c/james/870/a2$ ./task6
now is  01FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFF
FFFFFFFFFFFFFFFFFF003041300D060960864801650304020205000430F1B49DA16D7E466
98607236395B19E1D67FDD647F6AC42A19B86F087172E69724D19FB1492076750ED018D92
E670A4FA
The signature is validjames@LAPTOP-7AJ4SITH:/mnt/c/james/870/a2$
```