Modern Cryptography:
Project Report - OpenSSL Application

Xiaoqi LIU, liu09335@gtiit.edu.cn

Project Description:

We need to write a program that receives as input:

1) The **indication** of a public key encryption scheme (rsa, dsa, ecc),
2) the **public key** in a file of a corresponding receiver,
3) the **private key** of the sender (transmitter),
4) an **indication** of a symmetric encryption scheme using keys of appropriate length, and
5) a **file** to be encrypted, signed and transmitted.

Then the program:

1) generates **randomly a key** of appropriate length for the symmetric encryption scheme,
2) **encrypts key** with the specified public key in the specified public key scheme, and puts this ciphertext in **base64 in a file, symm.key**,
3) builts a tar-ball, say **envelope.env** containing symm.key and the message to be transmitted,
4) **encrypts envelope.env** using the specified symmetric encryption scheme and the generated key key, and puts the ciphertext in a file cipher.enc,

The result of this program is the ciphertext cipher.enc. Also Write the corresponding **deciphering program**.

Table of Contents

1. Configuring OpenSSL Environment

The version I used is Openssl-3.2.1.The environment is Linux - Ubantu20.
After downloading the package and unpacked the files, I followed the instructions of compilation from the OpenSSL website. I used the following commands:

```
./config
make (Spend 10 minutes)
sudo make install (Requires permission to go through some directories)
sudo ldconfig /usr/local/lib64/ (openssl: error while loading shared libraries: libssl.so.3)
```

After several minutes of configuring and fixed some bugs, I start to looking through the tutorial and examples of the C++ API and coding.:

2. Deprecated Low-level Functions

I found some examples and tried to do the same with them. But when I compiled, the compiler reported warnings that the functions I used was deprecated since openssl 3.0, for examples: RSA_new(), etc.
Then I went to the internet and found 2 solutions, the first one is add some sentence in the command line to forbid the compiler to report this kind of warning. But the team of openssl posted on the website to recommend us to use the new versions of functions in the package EVP, since they believe the previous functions are not stable any more.
So I choose the second solution, to study how to use the latest version of functions.

3. Generate Key Pairs

The project require us to write a encryption program that accepts public key and private key of specific public-key scheme as inputs. So we need to write a program that can generate a random key pairs first (public and private keys). The specific public-key scheme I chose is RSA.

Codes:

```
EVP_PKEY* pkey = NULL;
EVP_PKEY_CTX* ctx = EVP_PKEY_CTX_new_id(EVP_PKEY_RSA, NULL);
EVP_PKEY_keygen_init(ctx);

// length of the desired keys, and generate keys
EVP_PKEY_CTX_set_rsa_keygen_bits(ctx, 2048);
EVP_PKEY_keygen(ctx, &pkey);
EVP_PKEY_CTX_free(ctx);
```

```
// write keys
BIO* pub_bio = BIO_new_file("public_key.pem", "wb");
BIO* priv_bio = BIO_new_file("private_key.pem", "wb");
PEM_write_bio_PUBKEY(pub_bio, pkey);
PEM_write_bio_PrivateKey(priv_bio, pkey, NULL, NULL, 0, NULL,
NULL);
BIO_free(pub_bio);
BIO_free(priv_bio);
```

Description:

1) Firstly, I used "EVP_PKEY" to initialize a key pairs, then I used "EVP_PKEY_keygen()" to generate a random key pairs.
2) I used "BIO" and "PEM_write _bio_PUBKEY()" and "PEM_write_bio_PrivateKey()" to write the random key pairs into "public_key.pem" and "private_key.pem" specifically.
3) The function is written in "generate_keys.cpp", in order to run it, you need to compile:
   `g++ -o generate_keys generate_keys.cpp -lssl -lcrypto`
4) I also write a function "show_key_rsa()" inside the file, thus when you run it by "./generate_keys", you will also see the content of the generated files.

Outputs:

```
xiaoqi@xiaoqi:~/Desktop/Chebyshev/project2$ ./generate_keys
Read public_key.pem:
-----BEGIN PUBLIC KEY-----
MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEA0i8EgU/oBVhmUvXYPuTd
EMoIEl+hYrQMghRlWE+1Y4IQ2PNogIbdQV8Y44Lg5fk72Ff/js9QMGs5YMvhEBZP
EaJNGhKy7BDI/sXz3gg7srLeH1XymwIGg+7ZCJEyHaPVrNKbrHpcY8HLCpRhRrg0
QudKA35kxKXXyqmmn8JhOi9zbVDGuPezqW3uOMwNVx1O/j7Yta7LPbQPDh+RQDKm
frkFnsckfDBYekXk4yxk0kxi+kUj3x+T88mEx4zOTO4ypEgTFEjLw1yiM7aHtQ1z
vHNjW3hIiiimfq/QCFk7KjTkNB6BcqHg8x4ktFVpx/nlcMZZxj5pvHzraHGMK30O
zQIDAQAB
-----END PUBLIC KEY-----

Read public_key.pem:
-----BEGIN PRIVATE KEY-----
MIIEvQIBADANBgkqhkiG9w0BAQEFAASCBKcwggSjAgEAAoIBAQDSLwSBT+gFWGZS
9dg+5N0QyggSX6FitAyCFGVYT7VjghDY82iAht1BXxjjguDl+TvYV/+Oz1Awazlg
y+EQFk8Rok0aErLsEMj+xfPeCDuyst4fVfKbAgaD7tkIkTIdo9Ws0puselxjwcsK
lGFGuDRC50oDfmTEpdfKqaafwmE6L3NtUMa497Opbe44zA1XHU7+Pti1rss9tA8O
H5FAMqZ+uQWexyR8MFh6ReTjLGTSTGL6RSPfH5PzyYTHjM5M7jKkSBMUSMvDXKIz
toe1DXO8c2NbeEiKKKZ+r9AIWTsqNOQ0HoFyoeDzHiS0VWnH+eVwxlnGPmm8fOto
cYwrfQ7NAgMBAAECggEAERFRWoDXH1aqLygYkHmRkV3z4YNo0wo6OSFCiN/ReSGk
```

```
Oux/0Bljg9NJmEarUEbOEHI2O0NkHcfEkogryrHNEj+tkXry6DmFYNmemn6XqRoI
8Il7AxGwiWvOraPramhrSbaWfp83ERYOZgReO9m/IK8FyG/Ylfa+UGOODMts4sZh
oKLFcdSgNO02u+xwWzEiqhWTPiCwAJtgyEmTQnWbdpDSV3Ag0LlVCsK5htd0rpze
5CRVUAxwuuCZp2Ugv3DseHwCodw3rDR/QfxJ/g1JzUZa2WSP4sTAJ3bJ2vz6sAGr
q8RZ2bbUb/ThcF7HJHDxu2dy3kDWZKhfyJP3l3pcEQKBgQDvmYCtYNEXBnBG4USs
ILsRaGTHtcs4Kji93M5EDYoBya6ZtyyDEKUb/4zVaCjnknJX4siRUHRXyPHZzjaT
4K4Cjkc5dX6u/pJN88Cpbcc1bD+KvZ/San7fLXV+dsqjG1VzR+4XFOG4unjmovdp
zsXRNAJPNUC4D4R1YLoaI/2pJQKBgQDgkg9Z/agEYLtVwFrEH5lZrOXkZ9o8c4Cj
UFPI3vLa6Wl2f8P80PantMVw/dk8rMWlrCX865njZIbogcLvjdnX6IGR7Yri74vg
UAyCmbHTj2EU7T65fC0W5QNihRUA3FrNdg2np73TJu+ftNI74lV+qoegPUnkLUIi
fggbcfpCiQKBgQClr/kT7EerLICoA0+B2ICEYTSQBl8pz8i9QtxAozw0UEu1cL4P
RIto+SdizEy9FsCLlgAsvP43VnYosDl9uXmbutf2HBIm4VQ7I7yvbYK8psaqIsdK
supZWw5NDcjDUjudJLi4rXAKVqkUlM2TFbImIVkXfTPsFI78vI71VL4tIQKBgBr9
0zSIxVoJWlrbLM2n0qGxo0wJo3RimZKHhBp/mYbwSQnCH4dSgmLz2ktTZ3ngdj0M
bOByXOO04HYvqoNqdEV8H56vg2L1UcWbOXW1DWeVMyD1xvU6VJ0M15RO1JBu68cI
Kmn9QlaPtntmm2IxOP9pdbWvyCLRwcItjcHmuCKpAoGAFJ1sXHd2SAF4/zihGcLH
0csGfngfp3vW0OfqT1px+LcokCkD95T4rqTP4QLqGDE+2nTZNpZPSelq5WaG3yYq
bYG4/9rKtk4yh++vcrLvUtGWOpv/qCt0DtZYUXMjngfGZSgDeA0QuTa8B8wcnGj6
NW6q78V6S0Id/XAIJx/vXYk=
-----END PRIVATE KEY-----
```

3. Encryption - Part 1: Encrypt key - symm.key

The program asks us to generate randomly symmetric key *key* of appropriate length for the symmetric encryption scheme. And encrypts *key* with the specified public-key scheme, and puts this ciphertext in base64 in a file, *symm.key*.

Codes:

```
// ---------- Encrypt key using RSA -----------------
// read keys
BIO* pub_bio = BIO_new_file(pub_key_file, "r");
BIO* priv_bio = BIO_new_file(priv_key_file, "r");
EVP_PKEY *pub_key = PEM_read_bio_PUBKEY(pub_bio, NULL, NULL, NULL);
EVP_PKEY *priv_key = PEM_read_bio_PrivateKey(priv_bio, NULL, NULL,
NULL);
BIO_free(pub_bio);
BIO_free(priv_bio);

// generate random symmetric keys
unsigned char key[AES_KEYLEN/8];
unsigned char iv[AES_KEYLEN/8];
RAND_bytes(key, sizeof(key));
```

```
RAND_bytes(iv, sizeof(iv));
cout << "Random symmetric key (AES256): " << endl;
BIO_dump_fp(stdout, key, sizeof(key));
cout << "Random initialization vector: " << endl;
BIO_dump_fp(stdout, iv, sizeof(iv));

// encrypt symmetric key using public key scheme
EVP_PKEY_CTX* ctx = EVP_PKEY_CTX_new(pub_key, NULL);
EVP_PKEY_encrypt_init(ctx);
EVP_PKEY_CTX_set_rsa_padding(ctx, RSA_PKCS1_PADDING);
EVP_PKEY_encrypt(ctx,     cipher_key,     &cipher_key_len,     key,
sizeof(key));
EVP_PKEY_CTX_free(ctx);
cout << endl << "Ciphert key using RSA is: " << endl;
BIO_dump_fp(stdout, cipher_key, cipher_key_len);

// ---------- Write Encrypted Symmetric Key -----------------
// write cipher_key - pem_write will be base64 format
BIO* cipher_key_bio = BIO_new_file("symm.key", "w");
PEM_write_bio(cipher_key_bio,  "DECRYPTED  SYMMETRIC  KEY",  "",
cipher_key, cipher_key_len);
BIO_free(cipher_key_bio);
```

Description:

1) I used “BIO” and “PEM_read_bio_PUBKEY()” and “PEM_read_bio_PrivateKey()” to read the stored key pairs from the Program 1.
2) I used “RAND_bytes()” to assign random bits into my *key* and *iv* buffer. Also note that, since I want to use AES256 symmetric scheme, the corresponding AES_KEYLEN is 32.
3) Then I used “EVP_PKEY_CTX_rsa_padding()” and “EVP_PKEY_encrypt()” to encrypt the *key* using RSA scheme, and showed the raw results by “BIO_dump_fp()”.
4) Then, I used “BIO” and “PEM_write_bio()” to write the *ciphered key* in raw format into a file *symm.key* with base64 format.
5) The function is written in file “rsa_aes.cpp”, in order to run it, please compile with:

```
g++ -o rsa_aes rsa_aes.cpp -lssl -lcrypto
```

6) As the project required, the program need to receive 5 arguments as inputs, so in order to run it, you need following the command:

```
Usage: ./rsa_aes <public-key scheme> <pub_key> <priv_key> <symmetric
scheme> <plaintext file>
For  example:  ./rsa_aes  rsa  public_key.pem  private_key.pem  ase
plaintext.txt
```

Outputs:

```
Random symmetric key (AES256):
0000 - 42 91 71 0a                                       B.q.
Random initialization vector:
0000 - ae 23 da 13                                       .#..

Ciphert key using RSA is:
0000 - 7e 16 1f f5 dc e2 36 d6-94 41 2c 0f 5f 8b 3c 08   ~.....6..A,._.<.
0010 - 03 d3 48 28 62 f5 fe 07-fe 78 65 d6 4f 6d 94 18   ..H(b....xe.Om..
0020 - 0b 8c 0f 9d 12 f1 28 64-5c a9 f5 bf 10 19 ce 51   ......(d\......Q
0030 - 14 c4 24 e0 ee 50 ad a9-c0 e9 90 a0 2d e4 91 89   ..$..P......-...
0040 - f8 0d a8 64 fa d2 13 21-bd 27 16 fe 70 81 f0 11   ...d...!.'..p...
0050 - a9 8e b1 30 ab 70 f8 b8-79 3c a3 c8 44 0d 1a 2f   ...0.p..y<..D../
0060 - fc 57 24 a5 3e dc 56 89-e0 ab a3 d6 d6 f4 44 7c   .W$.>.V.......D|
0070 - 52 36 b2 31 59 99 db 66-a9 7d 5e 54 4e a6 fc 1a   R6.1Y..f.}^TN...
0080 - 62 87 80 99 9b 48 45 90-a7 4d 68 18 9f 5a de c7   b....HE..Mh..Z..
0090 - e1 c2 66 f9 71 4c f7 03-13 22 be fb 59 c6 66 d2   ..f.qL..."..Y.f.
00a0 - 4d 5b e8 35 af 62 18 26-92 60 23 5c 9d 76 01 88   M[.5.b.&.`#\.v..
00b0 - a7 bd c4 f3 89 36 ed c6-3f 20 86 69 15 0c d1 b6    .....6..? .i....
00c0 - 8d 0e b4 bc 05 09 30 0d-9a 30 24 d7 a3 94 02 41   ......0..0$....A
00d0 - 35 4d fe 72 dc d9 aa 18-d8 15 23 77 20 6e bd 9e   5M.r......#w
n..
00e0 - 78 0e 94 87 46 a5 af 16-48 5b 8f 81 1e 22 fa 2f   x...F...H[..."./
00f0 - 8d 14 a2 a7 66 27 19 72-fa d0 b3 96 9c 59 71 91   ....f'.r.....Yq.

Read symm.key:
-----BEGIN DECRYPTED SYMMETRIC KEY-----
fhYf9dziNtaUQSwPX4s8CAPTSChi9f4H/nhl1k9tlBgLjA+dEvEoZFyp9b8QGc5R
FMQk4O5QranA6ZCgLeSRifgNqGT60hMhvScW/nCB8BGpjrEwq3D4uHk8o8hEDRov
/FckpT7cVongq6PW1vREfFI2sjFZmdtmqX1eVE6m/Bpih4CZm0hFkKdNaBifWt7H
4cJm+XFM9wMTIr77WcZm0k1b6DWvYhgmkmAjXJ12AYinvcTziTbtxj8ghmkVDNG2
jQ60vAUJMA2aMCTXo5QCQTVN/nLc2aoY2BUjdyBuvZ54DpSHRqWvFkhbj4EeIvov
jRSip2YnGXL60LOWnFlxkQ==
-----END DECRYPTED SYMMETRIC KEY-----
```

4. Encryption - Part 2: Built a tar-ball - envelope.env

The project asks us to built a tar-ball, say *envelope.enc* that containing *symm.key* and the *message* to be transmitted.

Codes:

```
// combine cipher_key and plaintext in a tar-ball
// read cipher_key in base64 format
ifstream file2 ("symm.key");
// must use a new stirng, or previous plaintext will be changed
string readfile2 = "";
tmp = "";
while (getline(file2, tmp)) {
    readfile2 += tmp;
    readfile2 += "\n";
}
file2.close();
const char* cipher_key_base64 = (const char *)readfile2.c_str();

// write cipher_key and plaintext into envelope.env
FILE* file3 = fopen("envelope.env", "w");
fprintf(file3, "%s", cipher_key_base64);
fprintf(file3, "%s", "-----BEGIN MESSAGE-----\n");
fprintf(file3, "%s", plaintext);
fprintf(file3, "%s", "-----END MESSAGE-----\n");
fclose(file3);

// read envelope_env
fstream file4 ("envelope.env");
string readfile3 = "";
tmp = "";
while (getline(file4, tmp)) {
    readfile3 += tmp;
    readfile3 += "\n";
}
file4.close();
const unsigned char* envelope_env = (const unsigned char
*)readfile3.c_str();
cout << endl << "Tar-ball envelope.env is: " << endl << envelope_env
<< endl;
size_t envelope_env_len = readfile3.length();
```

Description:

The idea is very simple, just use the file operations in C++ to read the *plaint.txt* and *symm.key*, then combine them together in a new file *envelope.env*. Just noted that I used 2 file structures, one is FILE with fprintf(), the other is fstream with getline().

Outputs:

```
Tar-ball envelope.env is:
-----BEGIN DECRYPTED SYMMETRIC KEY-----
fhYf9dziNtaUQSwPX4s8CAPTSChi9f4H/nhl1k9tlBgLjA+dEvEoZFyp9b8QGc5R
FMQk4O5QranA6ZCgLeSRifgNqGT60hMhvScW/nCB8BGpjrEwq3D4uHk8o8hEDRov
/FckpT7cVongq6PW1vREfFI2sjFZmdtmqX1eVE6m/Bpih4CZm0hFkKdNaBifWt7H
4cJm+XFM9wMTIr77WcZm0k1b6DWvYhgmkmAjXJ12AYinvcTziTbtxj8ghmkVDNG2
jQ60vAUJMA2aMCTXo5QCQTVN/nLc2aoY2BUjdyBuvZ54DpSHRqWvFkhbj4EeIvov
jRSip2YnGXL60LOWnFlxkQ==
-----END DECRYPTED SYMMETRIC KEY-----
-----BEGIN MESSAGE-----
Hi! I'm Xiaoqi.
Nice to meet you! ^_^
-----END MESSAGE-----
```

5. Encryption - Part 3: Encrypt envelope.env using AES.

The project requires us to encrypt *envelope.env* using the specified symmetric encryption scheme and the generated symmetric key *key*, and put the *ciphertext* of the *envelope.env* in a file *cipher.enc*.

Codes:

```
unsigned char cipher_enc[1024];
size_t cipher_enc_len;
int len;
EVP_CIPHER_CTX *ctx2;
ctx2 = EVP_CIPHER_CTX_new();

EVP_EncryptInit_ex(ctx2, EVP_aes_256_cbc(), NULL, key, iv);
EVP_EncryptUpdate(ctx2,    cipher_enc,    &len,    envelope_env,
envelope_env_len);
cipher_enc_len = len;
EVP_EncryptFinal_ex(ctx2, cipher_enc + len, &len);
cipher_enc_len += len;

EVP_CIPHER_CTX_free(ctx2);
cout << "After AES_256_cbc symmetric encryption of envelope: " << endl;
BIO_dump_fp(stdout, cipher_enc, cipher_enc_len);

// write cipher.enc
BIO* cipher_enc_bio = BIO_new_file("cipher.enc", "w");
```

```
PEM_write_bio(cipher_enc_bio,   "CIPHER   ENC",   "",   cipher_enc,
cipher_enc_len);
BIO_free(cipher_enc_bio);

// read cipher.enc
fstream file5 ("cipher.enc");
string readfile4 = "";
tmp = "";
while (getline(file5, tmp)) {
    readfile4 += tmp;
    readfile4 += "\n";
}
file5.close();
cout << endl << "Read cipher.enc: " << endl << readfile4 << endl;
```

Description:

1) I initialed a ciphertext buffer cipher_enc[1024] and ciphertext length cipher_enc_len to prepare as an input of the encryption functions.
2) I used “EVP_EncryptInit_ex()”, “EVP_EncryptUpdate()”, “EVP_EncryptFinal_ex()” to encrypt *envelope.env* using AES256 symmetric scheme. During the procedures, I also get the ciphertext length.
3) I showed the raw format of the ciphertext using “BIO_dump_fp()”.
4) I used “BIO” and “PEM_write_bio()” to write the base64 format of the ciphertext in a file *cipher.enc*. I also showed the content of the file in the program.

Outputs:

```
After AES_256_cbc symmetric encryption of envelope:
0000 - b9 38 0f ad 31 ea bc a9-98 37 dc 53 82 3c 5b 2c   .8..1....7.S.<[,
0010 - fa c0 63 88 22 b5 d3 78-5f 23 dc 12 13 7c cb f0   ..c."..x_#...|..
0020 - 36 0a ca 2b 6c 35 12 c1-53 1a bd 87 48 fb 91 53   6..+l5..S...H..S
0030 - d7 c2 cb b2 43 9f dd ee-47 a2 ae 54 48 08 f6 4e   ....C...G..TH..N
0040 - f0 0a 2a bf f3 5b 19 75-12 94 c7 7a 79 6e 16 ba   ..*..[.u...zyn..
0050 - d0 d2 ae cf 37 4d aa 0e-8e 07 5f ef 65 61 01 6f   ....7M...._.ea.o
0060 - 50 36 05 b1 67 b5 9c dd-e5 dd 95 c7 c0 1b d0 e9   P6..g...........
0070 - d2 fd a5 34 18 8b 86 92-3e 6c 1b ad 39 8e b4 76   ...4....>l..9..v
0080 - d3 24 1c 5b db 1f 34 63-99 9b 2c 82 e3 be d4 69   .$.[..4c..,....i
0090 - 48 07 c5 66 8e 2c 63 40-9c c3 a2 d2 1c f4 fb b9   H..f.,c@........
00a0 - 92 0f 6b c0 9d c6 fa 73-dd fb f2 9c aa 4d be 92   ..k....s.....M..
00b0 - 6e ce 5f 20 f9 87 b1 de-94 f9 06 dd bc 51 7b 12   n._ .........Q{.
00c0 - 1d 96 85 a1 8b 4e cf c2-e5 99 ed 2c b0 77 13 45   .....N.....,.w.E
00d0 - 36 98 7c 97 b2 69 17 09-81 c7 94 c8 53 b7 75 f9   6.|..i......S.u.
00e0 - f0 ad 45 78 a9 93 fa 54-99 46 ce 1d 12 0d 9e 1c   ..Ex...T.F......
```

```
00f0 - ac c5 08 c4 1f b9 7f d4-04 4f 4c d4 59 cf 7b 9f   .........OL.Y.{.
0100 - d1 44 bc 21 71 93 02 6e-42 04 cd 53 35 3f 0a c6   .D.!q..nB..S5?..
0110 - da 9c 8d a1 58 a0 7b 0c-20 80 17 45 cd 6a c4 36   ....X.{. ..E.j.6
0120 - a8 86 a8 cd fc 60 f2 e9-f5 07 14 a2 50 93 02 14   .....`......P...
0130 - b9 70 b8 72 64 88 ef 74-55 46 1f de 92 09 6c 42   .p.rd..tUF....lB
0140 - 3e 46 b0 2c e4 31 25 a1-ab 7f 3d 03 ab d9 20 ed   >F.,.1%...=... .
0150 - de 60 56 5f 8d 1b 99 34-9e 1e c9 a4 09 da 4d 67   .`V_...4......Mg
0160 - 29 9c 82 88 87 23 d3 5e-6a 8e ec da 1d 34 b9 1c   ).....#.^j....4..
0170 - 0a f7 ba 32 9b 01 33 c6-46 3f 84 e5 bc 52 f9 bc   ...2..3.F?...R..
0180 - 14 74 7d 39 77 6f 23 d1-95 3a 97 68 a2 83 ab 6e   .t}9wo#..:.h...n
0190 - 6c 52 fe 7d 45 6a 2f 7d-78 e6 3d 6f 49 88 91 c7   lR.}Ej/}x.=oI...
01a0 - 93 27 80 14 0d 23 2a 98-c9 90 f6 80 bd e8 e2 5c   .'...#*........\
01b0 - 63 37 13 70 4d e3 a5 c4-43 cd 3d 0d d8 1a 20 3e   c7.pM...C.=... >
01c0 - bb 66 4f 2d 46 ed af 23-70 a0 97 b2 d9 bc 7d ca   .fO-F..#p.....}.
01d0 - e3 c5 a2 72 d5 9e 3c 1b-c5 a1 28 4e 97 db 69 f9   ...r..<...(N..i.
01e0 - e5 cb ac 64 0d 34 e5 44-51 19 03 10 60 3d a1 41   ...d.4.DQ...`=.A
01f0 - a2 e4 7b b5 1c 4b bb 99-17 78 45 b7 ec b9 b4 50   ..{..K...xE....P
0200 - e6 fe d9 48 65 ee 56 f4-6e 18 3b 72 b7 2b 69 c1   ...He.V.n.;r.+i.

Read cipher.enc:
-----BEGIN CIPHER ENC-----
uTgPrTHqvKmYN9xTgjxbLPrAY4gitdN4XyPcEhN8y/A2CsorbDUSwVMavYdI+5FT
18LLskOf3e5Hoq5USAj2TvAKKr/zWxl1EpTHenluFrrQ0q7PN02qDo4HX+9lYQFv
UDYFsWe1nN3l3ZXHwBvQ6dL9pTQYi4aSPmwbrTmOtHbTJBxb2x80Y5mbLILjvtRp
SAfFZo4sY0Ccw6LSHPT7uZIPa8Cdxvpz3fvynKpNvpJuzl8g+Yex3pT5Bt28UXsS
HZaFoYtOz8Llme0ssHcTRTaYfJeyaRcJgceUyFO3dfnwrUV4qZP6VJlGzh0SDZ4c
rMUIxB+5f9QET0zUWc97n9FEvCFxkwJuQgTNUzU/CsbanI2hWKB7DCCAF0XNasQ2
qIaozfxg8un1BxSiUJMCFLlwuHJkiO90VUYf3pIJbEI+RrAs5DEloat/PQOr2SDt
3mBWX40bmTSeHsmkCdpNZymcgoiHI9Neao7s2h00uRwK97oymwEzxkY/hOW8Uvm8
FHR9OXdvI9GVOpdoooOrbmxS/n1Fai99eOY9b0mIkceTJ4AUDSMqmMmQ9oC96OJc
YzcTcE3jpcRDzT0N2BogPrtmTy1G7a8jcKCXstm8fcrjxaJy1Z48G8WhKE6X22n5
5cusZA005URRGQMQYD2hQaLke7UcS7uZF3hFt+y5tFDm/tlIZe5W9G4YO3K3K2nB
-----END CIPHER ENC-----
```

6. Decryption - Part 1: Decrypt cipher.enc using AES.

Since the *cipher.enc* is encrypted from *envelop.env* using AES, we need to use inverse AES to decrypt it and verify whether it's the same with the plaintext *envelope.env*.

Codes:

```
unsigned char decryptedtext[1024];
```

```
size_t decrypted_len;
int len2 = 0;
EVP_CIPHER_CTX *ctx3;
ctx3 = EVP_CIPHER_CTX_new();

EVP_DecryptInit_ex(ctx3, EVP_aes_256_cbc(), NULL, key, iv);
EVP_DecryptUpdate(ctx3,    decryptedtext,    &len2,    cipher_enc,
cipher_enc_len);
decrypted_len = len2;
EVP_DecryptFinal_ex(ctx3, decryptedtext + len2, &len2);
decrypted_len += len2;

EVP_CIPHER_CTX_free(ctx3);
cout << endl << "Use AES to decrpt and verify: " << endl;
cout << decryptedtext << endl;
```

Description:

1) It is quite similar to the AES Encryption part, I initialed a decryptedtext buffer decryptedtext[1024] and decryptedtext length decrypted_len to prepare as an input of the decryption functions.
2) I used “EVP_DecryptInit_ex()”, “EVP_DecryptUpdate()”, “EVP_DecryptFinal_ex()” to decrypt *cipher.enc* using AES256 symmetric scheme. During the procedures, I also get the decryptedtext length.
3) Since the decryptedtext is in the format of base64, we can simply use “cout << decryptedtext << endl;” to show the result, and the result is the same with *envelope.env*.

Outputs:

```
Use AES to decrpt and verify:
-----BEGIN DECRYPTED SYMMETRIC KEY-----
Pj+0u7FdyBoPHnguyhhSpBKzvaNjUac05XOB4HbiXGy8OF0D+dqhqWeh2Zx6NijB
JBtuh7kZ2KIr3MECWU/HWj5JYHBH/RFre2yoAgK0URjT4CsJ3BwlrbW2XP2DuCX9
4OgLUzBHoLvUXiZvAihgRUXTWwg7xgZpnizwNEOYLazLuY3836uwtBE/mt+gFylW
a6nVALC8mWsqlS4VE7NWpULhFCLOEciKKk3vxcFQRboU8ACf3N2Yxher5dPZdj5l
C+PcHtdd2F5cQ5ZDcbUL7+IPEgqkWJ3oGiu0+ibm1TaZmUVCNkpckFIWJSbQHlYI
inPnaaf0BLst2W6iXQuvEg==
-----END DECRYPTED SYMMETRIC KEY-----
-----BEGIN MESSAGE-----
Hi! I'm Xiaoqi.
Nice to meet you! ^_^
-----END MESSAGE-----
```

7. Decryption - Part 2: Decrypt symm.key using RSA.

As you remember, the *symm.key* is encrypted from a randomly generated symmetric key using RSA public-key scheme. We need to decrypt the *symm.key* using private_key.pem and compare with the original randomly generated symmetric key.

Codes:

```
unsigned char decrypted_key[256];
size_t decrypted_key_len;

EVP_PKEY_CTX* ctx4 = EVP_PKEY_CTX_new(priv_key, NULL);
EVP_PKEY_decrypt_init(ctx4);
EVP_PKEY_decrypt(ctx4,   NULL,   &decrypted_key_len,   cipher_key,
cipher_key_len);

EVP_PKEY_CTX_set_rsa_padding(ctx4, RSA_PKCS1_PADDING);
EVP_PKEY_decrypt(ctx4,     decrypted_key,     &decrypted_key_len,
cipher_key, cipher_key_len);
decrypted_key[decrypted_key_len] = '\0';

EVP_PKEY_CTX_free(ctx4);
EVP_PKEY_free(pub_key);
EVP_PKEY_free(priv_key);

// comparision
cout << "Decrypted symmetric key using RSA is: " << endl;
BIO_dump_fp(stdout, decrypted_key, decrypted_key_len);
cout << "Original random symmetric key: " << endl;
BIO_dump_fp(stdout, key, sizeof(key));
```

Description:

1) It's also quite similar to the RSA encryption part. I used "EVP_PKEY_CTX_decrypt()", "EVP_PKEY_decrypt()", and "EVP_PKEY_set_rsa_padding()" to decrypt the *symm.key* using RSA scheme.
2) I showed the original random symmetric key and decrypted symmetric key in base64 format together, and we can see they're the same.

Outputs:

```
Decrypted symmetric key using RSA is:
0000 - d5 cb 6c 7f                                      ..l.
Original random symmetric key:
0000 - d5 cb 6c 7f                                      ..l.
```

8. Conclusion.

In conclusion, I have written 2 files: “generate_keys.cpp” to generate random key pairs and “rsa_aes.cpp” to manipulate all the operations about encryption and decryption that the project requires.

To start experiment, you need to prepare a plaintext.txt that includes the content you want to encrypt, and compile and run the “generate_key.cpp” to get a random public key “pubic_key.pem” and a random private key “private_key.pem”.

```
g++ -o generate_keys generate_keys.cpp -lssl -lcrypto
```

Then you can compile and run the main program “rsa_aes.cpp” to see the operations showing on the terminal.

```
g++ -o rsa_aes rsa_aes.cpp -lssl -lcrypto
./rsa_aes rsa public_key.pem private_key.pem ase plaintext.txt
```

The program will generate a random symmetric key *key*, and encrypt it using RSA and public key, and puts it in *symm.key*. Then it will built an *envelope.env* that contains *symm.key* and *plaintext*. Then, it will encrypt *envelope.env* using AES and *key* and puts it in *cipher.enc*. After that, it will decrypt the *cipher.enc* and compare with *envelop.env*, also decrypt *symm.key* and compare with *key*.

That’s the end of this report, thank you very much for your attention.

Author: Xiaoqi LIU, liu09335@gtiit.edu.cn