SEEDLab


```
Nguyen Truong Lam - 19125101
Bui Quang Huy - 19125047
```


Instruction

Run:

```
bash run.bash
```

Report

Overview Result of 6 tasks

```
[07/04/22]seed@VM:~/Lab$ bash run.bash
[Task 1] The private key is 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB

[Task 2] encypted text:  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
[Task 2] decypted text:  4120746F702073656372657421
[Task 2] Decypted text == Original text

[Task 3] decypted value:  50617373776F72642069732064656573
[Task 3] raw text: Password is dees

[Task 4] signed value:  0B327F9EF80760C3136DB55C90E963429290E31ECC7D3975398263269F2965C8
[Task 4] unsigned value:  49206F776520796F7520323530302E
[Task 4] Origin text: I owe you 2500.

[Task 5] extracted sign: Launch a missile.
[Task 5] corrupted text:
@@,O@c@@rm=f@:N@@@@@

[Task 6] private key:  7061DF0A50B8F2BA3367ECFABAB273A16F3BB1378DBE1FE524E6DFD90DFA3B91
Private key extracted from certificate:
7061df0a50b8f2ba3367ecfabab273a16f3bb1378dbe1fe524e6dfd90dfa3b91  c0_body.bin
[07/04/22]seed@VM:~/Lab$
```

Task 1

First of all, we initialize 3 variables of p,q and e with the values given in the problem

```
BIGNUM *p = BN_new();
BIGNUM *q = BN_new();
BIGNUM *e = BN_new();

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

The next step is to calculate phi[n] (n = p*q) through the function with p and q are 2 big prime numbers
=> *phi[n] = (p - 1)*(q - 1)*

```
BIGNUM* getPhiOf(BIGNUM* p, BIGNUM* q) {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM* p_minus_one = BN_new();
    BIGNUM* q_minus_one = BN_new();
    BIGNUM* one = BN_new();
    BIGNUM* phi = BN_new();

        BN_dec2bn(&one, "1");
        BN_sub(p_minus_one, p, one);
        BN_sub(q_minus_one, q, one);
        BN_mul(phi, p_minus_one, q_minus_one, ctx);

        BN_CTX_free(ctx);

        return phi;

}
```

Finally, we have the theory:

e*d = 1 (mod phi[n]) => d is inverse modulo of e with mod phi[n];

So, the private of key *d* this problem is the inverse modulo of the big prime number *e* mod *phi[n]* computed by the following function using the API ***BN_mod_inverse***

```
BIGNUM* getModuloInverseOf(BIGNUM* e, BIGNUM* mod) {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM* res = BN_new();
    BN_mod_inverse(res, e, mod, ctx);
    BN_CTX_free(ctx);
    return res;
}
```

The result of this problem is printed to the console as the following:

```
[Task 1] The private key is 3587A24598E5F2A21DB007D89D18CC50ABA5075BA19A33890FE7C28A9B496AEB
```

Task 2

In this task, we need 4 variables according to the requirment with the message is encoded:

```
    BIGNUM* n = BN_new();    // n = p * q
    BIGNUM* d  = BN_new();    //private key
    BIGNUM* message = BN_new();
    BIGNUM* e  = BN_new();    //public key

    BN_hex2bn(&message, "4120746f702073656372657421"); // after encoding "A top secret!" with the pytho
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5"); BN_hex2bn(&e,
    "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

Now, the message is encoded to hex string so what we need to do is to convert it to BIG NUM to achive the solution. We create a function in order to conduct this conversion as following:

```
BIGNUM* encryptRSA(BIGNUM* rawValue, BIGNUM* encryptKey, BIGNUM* n) {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM* cipherValue = BN_new();
    BN_mod_exp(cipherValue, rawValue, encryptKey, n, ctx);
    BN_CTX_free(ctx);
    return cipherValue;
}
```

The core idea of this function is the formula:

e(x) = (x ^ e) mod n

Which means we can achive the encryption through the public key (e,n) by computing the exponential modulo *(x ^ e) mod n* with:

- x: the raw value that need to be encrypted
- e: the encrypt key

We also create a decrypt function shares the common idea with the encrypt one with:

d(x) = (x ^ e) mod n

```
BIGNUM* decryptRSA(BIGNUM* cipherValue, BIGNUM* decryptKey, BIGNUM* n) {
    BN_CTX *ctx = BN_CTX_new();
    BIGNUM* rawValue = BN_new();
    BN_mod_exp(rawValue, cipherValue, decryptKey, n, ctx);
    BN_CTX_free(ctx);
    return rawValue;
}
```

With these 2 functions we are able to print the result of both encrypted message and the decryption of the result and make a comparision to make sure these are equal.

```
    BIGNUM* cipherValue = encryptRSA(message, e, n);


    printBN("[Task 2] encypted text: ", cipherValue);

    BIGNUM* rawValue = decryptRSA(cipherValue, d, n);

    printBN("[Task 2] decypted text: ", rawValue);

    if (BN_cmp(rawValue, message) == 0) {
        printf("[Task 2] Decypted text == Original text\n");
    }
```

Finally, the result is

```
[Task 2] encypted text:  6FB078DA550B2650832661E14F4F8D2CFAEF475A0DF3A75CACDC5DE5CFC5FADC
[Task 2] decypted text: 4120746F702073656372657421
[Task 2] Decypted text == Original text
```

Task 3

As the public/private keys used in this task are the same as the ones used in Task 2, we have the variables with the ciphertext given by the requirement

```
    BIGNUM *cipherValue;
    BIGNUM* d = BN_new();    // private key
    BIGNUM* e  = BN_new();   // public key
    BIGNUM* n  = BN_new();

    BN_hex2bn(&cipherValue, "8C0F971DF2F3672B28811407E2DABBE1DA0FEBBBDFC7DCB67396567EA1E2493F");
    BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
    BN_hex2bn(&e, "010001");
    BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

The decrypt function is created in task 2 so we result it in this task to convert the ciphertext to the hex string then use API *BN_bn2hex* to convert the hex string to ASCII string

```
    BIGNUM* rawValue = decryptRSA(cipherValue, d, n);

    printBN("[Task 3] decypted value: ", rawValue);

    printf("[Task 3] raw text: ");
    printHX(BN_bn2hex(rawValue));
```

The final result printed to the console:

```
[Task 3] decypted value:  50617373776F72642069732064656573
[Task 3] raw text: Password is dees
```

Task 4

To generate a signature, it is neccessary to have variables with the values of the public and private key

```
BIGNUM* d = BN_new();    // private key
BIGNUM* e  = BN_new();   // public key
BIGNUM* n  = BN_new();

BN_hex2bn(&n, "DCBFFE3E51F62E09CE7032E2677A78946A849DC4CDDE3A4D0CB81629242FB1A5");
BN_hex2bn(&e, "010001");
BN_hex2bn(&d, "74D806F9F3A62BAE331FFE3F0A68AFE35B3D2E4794148AACBC26AA381CD7D30D");
```

The next step is to encode the message to hex string

```
BIGNUM* rawValue = BN_new();
BN_hex2bn(&rawValue, "49206f776520796f7520323530302e"); // after encoding: I owe you 2500.
```

With the encoded message, we use the function encryptRSA to generate the signed message
The different point of generating signature compared to normal encryption is that generating signature use private key to encrypt and public key to decrypt while normal encryption use public key for encrypting and decrypting.
Which mean the sender just need to give the receiver the public key to extract the signed message, but this message must be encrypt by the private key. If not, the decryption with the public key will give the wrong answer.

```
BIGNUM* rawValue = BN_new();
BN_hex2bn(&rawValue, "49206f776520796f7520323530302e"); // after encoding: I owe you 2500.

BIGNUM* signedValue = encryptRSA(rawValue, d, n); // sign: use private key to encypt
printBN("[Task 4] signed value: ", signedValue);
BIGNUM* unsignedValue = decryptRSA(signedValue, e, n); // extract signed value: use public key to d

printBN("[Task 4] unsigned value: ", unsignedValue);

printf("[Task 4] Origin text: ");
printHX(BN_bn2hex(unsignedValue));
```

Here we use the encoded value to generate signature and use public key to decrypt this value combined with API *BN_bn2hex* to archive the original message:

```
[Task 4] signed value:  0B327F9EF80760C3136DB55C90E963429290E31ECC7D3975398263269F2965C8
[Task 4] unsigned value:  49206F776520796F7520323530302E
[Task 4] Origin text: I owe you 2500.
```

Task 5

First of all, we assign the raw value and the public key to variables:

```
BIGNUM* rawValue = BN_new();
BIGNUM* signedValue = BN_new();
BIGNUM* e = BN_new();
BIGNUM* n = BN_new();
BN_hex2bn(&rawValue, "4c61756e63682061206d6973736c652e");    // after encoding: Launch a missile.
BN_hex2bn(&n, "AE1CD4DC432798D933779FBD46C6E1247F0CF1233595113AA51B450F18116115");
BN_hex2bn(&e, "010001");
BN_hex2bn(&signedValue, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6802F");
```

Then we use the public key and the decrypt RSA method with the function created in the previous task to verify the sign:

```
BIGNUM* decryptedValue = decryptRSA(signedValue, e, n);
printf("[Task 5] extracted sign: ");
printHX(BN_bn2hex(decryptedValue));
```

And we have the decrypted value:

```
[Task 5] extracted sign: Launch a missile.
```

Through observation, the extracted sign is similar to the original message Alice sent to Bob => the signature is Alice's.

So, now we change the last 2 byte of the signed value from 2F to 3F then repeat the verify process.

```
BN_hex2bn(&signedValue, "643D6F34902D9C7EC90CB0B2BCA36C47FA37165C0005CAB026C0542CBDB6803F");
BIGNUM* redecryptedValue = decryptRSA(signedValue, e, n);
printf("[Task 5] corrupted text:\n");
printHX(BN_bn2hex(redecryptedValue)); // corrupted value
```

We receive a corrupted value as following:

```
[Task 5] corrupted text:
��,O�c��rm=f�:N������
```

So, it's not the message Alice sent to Bob according to the modification of the signed value.

In conclusion, unless the message is correctly signed with the sender's private key, it will be impossible to extract the right message through the verification process.

Task 6

After following the steps in the requirement to download the certificate, we need to extract the public key and the signed message from the downloaded certificate.
We get the public key by:

- **n**: run '*extract in openssl x509 -in c1.pem -text -noout*'
- **e**: receive from the certificate x509
- **signedValue**: run '*extract in openssl x509 -in c0.pem -text -noout*'

```
BN_CTX *ctx = BN_CTX_new();

BIGNUM* e = BN_new();
BIGNUM* d = BN_new();
BIGNUM* n = BN_new();
BIGNUM* signedValue = BN_new();

// extract in openssl x509 -in c1.pem -text -noout
BN_hex2bn(&n, "00c14bb3654770bcdd4f58dbec9cedc366e51f311354ad4a66461f2c0aec6407e52edcdcb90a20eddfe3
BN_hex2bn(&e, "010001");
// extract in openssl x509 -in c0.pem -text -noout
BN_hex2bn(&signedValue, "aa9fbe5d911bade44e4ecc8f07644435b4ad3b133fc129d8b4abf3425149463bd6cf1e4183
```

Then, we use our decryptRSA function to extract the signedValue variable to archive the decrypted message then mod 2^256 to have the private key. Finally, we compare that key with the key extracted from the certificate archived by running '*bash key.bash*'

```
BIGNUM* decryptedValue = decryptRSA(signedValue, e, n);
BIGNUM* mod = BN_new();
BIGNUM* num_2 = BN_new();
BIGNUM* num_256 = BN_new();
BN_dec2bn(&num_2, "2");
BN_dec2bn(&num_256, "256");

BN_exp(mod, num_2, num_256, ctx);
BN_mod(d, decryptedValue,  mod, ctx);
printBN("[Task 6] private  key: ", d);   // then compare with private key after run key.bash
```

The final result:

```
[Task 6] private key:  7061DF0A50B8F2BA3367ECFABAB273A16F3BB1378DBE1FE524E6DFD90DFA3B91
Private key extracted from certificate:
7061df0a50b8f2ba3367ecfabab273a16f3bb1378dbe1fe524e6dfd90dfa3b91    c0_body.bin
```