Applied Cryptography Lab-05 Manual

10 October 2022 21:43

> ***Note:*** *The demo showed in the video for this lab has been performed on Ubuntu 22.04 running on wsl2. So far, there have been no changes noted between the execution of this lab on wsl and execution on the seedlabs Ubuntu 20.04 virtualbox vm.*
> *Therefore, despite the differences in environment, all tasks in the lab should run smoothly.*

Task 1: A Complete Example of BIGNUM

The program below shows a complete example of BIGNUM. This program uses three BIGNUM variables, a, b, and n; and then compute a ∗ b and (a b mod n)

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*/
    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;
}
```

Commands

```
$ gcc task1.c -o task1 -lcrypto
$ ./task1
```

Give your observations with a screenshot

Task 2: Deriving the Private Key

The objective of this task is to derive private key. Given are the hexadecimal values of p, q, e, and public key pair (e,n)

```
p = F7E75FDC469067FFDC4E847C51F452DF
q = E85CED54AF57E53E092113E62F436F4F
e = 0D88C3
```

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*/
    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();
    BIGNUM *res1 = BN_new();
    BIGNUM *res2 = BN_new();
    BIGNUM *res3 = 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(res1,p,one);
    BN_sub(res2,q,one);
    BN_mul(res3,res1,res2,ctx);
    BN_mod_inverse(d,e,res3,ctx);
    printBN("d=",d);
    return 0;
}
```

Commands

```
$ gcc task2.c -o task2 -lcrypto
$ ./task2
```

- **Give your observations with a screenshot.**
- **Explain your understanding (in terms of mathematical statements) of what the above code**

does

Task 3: Encrypting a Message

The objective of this task is to encrypt a given message. Given are the hexadecimal values of n, e, M (you can use whatever message you want). The value of “d” is also given to verify the result.

n = DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5

e = 010001 (This hex value equal to decimal 65537)

M = A top secret!

d = 74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D

Step 1

Convert the ASCII String message to a hex string

```
python3 -c "print('A top secret!'.encode().hex())"
```

Give your observation with a screenshot

Step 2

Execute the below program to encrypt the message M and verify it by decrypting it.

Code

```
#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*/
    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();
    // Initialize
    BN_hex2bn(&m,"<< Enter the message in hex, obtained in step 1 >>");
    BN_hex2bn(&e,"010001");
    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    // 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;
}
```

Commands

```
$ gcc task3.c -o task3 -lcrypto
$ ./task3
```

Task 4: Decrypting a Message

The objective of this task is to decrypt a given ciphertext. given are the hexadecimal values of n, e, d from the above task

```
C =
8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567E
A1E2493F
```

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*/
    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();
    // Initialize
    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    BN_hex2bn(&enc,"8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F"
);
    // Decryption
    BN_mod_exp(dec,enc,d,n,ctx);
    printBN("Decrypted Message =",dec);
    return 0;
}
```

Commands

```
$ gcc task4.c -o task4 -lcrypto
$ ./task4
```

Give your observation with screenshot

Task 5: Signing a Message

The objective of this task is to generate a signature for the following message. Use the public/private key set from task3

M= I owe you $2000

Step 1

Generate hex for M

```
python3 -c "print('I owe you $2000'.encode().hex())"
```

Step 2

Execute the e following program to generate signature of the given message. Using the signing algorithm M^d mod n

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*/
    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
    BN_hex2bn(&m,"<< Hex value of M >>");
    BN_hex2bn(&n,"DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&d,"74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
    // Signing
    BN_mod_exp(sign,m,d,n,ctx);
    printBN("Sign =",sign);
    return 0;
}
```

Commands

```
$ gcc task5.c -o task5 -lcrypto
$ ./task5
```

Step 3

Execute steps 1 and 2 for the message "I owe $3000"

Give your observations with screenshot

Task 6: Verifying a signature

The objective of this task is to verify if the signature received by Bob is Allice’s or not. Given are the Message M, signature S, Allice public key e and n.

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*/
    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,"643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802
F");
    BN_hex2bn(&n,"AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F1811611
5");
    BN_hex2bn(&e,"010001");
    // Signing
    BN_mod_exp(message,s,e,n,ctx);
    printBN("Message =",message);
    return 0;
}
```

Commands

```
$ gcc task6.c -o task6 -lcrypto
$ ./task6
$ python3 -c "print(bytes.fromhex('<< output of
task6 >>'))"
```

Task 7: Manually Verifying X.509 Certificate

The objective of this task is to verify the signature of a public key certificate from a server and show that the signature matches

To verify that a certificate was signed by a specific certificate authority we need the following details

1. Public key of the certificate authority (issuer).

2. signature and algorithm used to generate signature from the server's certificate.

Step 1

Download the certificate from any website (each student use a different website)

```
$ openssl s_client -connect www.example.org:443 -showcerts
```

Copy server certificate to c0.pem file and root certificate of the issuer to c1.pem
Give your observation with screen shot.

Step 2

Extract the public key (e, n) from the issuer's certificate. Openssl provides commands to extract certain attributes from the x509 certificates. We can extract the value of n using -modulus. There is no specific command to extract e, but we can print out all the fields and can easily find the value of e.

```
$ openssl x509 -in c1.pem -noout -modulus
$ openssl x509 -in c1.pem -text -noout |grep "Exponent"
```

Give your observation with screenshot

Step 3

Extract the signature from the server's certificate. There is no specific openssl command to extract the signature field. However, we can print out all the fields and then copy and paste the signature block into a file (note: if the signature algorithm used in the certificate is not based on RSA, find another certificate).

Commands

```
$openssl x509 -in c0.pem -text -noout
//extract only the signature part and paste it in signature file
$ cat signature | tr -d '[:space:]:'
Give your observation with screenshot.
```

Step 4

Verify the signature by substituting the values just found out, in the code given below and running it.

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*/
    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,"<< signature >>");
    BN_hex2bn(&n,"<< modulus >>");
    BN_hex2bn(&e,"<< exponent >>");
    // Signing
    BN_mod_exp(message,s,e,n,ctx);
    printBN("Message =",message);
    return 0;
}
```

Commands

```
$ gcc task7.c -o task7 -lcrypto
$ ./task7
```

Give your observation with screenshot