Leonardo Blas

RSA Public-Key Encryption and Signature Lab

Task 1

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

void printBN(char *msg, BIGNUM * p) {
        char * number_str = BN_bn2hex(p);
        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 *n = BN_new();
        BIGNUM *_p = BN_new();
        BIGNUM *_q = BN_new();
        BIGNUM *phi = BN_new();
        BIGNUM *d = BN_new();
        BN_hex2bn(&p, "F7E75FDC469067FFDC4E847C51F452DF");
        BN_hex2bn(&q, "E85CED54AF57E53E092113E62F436F4F");
        BN_hex2bn(&e, "0D88C3");
        BN_hex2bn(&_p, "F7E75FDC469067FFDC4E847C51F452DE");
        BN_hex2bn(&_q, "E85CED54AF57E53E092113E62F436F4E");
        BN_mul(n, p, q, ctx);
        BN_mul(phi, _p, _q, ctx);
        BN_mod_inverse(d, e, phi, ctx);
        printBN("p*q = ", n);
        printf("d: e*d mod (p-1)(q-1) = 1\n");
        printBN("d = ", d);
}
[10/28/21]seed@VM:~/Desktop$ gcc RSA_1.c -o out -lcrypto
[10/28/21]seed@VM:~/Desktop$ ./out
p*q = E103ABD94892E3E74AFD724BF28E78366D9676BCCC70118BD0AA1968DBB143D1
d: e*d mod (p-1)(q-1) = 1
d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

In this snippet, we use the given values for p, q, and e, and calculate private key d. We learn that d = 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB.

Task 2

```
[10/29/21]seed@VM:~/Desktop$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import codecs
>>> codecs.encode(b'A top secret!', 'hex')
b'4120746f702073656372657421'
```

Here, we convert the ASCII string "A top secret!" into a hex string, in Python, and obtain: '4120746f702073656372657421'.

```
[10/29/21]seed@VM:~/Desktop$ cat RSA_2.c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

void printBN(char *msg, BIGNUM * p) {
        char * number_str = BN_bn2hex(p);
        printf("%s%s\n", msg, number_str);
        OPENSSL_free(number_str);
}

int main () {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *n = BN_new();
        BIGNUM *e = BN_new();
        BIGNUM *M = BN_new();
        BIGNUM *C = BN_new();
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4
D0CB81629242FB1A5");
        BN_hex2bn(&e, "010001");
        BN_hex2bn(&M, "4120746f702073656372657421");
        BN_mod_exp(C, M, e, n, ctx);
        printBN("C = ", C);
}
[10/29/21]seed@VM:~/Desktop$ gcc RSA_2.c -o out -lcrypto
[10/29/21]seed@VM:~/Desktop$ ./out
C = 6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
```

In this image, we encrypt the previous hex string, using the given n and e, and obtain '6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC'.

Task 3:

```
[10/29/21]seed@VM:~/Desktop$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import codecs
>>> codecs.decode('4120746f702073656372657421', 'hex')
b'A top secret!'
```

In this snippet, we decode the hex string ‘4120746f702073656372657421’, in Python, and obtain the ASCII string: 'A top secret!'.

```
[10/29/21]seed@VM:~/Desktop$ cat RSA_3.c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

void printBN(char *msg, BIGNUM * p) {
        char * number_str = BN_bn2hex(p);
        printf("%s%s\n", msg, number_str);
        OPENSSL_free(number_str);
}

int main () {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *n = BN_new();
        BIGNUM *M = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *C = BN_new();
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4
D0CB81629242FB1A5");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AA
CBC26AA381CD7D30D");
        BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB
67396567EA1E2493F");
        BN_mod_exp(M, C, d, n, ctx);
        printBN("M = ", M);
}
[10/29/21]seed@VM:~/Desktop$ gcc RSA_3.c -o out -lcrypto
[10/29/21]seed@VM:~/Desktop$ ./out
M = 50617373776F72642069732064656573
```

Here, we decrypt the given the given ciphertext, C, given n and d. We obtain ‘50617373776F72642069732064656573’.

```
[10/29/21]seed@VM:~/Desktop$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import codecs
>>> codecs.decode('50617373776F7264206973206465657', 'hex')
b'Password is dees'
```

In this image, we decode the previously obtained hex string, in Python, and obtain the ASCII string ‘Password is dees’.

Task 4

```
[10/29/21]seed@VM:~/Desktop$ python3
Python 3.8.5 (default, Jul 28 2020, 12:59:40)
[GCC 9.3.0] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import codecs
>>> codecs.encode(b'I owe you $2000.', 'hex')
b'49206f776520796f752024323030302e'
>>> codecs.encode(b'I owe you $3000.', 'hex')
b'49206f776520796f752024333030302e'
```

In this snippet, we convert the ASCII strings ‘I owe you $2000.’ and ‘I owe you $3000.’ into hex strings in Python. We obtain: '49206f776520796f752024323030302e' and '49206f776520796f752024333030302e', respectively.

```
[10/29/21]seed@VM:~/Desktop$ cat RSA_4.c
#include <stdio.h>
#include <openssl/bn.h>
#define NBITS 256

void printBN(char *msg, BIGNUM * p) {
        char * number_str = BN_bn2hex(p);
        printf("%s%s\n", msg, number_str);
        OPENSSL_free(number_str);
}

int main () {
        BN_CTX *ctx = BN_CTX_new();
        BIGNUM *n = BN_new();
        BIGNUM *M = BN_new();
        BIGNUM *_M = BN_new();
        BIGNUM *d = BN_new();
        BIGNUM *C = BN_new();
        BIGNUM *_C = BN_new();
        BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4
D0CB81629242FB1A5");
        BN_hex2bn(&M, "49206f776520796f752024323030302e");
        BN_hex2bn(&_M, "49206f776520796f752024333030302e");
        BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AA
CBC26AA381CD7D30D");
        BN_mod_exp(C, M, d, n, ctx);
        BN_mod_exp(_C, _M, d, n, ctx);
        printBN("Signature 1: ", C);
        printBN("Signature 2: ", _C);
}
[10/29/21]seed@VM:~/Desktop$ gcc RSA_4.c -o out -lcrypto
[10/29/21]seed@VM:~/Desktop$ ./out
Signature 1: 55A4E7F17F04CCFE2766E1EB32ADDBA890BBE92A6FBE2D785ED6E73CC
B35E4CB
Signature 2: BCC20FB7568E5D48E434C387C06A6025E90D29D848AF9C3EBAC0135D9
9305822
```

Finally, in this image we generate the signatures for the previous, slightly different, messages. The difference between the strings was only one character, 2 for 3, yet these messages' signatures are very different. By comparing these signatures, we learn that changing even one character in a message may drastically affect the encryption.