RSA PUBLIC-KEY ENCRYPTION AND SIGNATURE LAB

Jasmine Joy (500924677)

Task 2.1: BIGNUM APIs

The `bn_sample.c` file:

bn_sample.c (~/Desktop/Lab4) - gedit

Open

```c
#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 a, b, n
    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^c mod n = ", res);
    return 0;
}
```

Running bn_sample.c, we get:

```
[03/25/20]seed@VM:~/.../Lab4$ gcc bn_sample.c -lcrypto
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out        task3    task4    task5    task6    task7    test
bn_sample.c  task3.c  task4.c  task5.c  task6.c  task7.c  test.c
[03/25/20]seed@VM:~/.../Lab4$ ./a.out
a * b =  A12898D6007919B0F5D6BF4D804362E8D546C6B62E014B0A0223B7AB24B71CD634E7DE205087E21B396F759D2B
71201C
a^c mod n =  15D489AD54325E61A413589A3B9A2841AB9A5C11BCAF5E1FEEDD67507401812D
[03/25/20]seed@VM:~/.../Lab4$
```

Task 3: Deriving the Private Key

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

We can calculate the private key, pk, from the *p,q and e* given. All parameters is converted to bignum. Executing the following code:

bn_sample.c (~/Desktop/Lab4) - gedit

Open

```c
#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 *p1 = BN_new();
    BIGNUM *q = BN_new();
    BIGNUM *q1 = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *res = BN_new();
    BIGNUM *pk = BN_new();
    BIGNUM *one = BN_new();

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

    BN_dec2bn(&one, "1");

    // p1 = p-1
    BN_sub(p1, p, one);

    // q1 = q-1
    BN_sub(q1, q, one);

    // res = p1*q1
    BN_mul(res, p1, q1, ctx);

    // e*pk mod res
    BN_mod_inverse(pk, e, res, ctx);

    printBN("Private key = ", pk);
    return 0;
}
```

We get:

```
[03/25/20]seed@VM:~/.../Lab4$ gcc -o task3 task3.c -lcrypto
[03/25/20]seed@VM:~/.../Lab4$ ./task3
Private key =  3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
[03/25/20]seed@VM:~/.../Lab4$
```

Task 4: Encrypting a message

We can get the hex of “A top secret!” from python

```
[03/24/20]seed@VM:~/.../Lab4$ 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.
>>> "A top secret!".encode("hex")
'4120746f702073656372657421'
>>>
```

We can calculate the encrypted message E, from the *M,n and e* given using the formula M^e mod n.

All parameters is converted to bignum. Executing the following code:

*task4.c (~/Desktop/Lab4) - gedit

Open

```c
#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 *M = BN_new(); // Message
    BIGNUM *n = BN_new();
    BIGNUM *E = BN_new(); // Encrypted
    BIGNUM *D = BN_new(); // Decrypted
    BIGNUM *e = BN_new();
    BIGNUM *d = BN_new();

    // Initialize M, n, e, d
    BN_hex2bn(&M, "4120746f702073656372657421"); // result from python encode to hex
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // Encrypting (Formula: M^e mod n)
    BN_mod_exp(E, M, e, n, ctx);
    printBN("Encrypted Message = ", E);

    // Decrypting (Formula: E^d mod n)
    BN_mod_exp(D, E, d, n, ctx);
    printBN("Decrypted Message = ", D);

    return 0;
}
```

We can check the result by decrypting the encrypted message to check of that is what we started out with:

```
[03/25/20]seed@VM:~/.../Lab4$ gcc -o task4 task4.c -lcrypto
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out  bn_sample.c  task4  task4.c  test  test.c
[03/25/20]seed@VM:~/.../Lab4$ ./task4
Encrypted Message =  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
Decrypted Message =  4120746F702073656372657421
[03/25/20]seed@VM:~/.../Lab4$
```

In this case, it is true.

Task 5: Decrypting a message

The cipher text can be decrypted using: *c^d mod n.*

task5.c (~/Desktop/Lab4) - gedit

Open

```c
#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 *n = BN_new();
    BIGNUM *D = BN_new(); // Decrypted
    BIGNUM *C = BN_new();
    BIGNUM *d = BN_new();

    // Initialize M, n, e, d
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&C, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // Decrypting (Formula: C^d mod n)
    BN_mod_exp(D, C, d, n, ctx);
    printBN("Decrypted Message = ", D);

    return 0;
}
```

Upon decryption, we will get the hex of the original message, which we can then decode using python.

```
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out  bn_sample.c  task4  task4.c  task5.c  test  test.c
[03/25/20]seed@VM:~/.../Lab4$ gcc -o task5 task5.c -lcrypto
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out  bn_sample.c  task4  task4.c  task5  task5.c  test  test.c
[03/25/20]seed@VM:~/.../Lab4$ ./task5
Decrypted Message =  50617373776F72642069732064656573
[03/25/20]seed@VM:~/.../Lab4$ 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.
>>> "50617373776F72642069732064656573
  File "<stdin>", line 1
    "50617373776F72642069732064656573
                                     ^
SyntaxError: EOL while scanning string literal
>>> "50617373776F72642069732064656573".decode("hex")
'Password is dees'
>>>
```

Task 6: Signing a message

Using python to get the hex value:

```
[03/25/20]seed@VM:~/.../Lab4$ 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.
>>> "I owe you $2000".encode("hex")
'49206f776520796f75202432303030'
>>>
KeyboardInterrupt
>>>
[5]+  Stopped                 python
```

Running code for “I owe you $2000”:

```
task6.c (~/Desktop/Lab4) - gedit
Open
```

```c
#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 *Message = BN_new(); // Message
    BIGNUM *n = BN_new();
    BIGNUM *result = BN_new();
    BIGNUM *d = BN_new();

    // Initialize M, n, e, d
    BN_hex2bn(&Message, "49206f776520796f75202432303030"); // hex of "I owe you $2000"
    BN_hex2bn(&n, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");

    // Decrypting (Formula: C^d mod n)
    BN_mod_exp(result, Message, d, n, ctx);
    printBN("Message Signature = ", result);

    return 0;
}
```

```
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out  bn_sample.c  task4  task4.c  task5  task5.c  task6.c  test  test.c
[03/25/20]seed@VM:~/.../Lab4$ gcc -o task6 task6.c -lcrypto
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out  bn_sample.c  task4  task4.c  task5  task5.c  task6  task6.c  test  test.c
```

We get the signature:

```
[03/25/20]seed@VM:~/.../Lab4$ ./task6
Message Signature =  2C595D23711B74724D86428BF314D89D8B7BEB6FCF6B35D29FE08A7AB5DF7CA0
[03/25/20]seed@VM:~/.../Lab4$
```

Changing the message from “I owe you $2000” to “I owe you $2001”

```c
// Initialize M, n, e, d
//BN_hex2bn(&Message, "49206f776520796f75202432303030"); // hex of "I owe you $2000"
BN_hex2bn(&Message, "49206f776520796f75202432303031"); // hex of "I owe you $2001"
BN_hex2bn(&n, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

```
[03/25/20]seed@VM:~/.../Lab4$ 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.
>>> "I owe you $2001".encode("hex")
'49206f776520796f75202432303031'
>>>
[6]+  Stopped                 python
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out  bn_sample.c  task4  task4.c  task5  task5.c  task6  task6.c  test  test.c
[03/25/20]seed@VM:~/.../Lab4$ gcc -o task6 task6.c -lcrypto
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out  bn_sample.c  task4  task4.c  task5  task5.c  task6  task6.c  test  test.c
[03/25/20]seed@VM:~/.../Lab4$ ./task6
Message Signature =  5AFD6EC748FDD1FA9E8428D91A7B59AAD35BEF3D3DE5FA247754359B8CEAF527
[03/25/20]seed@VM:~/.../Lab4$
```

	Original Message	Hex	Message Signature
Message 1	I owe you $2000	49206f776520796f75202432303030	2C595D23711B74724D86428BF314D89D8B7BEB6FCF6B35D29FE08A7AB5DF7CA0
Message 2	I owe you $2001	49206f776520796f75202432303031	5AFD6EC748FDD1FA9E8428D91A7B59AAD35BEF3D3DE5FA247754359B8CEAF527

Comparing the signatures of the first message and second message, we see that with one different byte in the message produces a very signature.

Task 7: Verifying a signature

As done previously, we get the hex of the message “Launch a missile.” from python.

```
[GCC 5.4.0 20160609] on linux2
Type "help", "copyright", "credits" or "license" for mo
re information.
>>> "Launch a missile.".encode("hex")
'4c61756e63682061206d697373696c652e'
>>>
[13]+  Stopped                 python
[03/25/20]seed@VM:~/.../Lab4$ ls
a.out  bn_sample.c  task4  task4.c  task5  task5.c  tas
k6  task6.c  task7  task7.c  test  test.c
[03/25/20]seed@VM:~/.../Lab4$ gcc -o task7 task7.c -lcr
ypto
[03/25/20]seed@VM:~/.../Lab4$ ./task7
Original Message in hex   =  4C61756E63682061206D697373
696C652E
Computed Value of Message =  4C61756E63682061206D697373
696C652E
Alice's signature.
[03/25/20]seed@VM:~/.../Lab4$
```

By changing the signature, we get:

```
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 mo
re information.
>>> "Launch missile."encode("hex")
  File "<stdin>", line 1
    "Launch missile."encode("hex")
                     ^
SyntaxError: invalid syntax
>>> "Launch missile.".encode("hex")
'4c61756e6368206d697373696c652e'
>>>
[14]+  Stopped                 python
[03/25/20]seed@VM:~/.../Lab4$ gcc -o task7 task7.c -lcr
ypto
[03/25/20]seed@VM:~/.../Lab4$ ./task7
Original Message in hex   =  4C61756E6368206D697373696C
652E
Computed Value of Message =  4C61756E63682061206D697373
696C652E
Not Alice's signature.
[03/25/20]seed@VM:~/.../Lab4$
```

In this script, once we get the signature of message Message1, we use `BN_cmp` in order to compare the Message1 with the original, to check if the signature is that of Alice or not.

task7.c (~/Desktop/Lab4) - gedit

Open

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

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 *Message = BN_new(); // Message
    BIGNUM *n = BN_new();
    BIGNUM *S = BN_new();
    BIGNUM *e = BN_new();
    BIGNUM *Message1 = BN_new();

    // Initializing
    //BN_hex2bn(&Message, "4c61756e63682061206d697373696c652e"); // hex of "Launch a missile"
    BN_hex2bn(&Message, "4c61756e6368206d697373696c652e"); // hex of "Launch missle"
    BN_hex2bn(&S, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
    BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
    BN_hex2bn(&e, "010001");

    printBN("Original Message in hex   = ", Message);

    // Decrypting (Formula: C^d mod n)
    BN_mod_exp(Message1, S, e, n, ctx);
    printBN("Computed Value of Message = ", Message1);

    if (BN_cmp(Message1, Message) == 0 )
        printf("Alice's signature.\n");
    else
        printf("Not Alice's signature.\n");

    return 0;
}
```

Task 8: Manually verifying an X.509 Certificate

I couldn't get the server's certificate in this task. I kept receiving the "gethostbyname" failure. I tried everything I could find from the documentation of openssl online.

Terminal

```
[03/25/20]seed@VM:~$ openssl s_client -connect www.google.com:443 -showcerts
gethostbyname failure
connect:errno=11
[03/25/20]seed@VM:~$ openssl s_client -host www.google.com -port 443 -showcerts
gethostbyname failure
connect:errno=11
[03/25/20]seed@VM:~$ openssl s_client -host www.ryerson.ca -port 443 -showcerts
gethostbyname failure
connect:errno=11
```

From the documentation I could find online, the following is what I would've attempted to do if I could retrieve any server's certificate.

In some cases, there were 2 certificates that were retrieved.

1. Save the certificates in two separate files: cert1.pem and cert2.pem
2. Use `openssl x509 -in cert1.pem -noout -modulus` to get the value of 'n' (modulus)

3. Use `openssl x509 -in cert1.pem -text -noout t`o get the value of ‘e’ (exponent)
4. Use `openssl x509 -in cert2.pem -text -noout t`o get the signature, which we can save in a file and then remove all the punctuations and spaces from it.
5. `openssl asnlparse -I in cert1.pem` extracts the body of the certificate.
6. Then we can verify it using what we did in task 7. Comparing the hash value of the computed message and the that of the original should be the same if the certificate is verified.