RSA Encryption & Decryption Example with OpenSSL in C

By Ravishanker Kusuma

Source: http://hayageek.com/rsa-encryption-decryption-openssl-c/

In this article, I have explained how to do RSA Encryption and Decryption with **OpenSSL** Library in C.

1).Generate RSA keys with OpenSSL → #generate
2).Public Encryption and Private Decryption → #public-encrypt
3).Private Encryption and Public Decryption. → #private-encrypt
4).Encryption and Decryption Example code. → #example

1).Generate RSA keys with OpenSSL

Use the below command to generate RSA keys with length of 2048.

```
openssl genrsa -out private.pem 2048
```

Extract public key from private.pem with the following command.

```
openssl rsa -in private.pem -outform PEM -pubout
-out public.pem
```

public.pem is RSA public key in PEM format.
private.pem is RSA private key in PEM format.

2).Public Encryption and Private Decryption

Below is the OpenSSL API for Public encryption and Private decryption.

```
 int RSA_public_encrypt(int flen, unsigned char
*from,
    unsigned char *to, RSA *rsa, int padding);

 int RSA_private_decrypt(int flen, unsigned char
*from,
     unsigned char *to, RSA *rsa, int padding);
```

2.1 Preparing RSA Structure

For encryption and decryption we need to prepare RSA structure. Use the below function to create RSA with key buffer.

```
RSA * createRSA(unsigned char * key,int public)
{
    RSA *rsa= NULL;
    BIO *keybio ;
    keybio = BIO_new_mem_buf(key, -1);
    if (keybio==NULL)
    {
        printf( "Failed to create key BIO");
        return 0;
    }
    if(public)
    {
            rsa =
PEM_read_bio_RSA_PUBKEY(keybio, &rsa,NULL,
NULL);
    }
    else
    {
            rsa =
PEM_read_bio_RSAPrivateKey(keybio, &rsa,NULL,
NULL);
    }

        return rsa;
}
```

Usage for public key: *createRSA("PUBLIC_KEY_BUFFER",1);*
Usage for private key: *createRSA("PRIVATE_KEY_BUFFER",0);*

If you want to create RSA with key file name, you can use this function

```
RSA * createRSAWithFilename(char * filename,int
public)
{
        FILE * fp = fopen(filename,"rb");

        if(fp == NULL)
        {
                printf("Unable to open file %s
\n",filename);
                return NULL;
        }
    RSA *rsa= RSA_new() ;
```

```
     if(public)
     {
               rsa = PEM_read_RSA_PUBKEY(fp,
&rsa,NULL, NULL);
     }
     else
     {
               rsa = PEM_read_RSAPrivateKey(fp,
&rsa,NULL, NULL);
     }

          return rsa;
}
```

2.1 Public Key Encryption.

For encryption we can use padding, below is the list of supported paddings.

RSA_PKCS1_PADDING

PKCS #1 v1.5 padding. This currently is the most widely used mode.

RSA_PKCS1_OAEP_PADDING

EME-OAEP as defined in PKCS #1 v2.0 with SHA-1, MGF1 and an empty encoding parameter. This mode is recommended for all new applications.

RSA_SSLV23_PADDING

PKCS #1 v1.5 padding with an SSL-specific modification that denotes that the server is SSL3 capable.

RSA_NO_PADDING

Raw RSA encryption. This mode should only be used to implement cryptographically sound padding modes in the application code. Encrypting user data directly with RSA is insecure.

You can use the below method, to encrypt the data with public key.

```
int padding = RSA_PKCS1_PADDING;

int public_encrypt(unsigned char * data,int
data_len,unsigned char * key, unsigned char
*encrypted)
{
        RSA * rsa = createRSA(key,1);
    int result =
RSA_public_encrypt(data_len,data,encrypted,rsa,p
adding);
    return result;
}
```

Note: *public key encryption supports all the paddings.*

2.2 Private Decryption.

You can use the below method to decrypt the data with private key

```
int private_decrypt(unsigned char * enc_data,int
data_len,unsigned char * key, unsigned char
*decrypted)
{
        RSA * rsa = createRSA(key,0);
        int  result =
RSA_private_decrypt(data_len,enc_data,decrypted,
rsa,padding);
    return result;
}
```

3).Private Key Encryption and Public Key Decryption.

Below is the OpenSSL API for private encryption and public decryption.

```
 int RSA_private_encrypt(int flen, unsigned char
*from,
    unsigned char *to, RSA *rsa, int padding);

 int RSA_public_decrypt(int flen, unsigned char
*from,
    unsigned char *to, RSA *rsa, int padding);
```

***Note**: private key encryption supports only these paddings.*
***RSA_PKCS1_PADDING** and **RSA_NO_PADDING**.*

3.1 Private Key Encryption.

You can use the below function for private key encryption.

```
int private_encrypt(unsigned char * data,int
data_len,unsigned char * key, unsigned char
*encrypted)
{
        RSA * rsa = createRSA(key,0);
    int result =
RSA_private_encrypt(data_len,data,encrypted,rsa,
padding);
    return result;
}
```

3.2 Public Key Decryption.

You can use the below function for public key decryption.

```
int public_decrypt(unsigned char * enc_data,int
data_len,unsigned char * key, unsigned char
*decrypted)
{
        RSA * rsa = createRSA(key,1);
        int  result =
RSA_public_decrypt(data_len,enc_data,decrypted,r
sa,padding);
    return result;
}
```

4) Encryption and Decryption Example code.

```
#include <openssl/pem.h>
#include <openssl/ssl.h>
#include <openssl/rsa.h>
#include <openssl/evp.h>
#include <openssl/bio.h>
#include <openssl/err.h>
#include <stdio.h>

int padding = RSA_PKCS1_PADDING;

RSA * createRSA(unsigned char * key,int public)
{
    RSA *rsa= NULL;
    BIO *keybio ;
    keybio = BIO_new_mem_buf(key, -1);
    if (keybio==NULL)
    {
        printf( "Failed to create key BIO");
        return 0;
    }
    if(public)
    {
            rsa =
PEM_read_bio_RSA_PUBKEY(keybio, &rsa,NULL,
NULL);
    }
    else
    {
            rsa =
PEM_read_bio_RSAPrivateKey(keybio, &rsa,NULL,
NULL);
    }
    if(rsa == NULL)
    {
        printf( "Failed to create RSA");
    }

        return rsa;
}

int public_encrypt(unsigned char * data,int
data_len,unsigned char * key, unsigned char
*encrypted)
{
        RSA * rsa = createRSA(key,1);
    int result =
RSA_public_encrypt(data_len,data,encrypted,rsa,p
adding);
    return result;
}
```

```
int private_decrypt(unsigned char * enc_data,int
data_len,unsigned char * key, unsigned char
*decrypted)
{
        RSA * rsa = createRSA(key,0);
        int  result =
RSA_private_decrypt(data_len,enc_data,decrypted,
rsa,padding);
    return result;
}


int private_encrypt(unsigned char * data,int
data_len,unsigned char * key, unsigned char
*encrypted)
{
        RSA * rsa = createRSA(key,0);
    int result =
RSA_private_encrypt(data_len,data,encrypted,rsa,
padding);
    return result;
}
int public_decrypt(unsigned char * enc_data,int
data_len,unsigned char * key, unsigned char
*decrypted)
{
        RSA * rsa = createRSA(key,1);
        int  result =
RSA_public_decrypt(data_len,enc_data,decrypted,r
sa,padding);
    return result;
}

void printLastError(char *msg)
```

```
{
        char * err = malloc(130);;
        ERR_load_crypto_strings();
        ERR_error_string(ERR_get_error(), err);
    printf("%s ERROR: %s\n",msg, err);
    free(err);
}

int main(){

  char plainText[2048/8] = "Hello this is Ravi";
//key length : 2048

 char publicKey[]="-----BEGIN PUBLIC KEY-----
\n"\
"MIIBIjANBgkqhkiG9w0BAQEFAAOCAQ8AMIIBCgKCAQEAy8D
bv8prpJ/0kKhlGeJY\n"\
"ozo2t60EG8L0561g13R29LvMR5hyvGZlGJpmn65+A4xHXIn
JYiPuKzrKUnApeLZ+\n"\
"vw1HocOAZtWK0z3r26uA8kQYOKX9Qt/DbCdvsF9wF8gRK0p
tx9M6R13NvBxvVQAp\n"\
"fc9jB9nTzphOgM4JiEYvlV8FLhg9yZovMYd6Wwf3aoXK891
VQxTr/kQYoq1Yp+68\n"\
"i6T4nNq7NWC+UNVjQHxNQMQMzU6lWCX8zyg3yH88OAQkUXI
XKfQ+NkvYQ1cxaMoV\n"\
"PpY72+eVthKzpMeyHkBn7ciumk5qgLTEJAfWZpe4f4eFZj/
Rc8Y8Jj2IS5kVPjUy\n"\
"wQIDAQAB\n"\
"-----END PUBLIC KEY-----\n";

 char privateKey[]="-----BEGIN RSA PRIVATE KEY--
---\n"\
"MIIEowIBAAKCAQEAy8Dbv8prpJ/0kKhlGeJYozo2t60EG8L
0561g13R29LvMR5hy\n"\
"vGZlGJpmn65+A4xHXInJYiPuKzrKUnApeLZ+vw1HocOAZtW
K0z3r26uA8kQYOKX9\n"\
"Qt/DbCdvsF9wF8gRK0ptx9M6R13NvBxvVQApfc9jB9nTzph
OgM4JiEYvlV8FLhg9\n"\
"yZovMYd6Wwf3aoXK891VQxTr/kQYoq1Yp+68i6T4nNq7NWC
+UNVjQHxNQMQMzU6l\n"\
"WCX8zyg3yH88OAQkUXIXKfQ+NkvYQ1cxaMoVPpY72+eVthK
zpMeyHkBn7ciumk5q\n"\
"gLTEJAfWZpe4f4eFZj/Rc8Y8Jj2IS5kVPjUywQIDAQABAoI
BADhg1u1Mv1hAAlX8\n"\
"omz1Gn2f4AAW2aos2cM5UDCNw1SYmj+9SRIkaxjRsE/C4o9
sw1oxrg1/z6kajV0e\n"\
"N/t008FdlVKHXAIYWF93JMoVvIpMmT8jft6AN/y3NMpivgt
2inmmEJZYNioFJKZG\n"\
"X+/vKYvsVISZm2fw8NfnKvAQK55yu+GRWBZGOeS9K+LbYvO
wcrjKhHz66m4bedKd\n"\
"gVAix6NE5iwmjNXktSQlJMCjbtdNXg/xo1/G4kG2p/MO1HL
cKfe1N5FgBiXj3Qjl\n"\
"vgvjJZkh1as2KTgaPOBqZaP03738VnYg23ISyvfT/teArVG
txrmFP7939EvJFKpF\n"\
"1wTxuDkCgYEA7t0DR37zt+dEJy+5vm7zSmN97VenwQJFWMi
ulkHGa0yU3lLasxxu\n"\
"m0oUtndIjenIvSx6t3Y+agK2F3EPbb0AZ5wZ1p1IXs4vktg
eQwSSBdqcM8LZFDvZ\n"\
"uPboQnJoRdIkd62XnP5ekIEIBAfOp8v2wFpSfE7nNH2u4Cp
AXNSF9HsCgYEA2l8D\n"\
"JrDE5m9Kkn+J4l+AdGfeBL1igPF3DnuPoV67BpgiaAgI4h2
5UJzXiDKKoa706S0D\n"\
"4XB74zOLX11MaGPMIdhlG+SgeQfNoC5lE4ZWXNyESJH1SVg
RGT9nBC2vtL6bxCVV\n"\
"WBkTeC5D6c/QXcai6yw6OYyNNdp0uznKURe1xvMCgYBVYYc
EjWqMuAvyferFGV+5\n"\
"nWqr5gM+yJMFM2bEqupD/HHSLoeiMm2O8KIKvwSeRYzNohK
TdZ7FwgZYxr8fGMoG\n"\
"PxQ1VK9DxCvZL4tRpVaU5Rmknud9hg9DQG6xIbgIDR+f79s
b8QjYWmcFGc1SyWOA\n"\
"SkjlykZ2yt4xnqi3BfiD9QKBgGqLgRYXmXp1QoVIBRaWUi5
5nzHg1XbkWZqPXvz1\n"\
"I3uMLv1jLjJlHk3euKqTPmC05HoApKwSHeA0/gOBmg404xy
AYJTDcCidTg6hlF96\n"\
"ZBja3xApZuxqM62F6dV4FQqzFX0WWhWp5n301N33r0qR6Fu
mMKJzmVJ1TA8tmzEF\n"\
"yINRAoGBAJqioYs8rK6eXzA8ywYLjqTLu/yQSLBn/4ta36K
8DyCoLNlNxSuox+A5\n"\
"w6z2vEfRVQDq4Hm4vBzjdi3QfYLNkTiTqLcvgWZ+eX44ogX
tdTDO7c+GeMKWz4XX\n"\
"uJSUVL5+CVjKLjZEJ6Qc2WZLl94xSwL71E41H4YciVnSCQx
Vc4Jw\n"\
"-----END RSA PRIVATE KEY-----\n";


unsigned char  encrypted[4098]={};
unsigned char decrypted[4098]={};

int encrypted_length=
public_encrypt(plainText,strlen(plainText),publi
cKey,encrypted);
if(encrypted_length == -1)
{
        printLastError("Public Encrypt failed
");
        exit(0);
}
printf("Encrypted length
=%d\n",encrypted_length);

int decrypted_length =
private_decrypt(encrypted,encrypted_length,priva
teKey, decrypted);
if(decrypted_length == -1)
{
        printLastError("Private Decrypt failed
");
        exit(0);
}
printf("Decrypted Text =%s\n",decrypted);
printf("Decrypted Length
=%d\n",decrypted_length);


encrypted_length=
private_encrypt(plainText,strlen(plainText),priv
ateKey,encrypted);
if(encrypted_length == -1)
{
        printLastError("Private Encrypt
failed");
        exit(0);
}
printf("Encrypted length
=%d\n",encrypted_length);

decrypted_length =
public_decrypt(encrypted,encrypted_length,public
Key, decrypted);
if(decrypted_length == -1)
{
        printLastError("Public Decrypt failed");
        exit(0);
}
printf("Decrypted Text =%s\n",decrypted);
printf("Decrypted Length
=%d\n",decrypted_length);




}
```

Reference:openssl documentaion → https://www.openssl.org/docs/crypto/pem.html