Министерство образования и науки Украины

Харьковский национальный университет радиоэлектроники

Кафедра БИТ

Отчет

 По лабораторной работе по АнКр №4

Тема: Хешифрование и ЭЦП

Выполнили:  Проверил: ст.гр БИКС-14-1 Мельникова О. А.

Д. Тхао

Харьков 2017

Формирования ЭЦП

#include <stdio.h>

#include <windows.h>

#include <Wincrypt.h>

void GethProv(HCRYPTPROV &hProv)

{

LPTSTR pszContainer = "ContainerLab\_2\_1";

if (CryptAcquireContext(&hProv, pszContainer, 0, PROV\_RSA\_AES, 0)) {

printf("The key container %s has been acquired.\n\n", pszContainer);

}

else {

if (GetLastError() != NTE\_BAD\_KEYSET) {

printf("ERROR creating new key container.\n");

}

if (!CryptAcquireContext(&hProv, pszContainer, 0, PROV\_RSA\_AES, CRYPT\_NEWKEYSET)) {

printf("ERROR acquiring a cryptographic context.\n");

}

printf("A new key container \"%s\" has been created.\n\n", pszContainer);

}

}

void GenAllKey(HCRYPTPROV hProv, HCRYPTKEY &hSignKey, HCRYPTKEY &hSessKey)

{

PROV\_ENUMALGS\_EX EnumData, DataSign, DataSess;

DWORD cbData = 1000;

DWORD dwFlag = CRYPT\_FIRST;

CryptGetProvParam(hProv, PP\_ENUMALGS\_EX, (BYTE\*)&EnumData, &cbData, CRYPT\_FIRST);

do {

if (EnumData.aiAlgid == CALG\_RSA\_SIGN)

DataSign = EnumData;

if (EnumData.aiAlgid == CALG\_AES\_128)

DataSess = EnumData;

} while (CryptGetProvParam(hProv, PP\_ENUMALGS\_EX, (BYTE\*)&EnumData, &cbData, CRYPT\_NEXT));

dwFlag = (DataSign.dwMinLen << 16) | CRYPT\_EXPORTABLE;

if (!CryptGenKey(hProv, DataSign.aiAlgid, dwFlag, &hSignKey))

{

printf(" GenKey Failed!\n\n");

}

else printf(" A Sign Key of %d bits length is created.\n", DataSign.dwMinLen);

if (!CryptGenKey(hProv, DataSess.aiAlgid, CRYPT\_EXPORTABLE, &hSessKey)){

printf(" GenKey Failed!\n\n");

}

else printf(" A Sess Key of defalt bits length is created.\n");

}

int WriteToFile(char \*filename, PBYTE pbData, DWORD cbData)

{

FILE \*fout;

if (!(fout = fopen(filename, "wb"))) {

printf("Error writing Signature to file.\n");

return 0;

}

if (!fwrite(&cbData, sizeof(DWORD), 1, fout)) {

printf("Error writing to file!\n");

return 0;

}

if (!fwrite(pbData, sizeof(BYTE), cbData, fout)) {

printf("Error writing to file!\n");

}

fclose(fout);

}

void ExportKey(HCRYPTKEY hKey, char \*filename)

{

PBYTE pbKeyBlob;

DWORD dwBlobLen;

if (CryptExportKey(hKey, NULL, PUBLICKEYBLOB, 0, NULL, &dwBlobLen)) {

printf("Size of the BLOB for the public key determined. \n");

}

else {

printf("Error computing BLOB length.\n");

}

if (pbKeyBlob = (BYTE\*)malloc(dwBlobLen)) {

printf("Memory has been allocated for the BLOB. \n");

}

else {

printf("Out of memory. \n");

}

if (CryptExportKey(hKey, NULL, PUBLICKEYBLOB, 0, pbKeyBlob, &dwBlobLen)) {

printf("Contents have been written to the BLOB. \n");

}

else {

printf("Error during CryptExportKey.");

}

WriteToFile(filename, pbKeyBlob, dwBlobLen);

}

int GetHash(HCRYPTPROV hProv, HCRYPTHASH &hHash, HCRYPTKEY hSessKey, char \*FileName)

{

BYTE \*pbBuffer = new BYTE[1024];

DWORD dwBufferLen;

if (CryptCreateHash(hProv, CALG\_MD5, 0, 0, &hHash)) {

printf("Hash object created. \n");

}

else {

MyHandleError("Error during CryptCreateHash.");

}

FILE \*fin = fopen("otchot.docx", "wb");

if (!fin) {

printf("Error reading Data.txt!\n");

return 0;

}

while (feof(fin)) {

dwBufferLen = fread(pbBuffer, sizeof(BYTE), 1024, fin);

if (dwBufferLen < 1024 && (!feof(fin)))

printf("Error reading file Data.txt\n");

if (!CryptHashData(hHash, pbBuffer, dwBufferLen, 0))

printf("Error during CryptHashData.\n");

else printf("The data has been added to the hash.\n\n");

if (CryptHashSessionKey(hHash, hSessKey, 0)) {

printf("The session key has been hashed. \n");

}

}

fclose(fin);

}

void SignHash(HCRYPTHASH &hHash, char \*FileName)//, BYTE \*&pbSignature, DWORD &dwSigLen)

{

BYTE \*pbSignature;

DWORD dwSigLen;

dwSigLen = 0;

if (CryptSignHash(hHash, AT\_SIGNATURE, NULL, 0, NULL, &dwSigLen)) {

printf("Signature length %d found.\n", dwSigLen);

}

else {

printf("Error during CryptSignHash.\n");

}

if (pbSignature = (BYTE \*)malloc(dwSigLen)) {

printf("Memory allocated for the signature.\n");

}

else {

printf("Out of memory.\n");

}

if (CryptSignHash(hHash, AT\_SIGNATURE, NULL, 0, pbSignature, &dwSigLen)) {

printf("pbSignature is the hash signature.\n");

}

else{

printf("Error during CryptSignHash.\n");

}

WriteToFile(FileName, pbSignature, dwSigLen);

}

int ReadFile(char \*filename, PBYTE &pbData, DWORD &cbData)

{

FILE \*fin;

if (!(fin = fopen(filename, "rb"))) {

printf("Error writing Signature to file.\n");

return 0;

}

if (!fread(&cbData, sizeof(DWORD), 1, fin)) {

printf("Error reading file!\n");

return 0;

}

pbData = new BYTE[cbData];

if (!fread(pbData, sizeof(BYTE), cbData, fin)) {

printf("Error reading file!\n");

return 0;

}

fclose(fin);

return 1;

}

int main(int argc, char \*argv[])

{

if (argc < 4) return 0;

HCRYPTPROV hProv;

HCRYPTHASH hHash;

HCRYPTKEY hKey;

HCRYPTKEY hSessKey;

HCRYPTKEY hPubKey;

LPTSTR pszPubKey = argv[2];

LPTSTR pszHData = argv[3];

LPTSTR pszSignature = argv[4];

GethProv(hProv);

GenAllKey(hProv, hKey, hSessKey);

ExportKey(hKey, pszPubKey);

GetHash(hProv, hHash, hSessKey, pszHData);

SignHash(hHash, pszSignature);

if (hHash) CryptDestroyHash(hHash);

printf("The hash object has been destroyed.\n");

printf("The signing phase of this program is completed.\n\n");

if (hHash) CryptDestroyHash(hHash);

if (hSessKey) CryptDestroyKey(hSessKey);

if (hPubKey) CryptDestroyKey(hPubKey);

if (hProv)

CryptReleaseContext(hProv, 0);

system("pause");

return 0;

}

Проверки ЭЦП

#include <stdio.h>

#include <windows.h>

#include <Wincrypt.h>

void GethProv(HCRYPTPROV &hProv)

{

LPTSTR pszContainer = "ContainerLab\_2\_1";

if (CryptAcquireContext(&hProv, pszContainer, 0, PROV\_RSA\_AES, 0)) {

printf("The key container %s has been acquired.\n\n", pszContainer);

}

else {

if (GetLastError() != NTE\_BAD\_KEYSET) {

printf("ERROR creating new key container.\n");

}

if (!CryptAcquireContext(&hProv, pszContainer, 0, PROV\_RSA\_AES, CRYPT\_NEWKEYSET)) {

printf("ERROR acquiring a cryptographic context.\n");

}

printf("A new key container \"%s\" has been created.\n\n", pszContainer);

}

}

void GenPrivateKey(HCRYPTPROV hProv, HCRYPTKEY &hSessKey)

{

PROV\_ENUMALGS\_EX DataSess;

DWORD cbData = 1000;

CryptGetProvParam(hProv, PP\_ENUMALGS\_EX, (BYTE\*)&DataSess, &cbData, CRYPT\_FIRST);

while (DataSess.aiAlgid == CALG\_AES\_128)

{

CryptGetProvParam(hProv, PP\_ENUMALGS\_EX,

(BYTE\*)&DataSess, &cbData, CRYPT\_NEXT);

}

if (!CryptGenKey(hProv, DataSess.aiAlgid, CRYPT\_EXPORTABLE, &hSessKey)) {

printf(" GenKey Failed!\n\n");

}

else printf(" A Sess Key of defalt bits length is created.\n");

}

int GetHash(HCRYPTPROV hProv, HCRYPTHASH &hHash, HCRYPTKEY hSessKey, char \*FileName)

{

BYTE \*pbBuffer = new BYTE[1024];

DWORD dwBufferLen;

if (CryptCreateHash(hProv, CALG\_MD5, 0, 0, &hHash)) {

printf("Hash object created. \n");

}

else {

printf("Error during CryptCreateHash.");

}

FILE \*fin = fopen(FileName, "wb");

if (!fin) {

printf("Error reading %s!\n", FileName);

return 0;

}

while (feof(fin)) {

dwBufferLen = fread(pbBuffer, sizeof(BYTE), 1024, fin);

if (dwBufferLen < 1024 && (!feof(fin)))

printf("Error reading file Data.txt\n");

if (!CryptHashData(hHash, pbBuffer, dwBufferLen, 0))

printf("Error during CryptHashData.\n");

else printf("The data has been added to the hash.\n\n");

if (CryptHashSessionKey(hHash, hSessKey, 0)) {

printf("The session key has been hashed. \n");

}

}

fclose(fin);

}

int ReadFile(char \*filename, PBYTE &pbData, DWORD &cbData)

{

FILE \*fin;

if (!(fin = fopen(filename, "rb"))) {

printf("Can not open file %s.\n", filename);

return 0;

}

if (!fread(&cbData, sizeof(DWORD), 1, fin)) {

printf("Error reading file!\n");

return 0;

}

pbData = new BYTE[cbData];

if (!fread(pbData, sizeof(BYTE), cbData, fin)) {

printf("Error reading file!\n");

return 0;

}

fclose(fin);

return 1;

}

int ImportKey(HCRYPTPROV hProv, char \*FileName, HCRYPTKEY &hPubKey)

{

PBYTE pbKeyBlob;

DWORD dwBlobLen;

if (!ReadFile(FileName, pbKeyBlob, dwBlobLen)) return 0;

if (CryptImportKey(hProv, pbKeyBlob, dwBlobLen, 0, 0, &hPubKey)) {

printf("The key has been imported.\n");

}

else {

printf("Public key import failed.");

}

return 1;

}

int VerifyHash(HCRYPTHASH hHash, HCRYPTKEY hPubKey, char \*FileName)

{

BYTE \*pbSignature;

DWORD dwSigLen = 0;

if (!ReadFile(FileName, pbSignature, dwSigLen)) return 0;

if (CryptVerifySignature(hHash, pbSignature, dwSigLen, hPubKey, NULL, 0)) {

printf("The signature has been verified.\n");

}

else {

printf("Signature not validated!\n");

}

return 1;

}

int main(int argc, char \*argv[])

{

if (argc < 4) return 1;

HCRYPTPROV hProv;

HCRYPTHASH hHash;

HCRYPTKEY hSessKey;

HCRYPTKEY hPubKey;

LPTSTR pszPubKey = argv[2];

LPTSTR pszHData = argv[3];

LPTSTR pszSignature = argv[4];

GethProv(hProv);

GenPrivateKey(hProv, hSessKey);

ImportKey(hProv, pszPubKey, hPubKey);

GetHash(hProv, hHash, hSessKey, pszHData);

VerifyHash(hHash, hPubKey, pszSignature);

if (hHash) CryptDestroyHash(hHash);

if (hSessKey) CryptDestroyKey(hSessKey);

if (hPubKey) CryptDestroyKey(hPubKey);

if (hProv) CryptReleaseContext(hProv, 0);

system("pause");

return 0;

}





