密码学实践

基于微软 CryptoAPI 的密码系统设计与实现

学　　院　计算机与电子信息学院

专　　业　信息安全

班　　级　[redacted]

姓　　名　[redacted]

学　　号　[redacted]

[redacted]　6　月 18 日

目录

1、系统需求分析

Cryptographic API (CryptoAPI) 是微软在 Windows 操作系统中添加的密码编译机能，作为数据加密与解密功能的重要基础，CryptoAPI 支持同步，异步的密钥加密处理，以及操作系统中的数字证书的管理工作。

本系统是基于 CryptoAPI 开发，使用 CryptoAPI 中提供的函数实现了生成公-私钥对并存储与加密解密信息与数据的功能。

2、系统功能说明

本系统实现基于 CryptoAPI 了如下功能：

- 生成公钥-私钥对
- 基于公钥私钥对进行文件对称加密

3、软硬件环境

开发环境：Microsoft Visual Studio Community 2019

开发语言：C++

操作系统：Window 7 Pro

硬件环境: ThinkPad X1 (2020)

4、相关 API 介绍

4.1 CryptAcquireContext function

用于获取句柄到一个特定的密钥容器的特定内加密服务提供商。

```
BOOL CryptAcquireContext(
    HCRYPTPROV *phProv,     //指向 CSP 句柄的指针
    LPCSTR      szContainer,  //密钥容器名称
    LPCSTR      szProvider,   //以空值结尾的字符串，其中包含要使用的 CSP 的名称
    DWORD       dwProvType,   //指定要获取的提供程序的类型
    DWORD       dwFlags       //标志值
);
```

4.2 CryptGenKey function

生成一个随机加密会话密钥或公共/私有密钥对。

```
BOOL CryptGenKey(
  HCRYPTPROV hProv,     //指向 CSP 句柄的指针
  ALG_ID     Algid,     //标识要为其生成密钥的算法
  DWORD      dwFlags,   //指定生成的密钥的类型
  HCRYPTKEY  *phKey     //新生成的 KEY 的句柄复制到的地址
);
```

4.3 CryptImportKey function

从 Key Blob 中提取密钥。

```
BOOL CryptImportKey(
  HCRYPTPROV hProv, //通过 CryptAcquireContext 函数获得的 CSP 的句柄
  const BYTE *pbData,  //Key Blob
  DWORD      dwDataLen,    //密钥 BLOB 的长度（以字节为单位）
  HCRYPTKEY  hPubKey,      //密钥 BLOB 的长度（以字节为单位）
  DWORD      dwFlags,//仅在将 PRIVATEKEYBLOB 形式的公钥/私钥对导入 CSP 时使用
  HCRYPTKEY  *phKey //指向 HCRYPTKEY 值的指针，该值接收导入的 KEY 的句柄
);
```

4.4 CryptExportKey function

导出密钥到 Key Blob。

```
BOOL CryptExportKey(
  HCRYPTKEY hKey,       //要导出的密钥的句柄
  HCRYPTKEY hExpKey,    //加密密钥的句柄(交换密钥)
  DWORD     dwBlobType,    //设定要在 pbData 中导出的 KEY BLOB 的类型
  DWORD     dwFlags,    //设定其他选项
  BYTE      *pbData,      //指向接收 KEY BLOB 数据的缓冲区的指针
  DWORD     *pdwDataLen
//指向 DWORD 值的指针，该值在输入时包含 pbData 参数指向的缓冲区的大小（以字节为单位）。函数返回时，此值包含缓冲区中存储的字节数。
);
```

4.5 CryptEncrypt function

用于加密数据。

```
BOOL CryptEncrypt(
    HCRYPTKEY    hKey,   //加密密钥的句柄。
    HCRYPTHASH hHash, //哈希对象句柄。
    BOOL         Final,    //一个布尔值，指定这是否是要解密的系列中的最后一个部分。
    DWORD        dwFlags,   //标志值。
    BYTE         *pbData,   //指向包含要加密的数据的缓冲区的指针。
    DWORD    *pdwDataLen, //指向 DWORD 值的指针，该值在输入时包含 pbData 缓冲区中纯文本的长度（以字节为单位）。
    DWORD        dwBufLen   //指定输入 pbData 缓冲区的总大小（以字节为单位）。
);
```

4.6 CryptDecrypt function

解密先前通过使用 CryptEncrypt 加密的数据。

```
BOOL CryptDecrypt(
  HCRYPTKEY    hKey,      //用于解密的密钥的句柄。
  HCRYPTHASH hHash,     //哈希对象句柄。
  BOOL         Final,       //一个布尔值，指定这是否是要解密的系列中的最后一个部分。
  DWORD          dwFlags,  //标志值。
  BYTE         *pbData,    //指向包含要解密的数据的缓冲区的指针。
  DWORD    *pdwDataLen  //指向 DWORD 值的指针，该值指示 pbData 缓冲区的长度。
);
```

5、系统设计的步骤

5.1 使用 CryptApi 进行公私钥生成的步骤

- 使用 CryptAcquireContext()连接 CSP 容器，获得指定连接 CSP,获得指定。
- 使用 CryptGenKey()创建随机的公钥私钥对
- 使用 CryptExportKey()获取公、私钥长度，然后使用 CryptExportKey()进行公、私钥的导出
- 使用 Createfile()打开文件，使用 Writefile()写入文件，完成公私钥生成

5.2 使用 CryptApi 进行文件加密

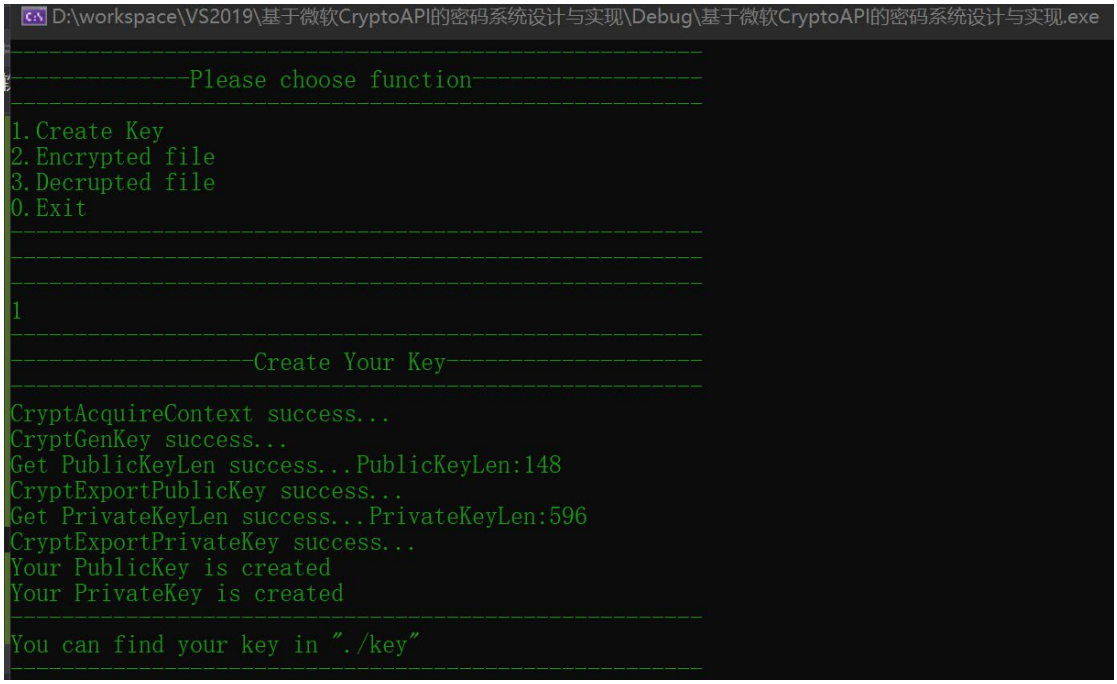
- 使用 CryptAcquireContext()连接 CSP 容器，获得指定连接 CSP,获得指定。
- 使用 Createfile(),Writefile()读取公钥文件。
- 使用 CryptImportKey()装载公钥。
- 使用 CryptGenKey()生成会话密钥。
- 调用 Crypt Encrypt()使用会话密钥进行文件加密。
- 使用 Createfile(),Writefile()存储被加密文件。
- 使用 CryptExportKey()导出会话密钥。
- 使用 Createfile(),Writefile()存储密钥文件。

5.3 使用 CryptApi 进行文件解密

- 使用 CryptAcquireContext()连接 CSP 容器，获得指定连接 CSP,获得指定。
- 使用 Createfile(),Writefile()读取私钥文件。
- 使用 CryptImportKey()装载私钥。
- 使用 Createfile(),Writefile()读取会话密钥文件。
- 使用 CryptImportKey()装载会话密钥。
- 使用 Createfile(),Writefile()读取被加密文件。
- 调用 CryptDecrypt()使用会话密钥进行文件解密。
- 使用 Createfile(),Writefile()存储解密文件。

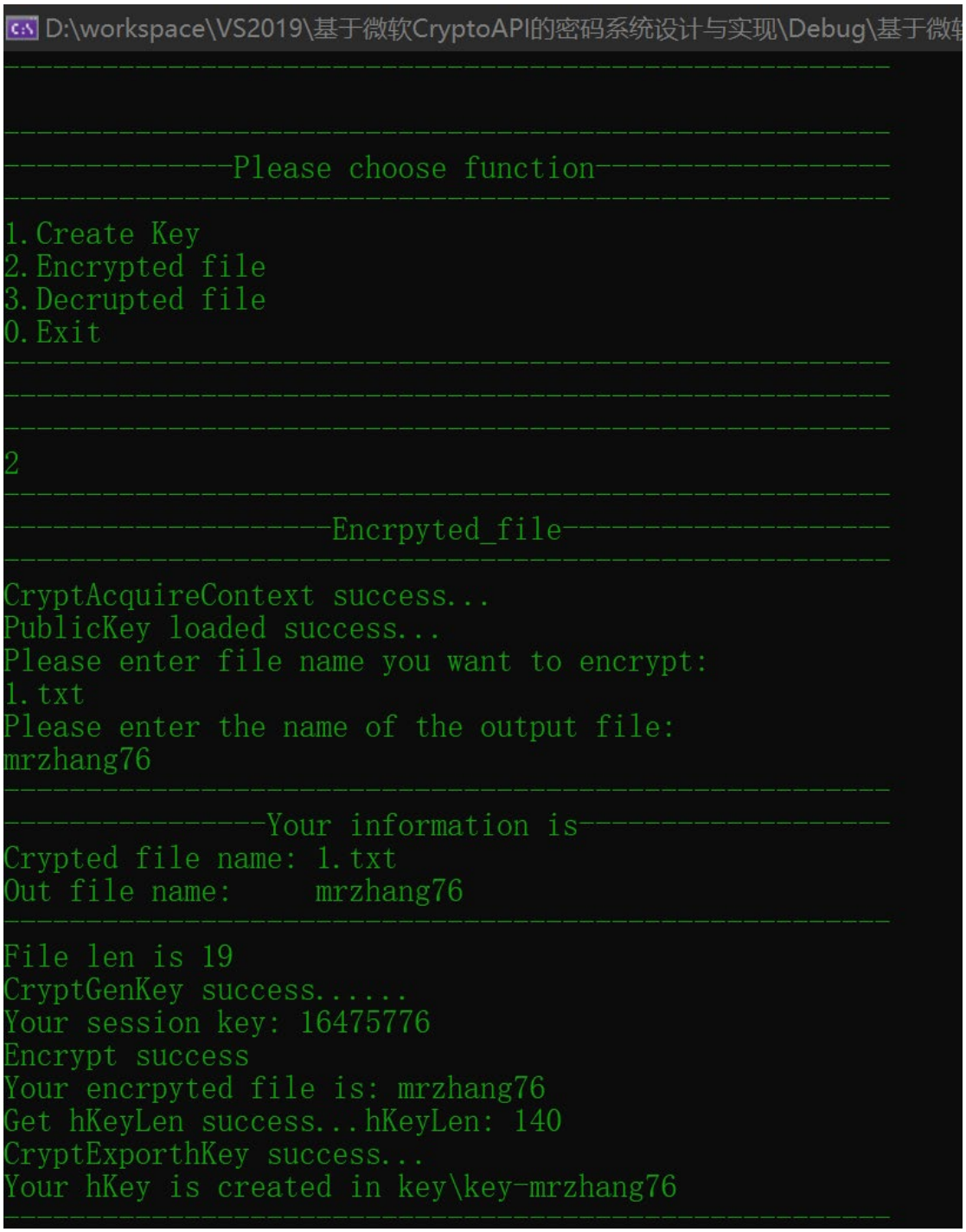
6、系统功能验证

6.1 使用 CryptApi 进行公私钥生成

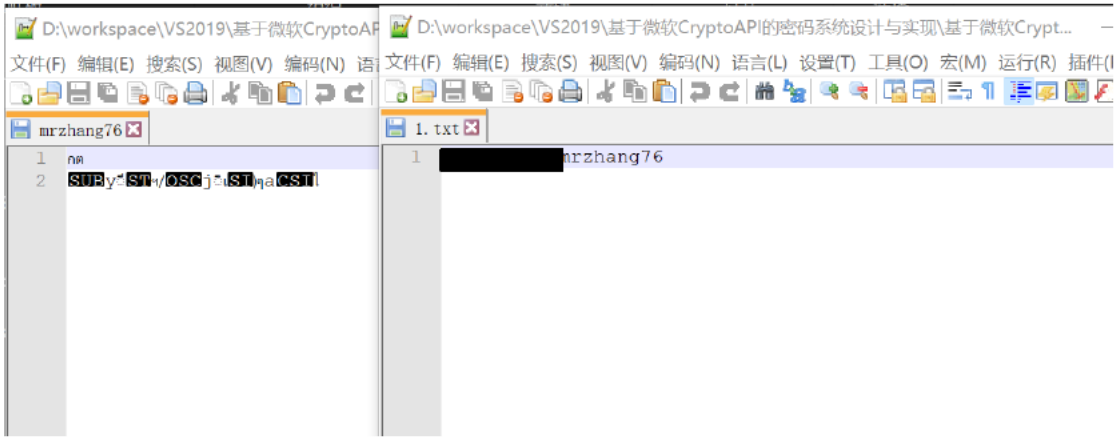


密钥已经生成：

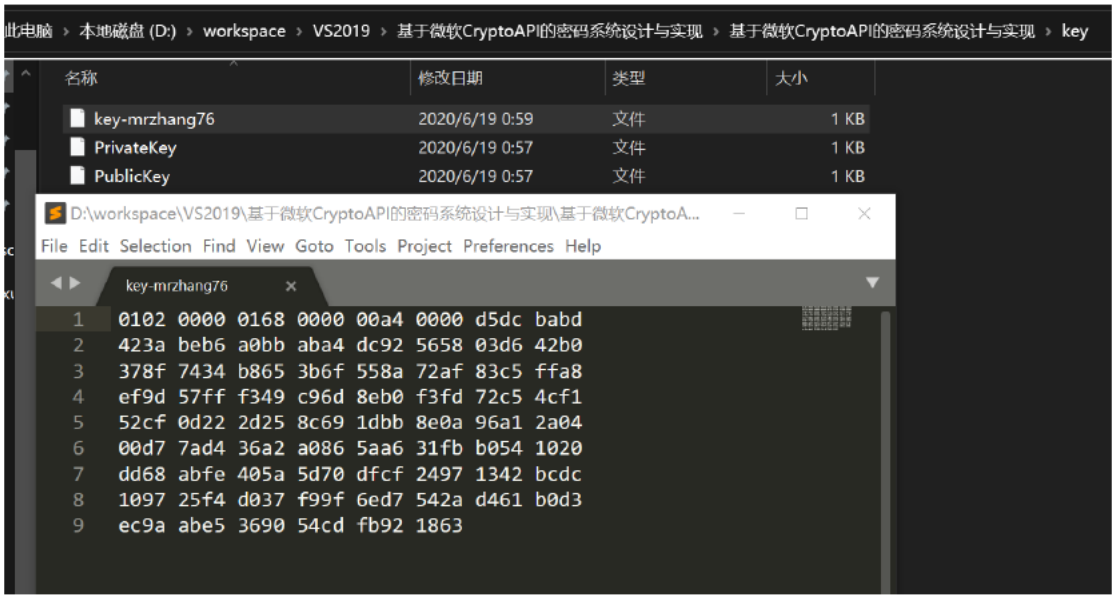
6.2 使用 CryptApi 进行文件加密



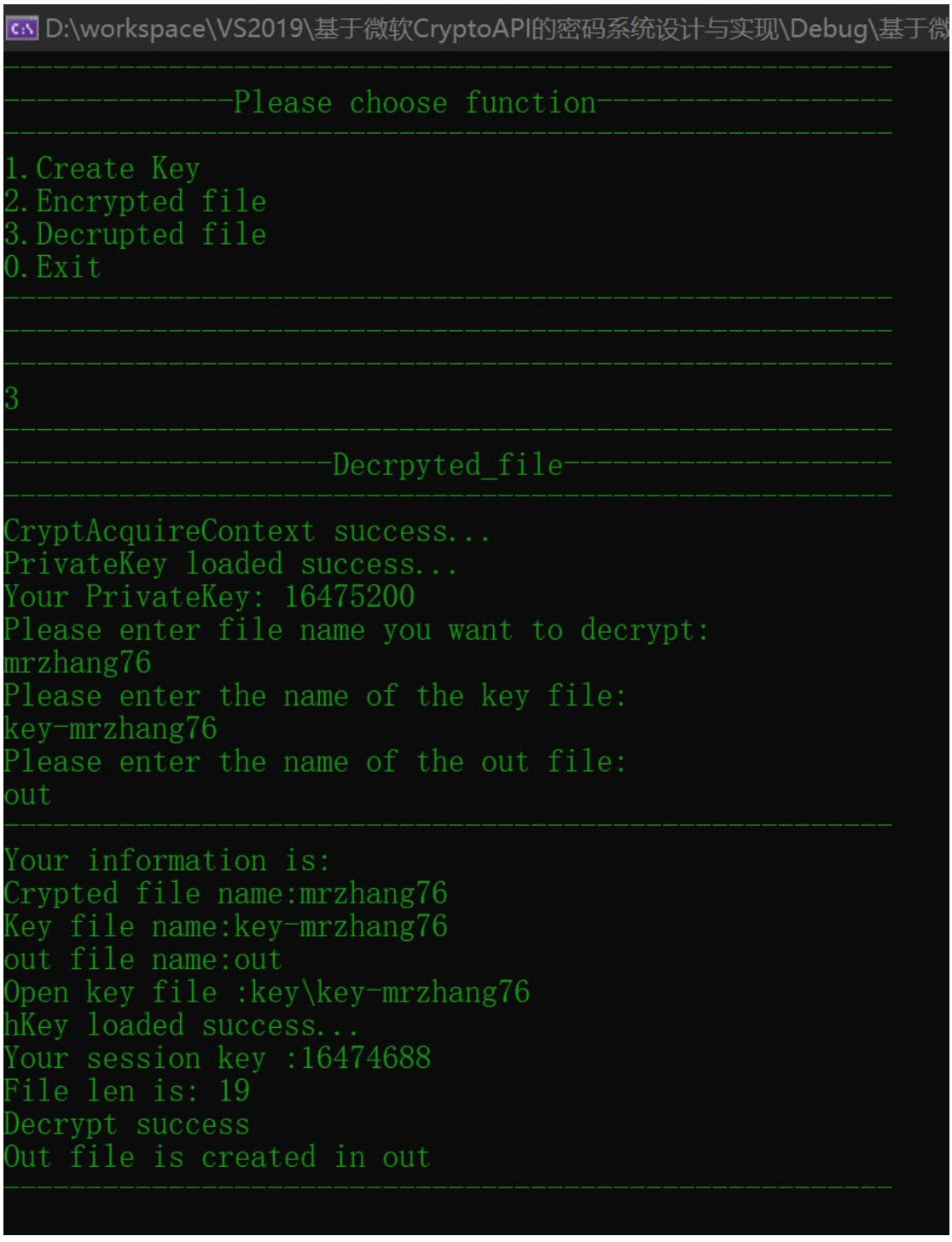
被加密文件：



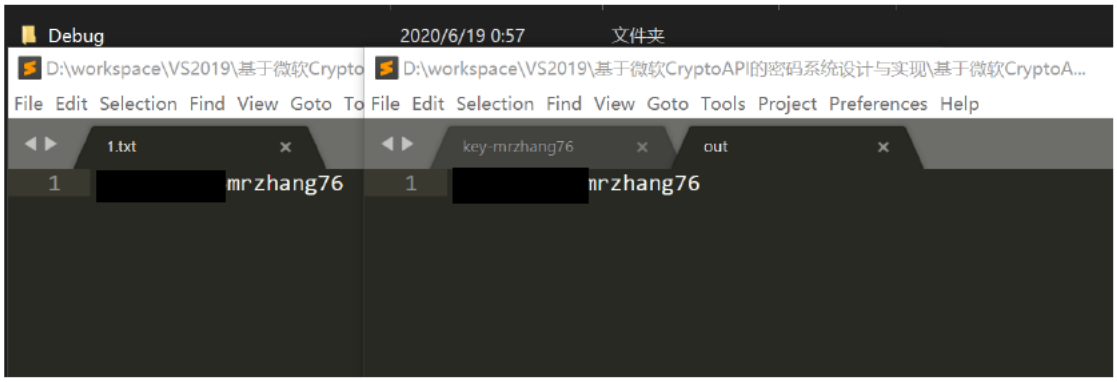
生成的会话密钥文件：



6.3 使用 CryptApi 进行文件解密



文件解密成功：



附录：源代码

```
#include "main.h"

using namespace std;

void HandleError(const char *s);

void Encrpyted_file();

void Decrpyted_file();

void Generate_keys();

int main(void) {

    while (1) {

        int flag = -1;

        cout << "------------------------------------------------" << endl;

        cout << "--------------Please choose function------------------" << endl;

        cout << "------------------------------------------------" << endl;

        cout << "1.Create Key" << endl;

        cout << "2.Encrypted file" << endl;
```

```
            cout << "3.Decrupted file" << endl;
            cout << "0.Exit" << endl;
            cout << "-----------------------------------------------------" << endl;
            cout << "-----------------------------------------------------" << endl;
            cout << "-----------------------------------------------------" << endl;
            cin >> flag;
            switch (flag) {
                case 1:
                    Generate_keys();
                    break;
                case 2:
                    Encrpyted_file();
                    break;
                case 3:
                    Decrpyted_file();
                    break;
                case 0:
                    exit(0);
            }

        }
    }
    void HandleError(const char *s) {
        cout <<"Error occurred"<< endl;
```

```
    cout <<    s << endl;
    printf("Error Code :%x", GetLastError());
    exit(1);
}

void Encrpyted_file() {
    HCRYPTPROV hProv = NULL;          //CSP 句柄指针
    HCRYPTHASH hHash = NULL;          //Hash 对象
    HCRYPTKEY hKey = NULL;            //会话密钥句柄
    PBYTE hKey_out = NULL;            //会话密钥
    HANDLE hKeyFile = NULL;           //会话密钥文件
    DWORD hKeyLen = NULL;             //会话密钥长度
    PBYTE pbBuffer = NULL;            //数据缓存
    DWORD r_file_len = NULL;          //文件长度
    HCRYPTKEY hXchgKey = NULL;        //公钥句柄
    HANDLE PublicKeyFile = NULL; //公钥文件
    PBYTE hPublicKey = NULL;          //公钥指针
    DWORD hPublicKeyLen = NULL; //公钥长度
    HANDLE m_File = NULL;             //加密文件
    DWORD FILESIZE = NULL;            //文件长度
    DWORD lpNumberOfBytesWritten = 0;

    cout << "------------------------------------------------------" << endl;
    cout << "--------------------Encrpyted_file--------------------" << endl;
```

```
cout << "----------------------------------------------------" << endl;


//连接 CSP,获得指定 CSP 密钥容器的句柄

if (CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_FULL, 0))

    cout << "CryptAcquireContext success..." << endl;

else

    HandleError("CryptAcquireContext error");




//读取公钥文件

if ((PublicKeyFile = CreateFile(L"key\\PublicKey", GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
NULL)) == INVALID_HANDLE_VALUE) {

    HandleError("OPEN PublicKey error");

}

FILESIZE = GetFileSize(PublicKeyFile, NULL);

hPublicKey = (PBYTE)malloc(FILESIZE);

if(!ReadFile(PublicKeyFile,
hPublicKey,FILESIZE,&hPublicKeyLen,NULL))

    HandleError("READ PublicKey error");

//公钥装载

if(CryptImportKey(hProv,hPublicKey,hPublicKeyLen,0,NULL,
&hXchgKey))

    cout << "PublicKey loaded success..." << endl;
```

```
char file_name[50];
char out_file_name[50];
cout << "Please enter file name you want to encrypt:" << endl;
cin >> file_name;
cout << "Please enter the name of the output file:" << endl;
cin >> out_file_name;
cout << "------------------------------------------------------" << endl;
cout << "----------------Your information is-------------------" << endl;
cout << "Crypted file name: " << file_name << endl;
cout << "Out file name:          " << out_file_name << endl;
cout << "------------------------------------------------------" << endl;

//打开待加密文件
ifstream r_file;
r_file.open(file_name, ios::in);
if(!r_file.is_open())
      HandleError("Cannot find your file");
streampos pos = r_file.tellg();
r_file.seekg(0, ios::end);
r_file_len = r_file.tellg();
r_file.seekg(pos);
cout << "File len is " << r_file_len << endl;

//创建会话密钥
```

```
        if (CryptGenKey(hProv, ENCRYPT_ALGORITHM, KEYLENGTH | CRYPT_EXPORTABLE, &hKey)) {
            cout << "CryptGenKey success....." << endl;
            cout << "Your session key: " << hKey << endl;
        }

        CHAR* temp = NULL;
        pbBuffer = (PBYTE)malloc(r_file_len);
        temp = (PCHAR)malloc(r_file_len);
        r_file.read(temp, r_file_len);
        memcpy(pbBuffer,temp,r_file_len);
        r_file.close();

        //加密数据文件
        if (CryptEncrypt(hKey,NULL,TRUE,  0,pbBuffer,&r_file_len,r_file_len))
            cout << "Encrypt success" << endl;
        else
            HandleError("Encrypt faild");

        //存储加密数据
        if ((m_File = CreateFileA(out_file_name, GENERIC_WRITE, 0, NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL)) == INVALID_HANDLE_VALUE)
            HandleError("Create m_File error");
        if (!(WriteFile(m_File, (LPCVOID)pbBuffer, r_file_len,
```

```
&lpNumberOfBytesWritten, NULL)))
        HandleError("Write m_File error");
    FlushFileBuffers(m_File);
    CloseHandle(m_File);
    cout << "Your encrpyted file is: " <<out_file_name<< endl;

    //导出会话密钥
    if (CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, NULL,
&hKeyLen))
        cout << "Get hKeyLen success...hKeyLen: " << hKeyLen << endl;
    else
        HandleError("Get hKeyLen error");
    //获取内存空间
    if (!(hKey_out = (PBYTE)malloc(hKeyLen)))
        HandleError("malloc error");
    //导出会话密钥到内存空间
    if (CryptExportKey(hKey, hXchgKey, SIMPLEBLOB, 0, hKey_out,
&hKeyLen))
        cout << "CryptExporthKey success..." << endl;
    else
        HandleError("CryptExporthKey error");
    //导出会话密钥到文件
    CHAR key_name[50] = { "key\\key-" };
    strcat_s(key_name, out_file_name);
    if ((hKeyFile = CreateFileA(key_name, GENERIC_WRITE, 0, NULL,
```

```
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL)) == INVALID_HANDLE_VALUE)
            HandleError("CreatehKeyFile error");
        if (!(WriteFile(hKeyFile, (LPCVOID)hKey_out, hKeyLen, &lpNumberOfBytesWritten, NULL)))
            HandleError("WritehKeyFile error");
        FlushFileBuffers(hKeyFile);
        CloseHandle(hKeyFile);
        cout << "Your hKey is created in " << key_name <<endl;

        CryptReleaseContext(hProv, 0);
        cout << "------------------------------------------------------\n" << endl;
    }

    void Decrpyted_file() {
        HCRYPTPROV hProv = NULL;   //CSP 句柄指针
        HCRYPTKEY hXchgKey = NULL; //私钥句柄
        HANDLE PrivateKeyFile = NULL;//私钥文件
        PBYTE hPrivateKey = NULL;//私钥
        DWORD hPrivateKeyLen = NULL; //私钥长度
        HANDLE hKeyFile = NULL; //会话密钥文件
        HCRYPTKEY hKey = NULL;          //会话密钥句柄
        PBYTE hKey_in = NULL;    //会话密钥
        DWORD hKeyLen = NULL;  //会话密钥长度
```

```
HANDLE mFile = NULL;            //待解密文件
PBYTE mfile_in = NULL;    //待解密文件指针
DWORD mfileLen = NULL;  //待解密文件长度
DWORD FILESIZE = NULL;      //文件长度
DWORD lpNumberOfBytesWritten = 0;
HANDLE OutFile = NULL; //导出文件



cout << "-----------------------------------------------------" << endl;
cout << "--------------------Decrpyted_file--------------------" << endl;
cout << "-----------------------------------------------------" << endl;


//连接 CSP,获得指定 CSP 密钥容器的句柄
if (CryptAcquireContext(&hProv, NULL, NULL, PROV_RSA_FULL, 0))
    cout << "CryptAcquireContext success..." << endl;
else
    HandleError("CryptAcquireContext error");


//读取私钥文件
if ((PrivateKeyFile = CreateFile(L"key\\PrivateKey", GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
NULL)) == INVALID_HANDLE_VALUE) {
    HandleError("OPEN PrivateKey error");
}
```

```
        FILESIZE = GetFileSize(PrivateKeyFile, NULL);
        hPrivateKey = (PBYTE)malloc(FILESIZE);
        if (!ReadFile(PrivateKeyFile, hPrivateKey, FILESIZE, &hPrivateKeyLen,
NULL))
            HandleError("READ PrivateKey error");
        //私钥装载
       if (CryptImportKey(hProv, hPrivateKey, hPrivateKeyLen, 0, 0, &hXchgKey))
{
            cout << "PrivateKey loaded success..." << endl;
            cout << "Your PrivateKey: " << hXchgKey << endl;
        }

        char file_name[50];
        char key_name[50];
        char out_file_name[50];
        cout << "Please enter file name you want to decrypt:" << endl;
        cin >> file_name;
        cout << "Please enter the name of the key file:" << endl;
        cin >> key_name;
        cout << "Please enter the name of the out file:" << endl;
        cin >> out_file_name;
        cout << "----------------------------------------------------" << endl;
        cout << "Your information is:" << endl;
        cout << "Crypted file name:" << file_name << endl;
        cout << "Key file name:" << key_name << endl;
```

```
        cout << "out file name:" << out_file_name << endl;


        CHAR key_file_name[50] = { "key\\" };
        strcat_s(key_file_name, key_name);
        cout << "Open key file :" << key_file_name << endl;


        //读取会话密钥文件
        if ((hKeyFile = CreateFileA(key_file_name, GENERIC_READ,
FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
NULL)) == INVALID_HANDLE_VALUE) {
            HandleError("OPEN hKeyfile error");
        }
        FILESIZE = GetFileSize(hKeyFile, NULL);
        hKey_in = (PBYTE)malloc(FILESIZE);
        if (!ReadFile(hKeyFile, hKey_in, FILESIZE, &hKeyLen, NULL))
            HandleError("READ PrivateKey error");


        //会话密钥装载
        if (CryptImportKey(hProv, hKey_in, hKeyLen, hXchgKey, NULL, &hKey))
{
            cout << "hKey loaded success..." << endl;
            cout << "Your session key :" << hKey << endl;
        }
        //读取以加密文件
        if ((mFile = CreateFileA(file_name, GENERIC_READ,
```

```
FILE_SHARE_READ, NULL, OPEN_EXISTING, FILE_ATTRIBUTE_NORMAL,
NULL)) == INVALID_HANDLE_VALUE) {

        HandleError("OPEN mfile error");

    }

    FILESIZE = GetFileSize(mFile, NULL);

    mfile_in = (PBYTE)malloc(FILESIZE);

    if (!ReadFile(mFile, mfile_in, FILESIZE, &mfileLen, NULL))

        HandleError("READ mfile error");

    cout << "File len is: " << mfileLen << endl;




    //解密文件

    if (CryptDecrypt(hKey,NULL,TRUE,  0,mfile_in,&mfileLen)) {

        cout << "Decrypt success" << endl;

    }


    //存储已解密文件

    if ((OutFile = CreateFileA(out_file_name, GENERIC_WRITE, 0, NULL,
CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL)) ==
INVALID_HANDLE_VALUE)

        HandleError("CreatehKeyFile error");

    if (!(WriteFile(OutFile, (LPCVOID)mfile_in, mfileLen,
&lpNumberOfBytesWritten, NULL)))

        HandleError("WritehKeyFile error");
```

```
    FlushFileBuffers(OutFile);
    CloseHandle(OutFile);
    cout << "Out file is created in " << out_file_name << endl;
    cout << "-----------------------------------------------------\n" << endl;
    CryptReleaseContext(hProv, 0);
}

void Generate_keys() {
    HCRYPTPROV hProv = NULL;   //CSP 句柄指针
    HCRYPTKEY hKey = NULL;      //会话密钥
    DWORD PublicKeyLen = 0; //公钥长度
    DWORD PrivateKeyLen = 0;//私钥长度
    PBYTE PublicKey = NULL; //公钥
    PBYTE PrivateKey = NULL;//私钥
    HANDLE hPublicKeyFile = NULL; //公钥文件
    HANDLE hPrivateKeyFile = NULL;//私钥文件
    DWORD lpNumberOfBytesWritten = 0;
    CHAR strPublicKeyFile;
    CHAR* strPrivateKeyFile;
    //连接 CSP,获得指定 CSP 密钥容器的句柄
    cout << "-----------------------------------------------------" << endl;
    cout << "-------------------Create Your Key-------------------" << endl;
    cout << "-----------------------------------------------------" << endl;
    if (CryptAcquireContext(&hProv, NULL,NULL, PROV_RSA_FULL, 0))
```

```
        cout << "CryptAcquireContext success..." << endl;
    else
        HandleError("CryptAcquireContext error");

    //创建随机的公钥私钥对
    if(CryptGenKey(hProv, AT_KEYEXCHANGE, CRYPT_ARCHIVABLE,
&hKey))
        cout << "CryptGenKey success..." << endl;
    else
        HandleError("CryptGenKey error");

    //导出公钥
    //获取公钥长度
    if(CryptExportKey(hKey,NULL,PUBLICKEYBLOB,0,NULL,&PublicKeyL
en))
        cout << "Get PublicKeyLen success...PublicKeyLen:" << PublicKeyLen
<<endl;
    else
        HandleError("Get PublicKeyLen error");
    //获取内存空间
    if (!(PublicKey = (PBYTE)malloc(PublicKeyLen)))
        HandleError("malloc error");
    //导出公钥到内存空间
    if (CryptExportKey(hKey, NULL, PUBLICKEYBLOB, 0, PublicKey,
&PublicKeyLen)) {
```

```
        cout << "CryptExportPublicKey success..." << endl;
    }
    else
        HandleError("CryptExportPublicKey error");
    //导出私钥
    //获取私钥长度
    if (CryptExportKey(hKey, NULL, PRIVATEKEYBLOB, 0, NULL, &PrivateKeyLen))
        cout << "Get PrivateKeyLen success...PrivateKeyLen:" << PrivateKeyLen <<endl;
    else
        HandleError("Get PrivateKeyLen error");
    //获取内存空间
    if (!(PrivateKey = (PBYTE)malloc(PrivateKeyLen)))
        HandleError("malloc error");
    //导出私钥到内存空间
    if (CryptExportKey(hKey, NULL, PRIVATEKEYBLOB, 0, PrivateKey, &PrivateKeyLen)) {
        cout << "CryptExportPrivateKey success..." << endl;
    }
    else
        HandleError("CryptExportPrivateKey error");

    //存放公钥至文件
    if((hPublicKeyFile=CreateFile(L"key\\PublicKey",
```

```
GENERIC_WRITE,0,NULL,CREATE_ALWAYS,FILE_ATTRIBUTE_NORMAL,N
ULL)) == INVALID_HANDLE_VALUE)

        HandleError("CreatePublicKeyFile error");

    if(!(WriteFile(hPublicKeyFile,(LPCVOID)PublicKey,
PublicKeyLen,&lpNumberOfBytesWritten,NULL)))

        HandleError("WritePublicKeyFile error");

    FlushFileBuffers(hPublicKeyFile);

    CloseHandle(hPublicKeyFile);

    cout << "Your PublicKey is created"<< endl;


    //存放私钥至文件

    if((hPrivateKeyFile = CreateFile(L"key\\PrivateKey", GENERIC_WRITE, 0,
NULL, CREATE_ALWAYS, FILE_ATTRIBUTE_NORMAL, NULL)) ==
INVALID_HANDLE_VALUE)

        HandleError("CreatePrivateKeyFile error");

    if(!WriteFile(hPrivateKeyFile, (LPCVOID)PrivateKey, PrivateKeyLen,
&lpNumberOfBytesWritten, NULL))

        HandleError("WritePrivateKeyFile error");

    FlushFileBuffers(hPrivateKeyFile);

    CloseHandle(hPrivateKeyFile);


    cout << "Your PrivateKey is created" << endl;

    cout << "-----------------------------------------------------" << endl;

    cout << "You can find your key in \"./key\"" << endl;

    cout << "-----------------------------------------------------\n" << endl;

    free(PublicKey);
```

```
        free(PrivateKey);

        CryptDestroyKey(hKey);

        CryptReleaseContext(hProv, 0);



}
```