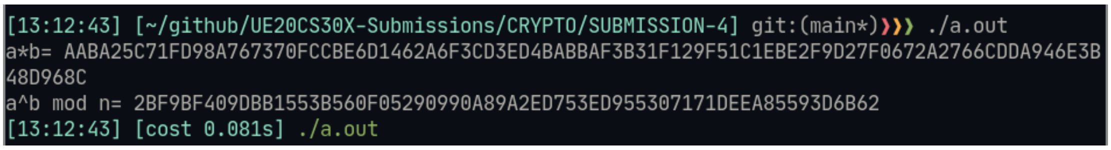
Cryptograhphy Hands-On submission 4 | RSA

Details :

- SRN : PES2UG20CS237
- Name : P K Navin Shrinivas
- Section : D

Task 1 : BIGNUM

Screenshots :



```
[13:12:43] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❯❯❯ ./a.out
a*b= AABA25C71FD98A767370FCCBE6D1462A6F3CD3ED4BABBAF3B31F129F51C1EBE2F9D27F0672A2766CDDA946E3B
48D968C
a^b mod n= 2BF9BF409DBB1553B560F05290990A89A2ED753ED955307171DEEA85593D6B62
[13:12:43] [cost 0.081s] ./a.out
```

Code :

```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
void printBN(char*msg, BIGNUM*a) {
  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 *a = BN_new();
  BIGNUM *b = BN_new();
  BIGNUM *n = BN_new();
  BIGNUM *res = BN_new();
  // Initialize
  BN_generate_prime_ex(a,NBITS,1,NULL,NULL,NULL);
  BN_dec2bn(&b,"273489463796838501848592769467194369268");
  BN_rand(n,NBITS,0,0);
  // res = a*b
  BN_mul(res,a,b,ctx);
  printBN("a*b=",res);
  // res = a^b mod n
```

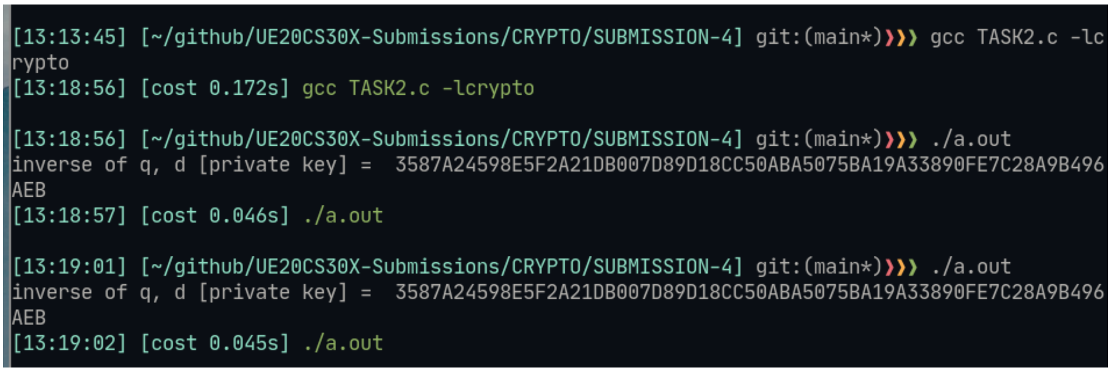
```
    BN_mod_exp(res,a,b,n,ctx);
    printBN("a^b mod n=",res);
    return 0;
  }
```

Observation :

- We were able to multiply and do operations over VERY LARGE number usingn BIGNUM. Which otherwise is not possible in C integers.
- Also, we note that all the big number when converted from binary to other formats are stored in char* (string) type.

TASK 2 : Calclatuing pivate key given P, Q and E

Screenshots :



Code :

```
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
void printBN(char*msg, BIGNUM*a) {
  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();
  BIGNUM *d = BN_new();
```

```c
    BIGNUM *p_minus_one = BN_new();
    BIGNUM *q_minus_one = BN_new();
    BIGNUM *phi_pq = BN_new();
    BIGNUM *one = BN_new();
    // Initialize
    BN_hex2bn(&p,"F7E75FDC469067FFDC4E847C51F452DF");
    BN_hex2bn(&q,"E85CED54AF57E53E092113E62F436F4F");
    BN_hex2bn(&e,"0D88C3");
    BN_hex2bn(&one,"1");
    BN_sub(p_minus_one,p,one);
    BN_sub(q_minus_one,q,one);
    BN_mul(phi_pq,p_minus_one,q_minus_one,ctx);
    BN_mod_inverse(d,e,phi_pq,ctx);
    printBN("inverse of q, d [private key] = ",d);
    return 0;
}
```

Observation :

- Given p,q and e. Calculation of private key d is very easy, almost and constant time process.
- It remains constant for a given set of p, q and e.
- In the absense of any one p, q and e, calculating d becomes exponentially harder.
- Cacluating inverse of e in mod space of phi becomes easier as calculating phi of a number whos prime constituents (p*q) is easy and is simply (p-1)*(q-1). Finding the two primes that constitutes a non-prime number is pretty much a hit or trial method.

TASK 3 : Encrpyting and Decrypting a message using RSA.

Screenshots :

```
[14:00:57] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)››› gcc TASK3.c -lcrypto
[14:00:58] [cost 0.179s] gcc TASK3.c -lcrypto

[14:00:58] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)››› ./a.out
Enter string for enc : Hello world
Plain text in hex code : 48656C6C6F20776F726C64
Encrypted Message = 4DE9882932DA74288F66E968BB023972E2FB50BC135C5637BF7C0739C706182F
Decrypted Message = 48656C6C6F20776F726C64
[14:01:01] [cost 2.118s] ./a.out
```

Code :

> Note : The code has been modified, in such a way that it take input from user, converts to hex and does RSA operations on the same!

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


void alphabets_to_hexcode(char* string, char* res){
  int i = 0;
  int out = 0;
  while(string[i]) {
    sprintf((char*)(res+out),"%02X",string[i]);
    out+=2;
    i++;
  }
}

void printBN(char*msg, BIGNUM*a) {
  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 *m = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *n = BN_new();
  BIGNUM *d = BN_new();
  BIGNUM *enc = BN_new();
  BIGNUM *dec = BN_new();

  char string[100];
  char res[100];
  printf("Enter string for enc : ");
  scanf("%[^\n]%*c",string);
  alphabets_to_hexcode(string,res);

  printf("Plain text in hex code : ");

  int out_print = 0;
  while(res[out_print]){
    printf("%c", res[out_print]);
    out_print++;
  }
  printf("\n");

  // Initialize
  BN_hex2bn(&m,res);
  BN_hex2bn(&e,"010001");
```

```
BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB816292


BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381

  // Encryption
  BN_mod_exp(enc,m,e,n,ctx);
  printBN("Encrypted Message =",enc);
  // Decryption
  BN_mod_exp(dec,enc,d,n,ctx);
  printBN("Decrypted Message =",dec);
  return 0;
}
```

Observation :

- None, except for the fact that RSA indeed works!

TASK 4 : Decryption of a hex code

Screenshots :

```
[17:37:00] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❱❱❱ gcc TASK4.c -lcrypto
[17:37:01] [cost 0.177s] gcc TASK4.c -lcrypto

[17:37:01] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❱❱❱ ./a.out
Decrypted Message in hex = 50617373776F72642069732064656573
Decrypted Message in ascii = Password is dees
[17:37:02] [cost 0.044s] ./a.out

[17:37:06] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❱❱❱
```

Code :

> Note : this is modified code to also pint ascii equivalent of hex code!

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

void hex_to_string(char* msg, char* hex)
{
   int hex_sz = 0;
   while(hex[hex_sz]){
     hex_sz+=1;
   }
```

```
    int msg_sz = (hex_sz/2)+1;
    if (hex_sz % 2 != 0 || hex_sz/2 >= msg_sz)
       return;

    for (int i = 0; i < hex_sz; i+=2)
    {
       uint8_t msb = (hex[i+0] <= '9' ? hex[i+0] - '0' : (hex[i+0] &
0x5F) - 'A' + 10);
       uint8_t lsb = (hex[i+1] <= '9' ? hex[i+1] - '0' : (hex[i+1] &
0x5F) - 'A' + 10);
       msg[i / 2] = (msb << 4) | lsb;
       msg[i+1] = '\0';
    }
}
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);
  char res[100];
  hex_to_string(res,number_str);
  printf("%s in hex = %s\n",msg,number_str);
  printf("%s in ascii = %s\n",msg,res);
  OPENSSL_free(number_str);
}

int main() {
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *m = BN_new();
  BIGNUM *e = BN_new();
  BIGNUM *n = BN_new();
  BIGNUM *d = BN_new();
  BIGNUM *enc = BN_new();
  BIGNUM *dec = BN_new();
  // Initialize

BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB816292


BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381


BN_hex2bn(&enc,"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567

      );
  // Decryption
  BN_mod_exp(dec,enc,d,n,ctx);
  printBN("Decrypted Message",dec);
  return 0;
}
```

Observation :

- Given d, which is the modular inverse of e. Decryption is very easy, all we needs to do is enc^d % n;

TASK 5 : Getting signature of a string

Screenshots :

```
[17:53:33] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)⟫⟫⟫ gcc TASK5.c -lcrypto
[17:53:36] [cost 0.183s] gcc TASK5.c -lcrypto

[17:53:36] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)⟫⟫⟫ ./a.out
Sign = 80A55421D72345AC199836F60D51DC9594E2BDB4AE20C804823FB71660DE7B82
[17:53:37] [cost 0.044s] ./a.out
```

> Note : Instead of using python for step 1, ive included a function in C to convert plain text to hex code!

Code :

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

void alphabets_to_hexcode(char* string, char* res){
  int i = 0;
  int out = 0;
  while(string[i]) {
    sprintf((char*)(res+out),"%02X",string[i]);
    out+=2;
    i++;
  }
}

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);
}
int main() {
  BN_CTX *ctx = BN_CTX_new();
  BIGNUM *m = BN_new();
  BIGNUM *n = BN_new();
  BIGNUM *d = BN_new();
```

```
    BIGNUM *sign = BN_new();
    // Initialize
    char hex_code[1000];
    char plain_text[] = "I owe you $2000";
    alphabets_to_hexcode(plain_text, hex_code);
    BN_hex2bn(&m,hex_code);

BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB816292


BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381

    // Signing
    BN_mod_exp(sign,m,d,n,ctx);
    printBN("Sign =",sign);
    return 0;
}
```

Observation :

- A point clearly observed is that the signature is much bigger than plain text, this maybe due to the fact of avalance effect.

TASK 6 : Verifying the previous signature

Screenshots :

Observation :

- We can observe a valid string sequence that come out of signature verification. Implying that it is indeed Alice's signature.

TASK 7 : Verifying X.509 certificate

Step 1 :

```
Server certificate
subject=CN = *.stackexchange.com

issuer=C = US, O = Let's Encrypt, CN = R3
```

- Here we see certificates from stackexchange.com. This particular website has been chosen as it uses RSA standards for encryption.

Step 2 :

```
[08:06:13] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❯❯❯ openssl x509 -in c1.pem -noout -modulus
Modulus=BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB52BB9C54CB1AF8E6BF975C8A3D70F479414553557 8C9EA8A23919F5823C42A94E6EF53BC32EDB8DC0B05CF35938E7EDCF69F05A0B1BBEC094242587FA377
1B313E71CACE19BEFDBE43B45524596A9C153CE34C852EEB5AEED8FDE6070E2A554ABB66D0E97A540346B2BD3BC66EB66347CFA6B8B8F572999F830175DBA726FFB81C5ADD286583D17C7E709BBF12BF786DCC1DA715DD446E3CCAD25C188
BC60677566B3F118F7A25CE653FF3A88B647A5FF1318EA9809773F9D53F9CF01E5F5A6701714AF63A4FF99B3939DDC53A706FE48851DA169AE2575BB13CC5203F5ED51A18BDB15
[08:06:14] [cost 0.062s] openssl x509 -in c1.pem -noout -modulus

[08:06:29] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❯❯❯ openssl x509 -in c1.pem -text -noout |grep "Exponent"
                Exponent: 65537 (0x10001)
[08:06:33] [cost 0.068s] openssl x509 -in c1.pem -text -noout |grep "Exponent"
```

- Here we see the modulus (n) ans also the exponent value (e)

Step 3 :

```
[08:11:43] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❯❯❯ cat > signature
85:ca:4e:47:3e:a3:f7:85:44:85:bc:d5:67:78:b2:98:63:ad:
        75:4d:1e:96:3d:33:65:72:54:2d:81:a0:ea:c3:ed:f8:20:bf:
        5f:cc:b7:70:00:b7:6e:3b:f6:5e:94:de:e4:20:9f:a6:ef:8b:
        b2:03:e7:a2:b5:16:3c:91:ce:b4:ed:39:02:e7:7c:25:8a:47:
        e6:65:6e:3f:46:f4:d9:f0:ce:94:2b:ee:54:ce:12:bc:8c:27:
        4b:b8:c1:98:2f:a2:af:cd:71:91:4a:08:b7:c8:b8:23:7b:04:
        2d:08:f9:08:57:3e:83:d9:04:33:0a:47:21:78:09:82:27:c3:
        2a:c8:9b:b9:ce:5c:f2:64:c8:c0:be:79:c0:4f:8e:6d:44:0c:
        5e:92:bb:2e:f7:8b:10:e1:e8:1d:44:29:db:59:20:ed:63:b9:
        21:f8:12:26:94:93:57:a0:1d:65:04:c1:0a:22:ae:10:0d:43:
        97:a1:18:1f:7e:e0:e0:86:37:b5:5a:b1:bd:30:bf:87:6e:2b:
        2a:ff:21:4e:1b:05:c3:f5:18:97:f0:5e:ac:c3:a5:b8:6a:f0:
        2e:bc:3b:33:b9:ee:4b:de:cc:fc:e4:af:84:0b:86:3f:c0:55:
        43:36:f6:68:e1:36:17:6a:8e:99:d1:ff:a5:40:a7:34:b7:c0:
        d0:63:39:35:39:75:6e:f2:ba:76:c8:93:02:e9:a9:4b:6c:17:
        ce:0c:02:d9:bd:81:fb:9f:b7:68:d4:06:65:b3:82:3d:77:53:
        f8:8e:79:03:ad:0a:31:07:75:2a:43:d8:55:97:72:c4:29:0e:
        f7:c4:5d:4e:c8:ae:46:84:30:d7:f2:85:5f:18:a1:79:bb:e7:
        5e:70:8b:07:e1:86:93:c3:b9:8f:dc:61:71:25:2a:af:df:ed:
        25:50:52:68:8b:92:dc:e5:d6:b5:e3:da:7d:d0:87:6c:84:21:
        31:ae:82:f5:fb:b9:ab:c8:89:17:3d:e1:4c:e5:38:0e:f6:bd:
        2b:bd:96:81:14:eb:d5:db:3d:20:a7:7e:59:d3:e2:f8:58:f9:
        5b:b8:48:cd:fe:5c:4f:16:29:fe:1e:55:23:af:c8:11:b0:8d:
        ea:7c:93:90:17:2f:fd:ac:a2:09:47:46:3f:f0:e9:b0:b7:ff:
        28:4d:68:32:d6:67:5e:1e:69:a3:93:b8:f5:9d:8b:2f:0b:d2:
        52:43:a6:6f:32:57:65:4d:32:81:df:38:53:85:5d:7e:5d:66:
        29:ea:b8:dd:e4:95:b5:cd:b5:56:12:42:cd:c4:4e:c6:25:38:
        44:50:6d:ec:ce:00:55:18:fe:e9:49:64:d4:4e:ca:97:9c:b4:
        5b:c0:73:a8:ab:b8:47:c2
[08:11:50] [cost 4.455s] cat > signature
```

```
[08:12:17] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❯❯❯ cat signature | tr -d '[:space:]:'
85ca4e473ea3f7854485bcd56778b29863ad754d1e963d336572542d81a0eac3edf820bf5fccb77000b76e3bf65e94dee4209fa6ef8bb203e7a2b5163c91ceb4ed3902e77c258a47e6656e3f46f4d9f0ce942bee54ce12bc8c274bb8c1982
fa2afcd71914a08b7c8b8237b042d08f908573e83d904330a472178098227c32ac89bb9ce5cf264c8c0be79c04f8e6d440c5e92bb2ef78b10e1e81d4429db5920ed63b921f81226949357a01d6504c10a22ae100d4397a1181f7ee0e08637
b55ab1bd30bf876e2b2aff214e1b05c3f51897f05eacc3a5b86af02ebc3b33b9ee4bdeccfce4af840b863fc0554336f668e136176a8e99d1ffa540a734b7c0d063393539756ef2ba76c89302e9a94b6c17ce0c02d9bd81fb9fb768d40665b
3823d7753f88e7903ad0a3107752a43d8559772c4290ef7c45d4ec8ae468430d7f2855f18a179bbe75e708b07e18693c3b98fdc6171252aafdfed255052688b92dce5d6b5e3da7dd0876c842131ae82f5fbb9abc889173de14ce5380ef6bd
2bbd968114ebd5db3d20a77e59d3e2f858f95bb848cdfe5c4f1629fe1e5523afc811b08dea7c9390172ffdaca20947463ff0e9b0b7ff284d6832d6675e1e69a393b8f59d8b2f0bd25243a66f3257654d3281df3853855d7e5d6629eab8dde
495b5cdb5561242cdc44ec6253844506decce005518fee94964d44eca979cb45bc073a8abb847c2%
[08:12:25] [cost 0.049s] cat signature | tr -d '[:space:]:'

[08:12:35] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❯❯❯
```

- Siganture type is RSA, but it was not possible to pull ouot signature using grep, hence we will have to printout the entire info.
- Doing cat on signature file ignoring space and ":", we get the siganture as a block of hex code.

Step 4 :

```
08:16:36] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❯❯❯ gcc TASK7.c -lcrypto
08:16:38] [cost 0.178s] gcc TASK7.c -lcrypto

08:16:38] [~/github/UE20CS30X-Submissions/CRYPTO/SUBMISSION-4] git:(main*)❯❯❯ ./a.out
essage = AFD06E3E9DDF190A4D57416FCEF217E9C73C9A0D18B9D04994B2F85C1A889F317F68FAC59FFA7C870C4D7C9F24006E7C57D69C171DC2E70BF7E2EDB0E33439778F21639DA198A55565357347FCB180789DCE495EC7D984C7B2F
B0325645DCBBDCB2D53BC39C799B6AA40310E6D9CA2FD32834046323DA7B1AF5845AEAB7BE1E0AF1FBCF40027A08CADDBE58B8594C9AEDDDF63AB7CDDBE48056D0921CE6F200C3827AFA46C42A320B3ED383E317FFDEE00A84D82A7D2F83
6C2A23F2E2038EE1F3C7DE39192DF47E604A94B8BC1FB4975CD8A6CCC4D9D247CE120D28B9E2085EAE892A8A911D537018B5EF56A91A7C107A5708A62193C5857EB22CCAE2CA04D
08:16:39] [cost 0.053s] ./a.out
```

Code :

```c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256
void printBN(char*msg, BIGNUM*a) {
  /* Use BN_bn2hex(a) for hex string
     Use BN_bn2dec(a) for decimal string*/
```

```c
    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 *s = BN_new();
    BIGNUM *n = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *message = BN_new();
    // Initialize

  BN_hex2bn(&s,"85ca4e473ea3f7854485bcd56778b29863ad754d1e963d336572542d8


  BN_hex2bn(&n,"BB021528CCF6A094D30F12EC8D5592C3F882F199A67A4288A75D26AAB

    BN_hex2bn(&e,"10001");
    // Signing
    BN_mod_exp(message,s,e,n,ctx);
    printBN("Message =",message);
    return 0;
  }
```

- Here we see the decrypted message that was used to create this certificate. This would have been taken at random by the certificate provider.