Министерство образования и науки РФ
Белгородский Государственный Технологический Университет им. В. Г. Шухова.

Кафедра программного обеспечения вычислительной техники и автоматизированных систем.

Лабораторная работа №3

«Создание криптографических сообщений с использованием интерфейса Microsoft CryptoAPI и цифровых сертификатов X.509»

Выполнил студент группы ПВ – 41

Котов Роман

Принял: Смышляев А. Г.

*Защита:*_____

Белгород 2015

Лабораторная работа №3
Создание криптографических сообщений с использованием интерфейса Microsoft CryptoAPI и цифровых сертификатов X.509

Цель лабораторной работы:

Ознакомиться со структурой и форматами представления сертификатов открытых ключей, способами их создания и импортирования в систему, а также получить навыки в создании криптографических сообщений средствами интерфейса Microsoft CryptoAPI.

Задание

1. С помощью криптографического пакета OpenSSL создать:
   - ключевую пару алгоритма RSA с длиной ключа 2048 бит и соответствующий ей самоподписанный сертификат центра сертификации;
   - две ключевые пары алгоритма RSA с длиной ключа 2048 бит и соответствующие им сертификаты в формате PKCS#12 для двух пользователей – участников процесса обмена криптографическими сообщениями. Сертификаты должны быть подписаны закрытым ключом центра сертификации.
2. Установить в системе созданные сертификаты. В отчет внести последовательность команд OpenSSL, использованных для создания сертификатов центра сертификации и пользователей.
3. Разработать на языке программирования C/C++ с использованием средств криптографического интерфейса Microsoft CryptoAPI консольное или оконное приложение, выполняющее создание криптографического сообщения по стандарту CMS из указанного пользователем файла и дальнейшего его расшифрования. Криптографическое сообщение должно содержать данные, зашифрованные алгоритмом AES-128 в режиме CBC и электронную подпись, созданную с помощью алгоритма RSA. Приложение должно предлагать пользователю перечень имен субъектов сертификатов, установленных в хранилище «Личное», и принимать его выбор имен отправителя и получателя криптографического сообщения. Перед созданием сообщения необходимо верифицировать ЭП в составе выбранных сертификатов и проверить соответствие текущей даты периоду, заданному в их составе. Созданное криптографическое сообщение необходимо выгружать в указанный пользователем файл и загружать из него в память для расшифрования. Расшифрованное сообщение также необходимо выгружать в файл, указанный пользователем.

Выполнение

Создание самоподписанного сертификата выполним с помощью команды:

```
req -x509 -newkey rsa:2048 -days 730 -keyout ca_test_key.pem -out
ca_test_cert.pem
```

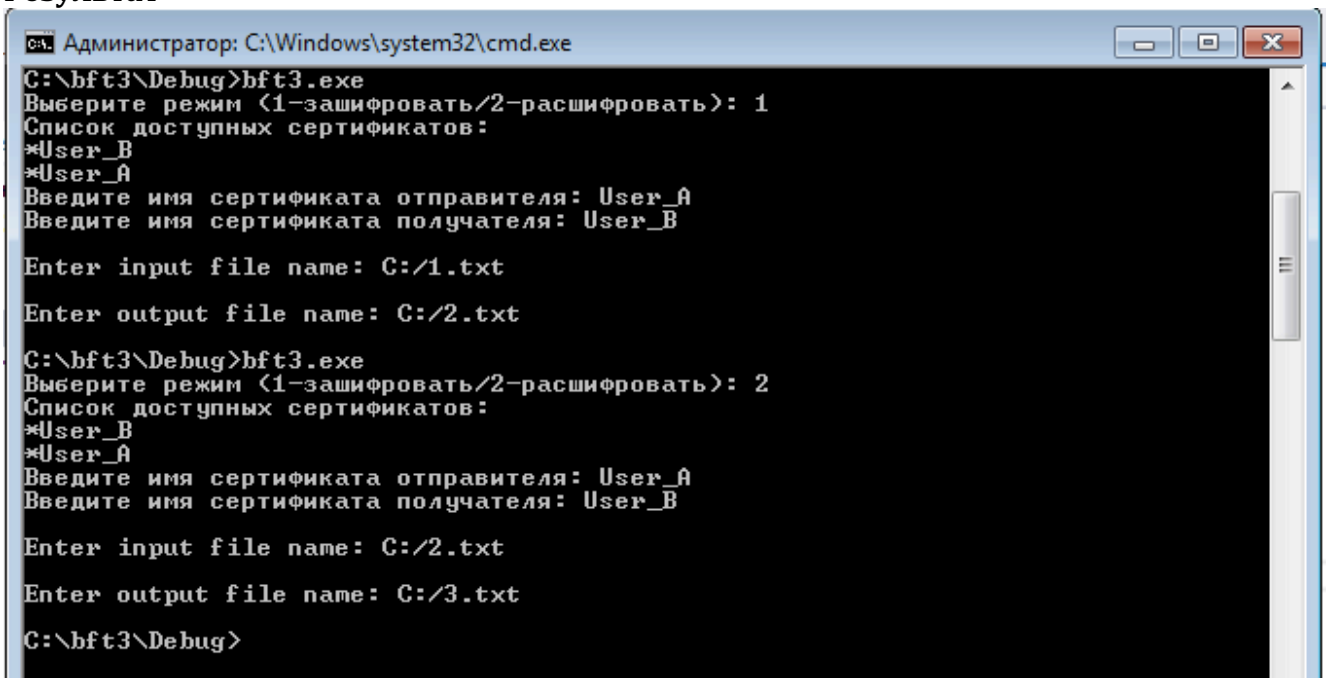
Создание ключевой пары и сертификата пользователя проводится в два этапа. Сначала командой req создается запрос на сертификацию с одновременной генерацией ключевой пары, а затем с помощью команды ca он подписывается закрытым ключом ЦС

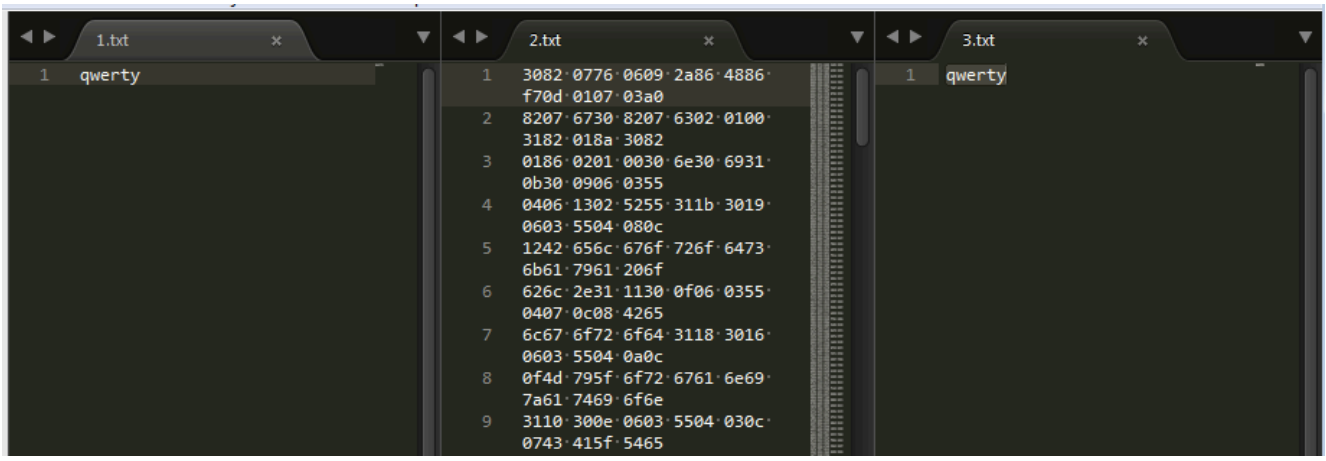
```
req -newkey rsa:2048 -keyout User_A_key.pem -out User_A_req.pem
ca    -md    sha256    -keyfile    ca_test_key.pem    -cert
ca_test_cert.pem -in User_A_req.pem -out User_A_cert.pem
```

Теперь на базе сертификата пользователя в формате PEM и его закрытого ключа создадим сертификат в формате PKCS#12:

```
pkcs12 -export -in User_A_cert.pem -inkey User_A_key.pem -out
User_A_cert.p12
```

Результат


Администратор: C:\Windows\system32\cmd.exe
C:\bft3\Debug>bft3.exe
Выберите режим (1-зашифровать/2-расшифровать): 1
Список доступных сертификатов:
*User_B
*User_A
Введите имя сертификата отправителя: User_A
Введите имя сертификата получателя: User_B
Enter input file name: C:/1.txt
Enter output file name: C:/2.txt
C:\bft3\Debug>bft3.exe
Выберите режим (1-зашифровать/2-расшифровать): 2
Список доступных сертификатов:
*User_B
*User_A
Введите имя сертификата отправителя: User_A
Введите имя сертификата получателя: User_B
Enter input file name: C:/2.txt
Enter output file name: C:/3.txt
C:\bft3\Debug>


1.txt
1 qwerty
2.txt
1 3082 0776 0609 2a86 4886 f70d 0107 03a0
2 8207 6730 8207 6302 0100 3182 018a 3082
3 0186 0201 0030 6e30 6931 0b30 0906 0355
4 0406 1302 5255 311b 3019 0603 5504 080c
5 1242 656c 676f 726f 6473 6b61 7961 206f
6 626c 2e31 1130 0f06 0355 0407 0c08 4265
7 6c67 6f72 6f64 3118 3016 0603 5504 0a0c
8 0f4d 795f 6f72 6761 6e69 7a61 7469 6f6e
9 3110 300e 0603 5504 030c 0743 415f 5465
3.txt
1 qwerty

Листинг
Задание А

```
#pragma comment(lib, "crypt32.lib")

#include <stdio.h>
#include <windows.h>
#include <wincrypt.h>
#include <locale.h>
#include <tchar.h>

#define IF_R(condition) if (condition) { printf("Ошибка: %#08x\n", GetLastError()); return 0; }
#define IF_G(condition, gotoLabelName) if (condition) { printf("Ошибка: %#08x\n", GetLastError()); goto gotoLabelName; }
#define IF_M_G(condition, msg, gotoLabelName) if (condition) { printf("%s\n", msg); goto gotoLabelName; }

 #define MY_ENCODING_TYPE (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)

void EnterFileNames(char *inFileName, char *outFileName);
BOOL OpenFiles(FILE **inFile, FILE **outFile, char *inFileName, char *outFileName, int *filesize);
void CloseFiles(FILE **inFile, FILE **outFile);
BOOL OpenCert(char *subject, PCCERT_CONTEXT *pccert_ctx, HCERTSTORE store);
BOOL CertValidate(HCERTSTORE store, PCCERT_CONTEXT pccert_ctx);
BOOL Encode(FILE *inFile, FILE *outFile, int inputBufSize, PCCERT_CONTEXT certA, PCCERT_CONTEXT certB);
BOOL Decode(FILE *inFile, FILE *outFile, int inputBufSize, HCERTSTORE store, PCCERT_CONTEXT certA, PCCERT_CONTEXT certB);

char inFileName[255];
char outFileName[255];

int main(int argc, char* argv[]) {
	setlocale(LC_ALL, "");
	int mode = 0;
	while (mode < 1 || mode > 2) {
		printf("Выберите режим (1-зашифровать/2-расшифровать): "); scanf("%d", &mode);
	}
	// ------------
	HCERTSTORE hStoreMy = NULL;
	IF_R(!(hStoreMy = CertOpenSystemStore(NULL, TEXT("MY"))));

	printf("Список доступных сертификатов:\n");
	PCCERT_CONTEXT pCert = NULL;
	while (pCert = CertEnumCertificatesInStore(hStoreMy, pCert)) {
		TCHAR szNameString[128];
		if (CertGetNameString(
			pCert, CERT_NAME_SIMPLE_DISPLAY_TYPE, 0, NULL,
			szNameString, 128) > 1) {
			_tprintf(TEXT("*%s\n"), szNameString);
		}
	}

	fflush(stdin);
	PCCERT_CONTEXT pCertA = NULL;
	IF_G(!OpenCert("отправителя", &pCertA, hStoreMy), closeRootStore);
	PCCERT_CONTEXT pCertB = NULL;
	IF_G(!OpenCert("получателя", &pCertB, hStoreMy), closeACert);

	HCERTSTORE hStoreRoot = NULL;
	IF_G(!(hStoreRoot = CertOpenSystemStore(NULL, TEXT("ROOT"))), closeMyStore);

	IF_M_G(!CertValidate(hStoreRoot, pCertA), "Ошибка проверки валидности сертификата или сертификат не валиден", end);
	IF_M_G(!CertValidate(hStoreRoot, pCertB), "Ошибка проверки валидности сертификата или сертификат не валиден", end);

	// ------------
	FILE *inFile;
	FILE *outFile;
	int inFileSize;

	EnterFileNames(inFileName, outFileName);
	OpenFiles(&inFile, &outFile, inFileName, outFileName, &inFileSize);

	if (mode == 1) {
		IF_G(!Encode(inFile, outFile, inFileSize, pCertA, pCertB), end);
	} else if (mode == 2) {
		IF_G(!Decode(inFile, outFile, inFileSize, hStoreMy, pCertA, pCertB), end);
	}

end:
	CloseFiles(&inFile, &outFile);
closeACert:
	CertFreeCertificateContext(pCertA);
closeRootStore:
	CertCloseStore(hStoreRoot, 0);
closeMyStore:
	CertCloseStore(hStoreMy, 0);

	return 0;
}
// ----------------EN/DE--------------
BOOL Encode(FILE *inFile, FILE *outFile, int inputBufSize, PCCERT_CONTEXT certA, PCCERT_CONTEXT certB) {
	BYTE *inputBuf = (BYTE*)calloc(inputBufSize, 1);
	if (!fread(inputBuf, inputBufSize, 1, inFile)) {
		return FALSE;
	}
	BYTE *outputBuf;
	DWORD outputBufSize;

	CRYPT_SIGN_MESSAGE_PARA SignPara = {
			sizeof(CRYPT_SIGN_MESSAGE_PARA) };
```

```
	SignPara.dwMsgEncodingType = MY_ENCODING_TYPE;
	SignPara.pSigningCert = certA;
	SignPara.HashAlgorithm.pszObjId = szOID_RSA_SHA256RSA;
	SignPara.cMsgCert = 1;
	SignPara.rgpMsgCert = &certA;

	CRYPT_ENCRYPT_MESSAGE_PARA EncryptPara = {
		sizeof(CRYPT_ENCRYPT_MESSAGE_PARA) };
	EncryptPara.dwMsgEncodingType = MY_ENCODING_TYPE;
	EncryptPara.ContentEncryptionAlgorithm.pszObjId =
		szOID_NIST_AES128_CBC;

	if (!CryptSignAndEncryptMessage(
		&SignPara,
		&EncryptPara,
		1,
		&certB,
		inputBuf,
		inputBufSize,
		NULL,
		&outputBufSize
		)) {
		return FALSE;
	}

	outputBuf = (BYTE*)calloc(outputBufSize, 1);

	if (!CryptSignAndEncryptMessage(
		&SignPara,
		&EncryptPara,
		1,
		&certB,
		inputBuf,
		inputBufSize,
		outputBuf,
		&outputBufSize
		)) {
		return FALSE;
	}

	if (!fwrite(outputBuf, outputBufSize, 1, outFile)) {
		return FALSE;
	}

	return TRUE;
}

BOOL Decode(FILE *inFile, FILE *outFile, int inputBufSize, HCERTSTORE store, PCCERT_CONTEXT certA, PCCERT_CONTEXT certB) {
	BYTE *inputBuf = (BYTE*)calloc(inputBufSize, 1);
	if (!fread(inputBuf, inputBufSize, 1, inFile)) {
		return FALSE;
	}
	BYTE *outputBuf;
	DWORD outputBufSize;

	CRYPT_DECRYPT_MESSAGE_PARA DecryptPara = {
			sizeof(CRYPT_DECRYPT_MESSAGE_PARA) };
	DecryptPara.dwMsgAndCertEncodingType = MY_ENCODING_TYPE;
	DecryptPara.cCertStore = 1;
	DecryptPara.rghCertStore = &store;

	CRYPT_VERIFY_MESSAGE_PARA VerifyPara = {
		sizeof(CRYPT_VERIFY_MESSAGE_PARA) };
	VerifyPara.dwMsgAndCertEncodingType = MY_ENCODING_TYPE;

	if (!CryptDecryptAndVerifyMessageSignature(
		&DecryptPara,
		&VerifyPara,
		0,
		inputBuf,
		inputBufSize,
		NULL,
		&outputBufSize,
		&certB,
		&certA)) {
		return FALSE;
	}

	outputBuf = (BYTE*)calloc(outputBufSize, 1);

	if (!CryptDecryptAndVerifyMessageSignature(
		&DecryptPara,
		&VerifyPara,
		0,
		inputBuf,
		inputBufSize,
		outputBuf,
		&outputBufSize,
		&certB,
		&certA)) {
		return FALSE;
	}

	if (!fwrite(outputBuf, outputBufSize, 1, outFile)) {
		return FALSE;
	}
```

```
	return TRUE;
}
// ----------------CERTS----------------
BOOL OpenCert(char *subject, PCCERT_CONTEXT *pccert_ctx, HCERTSTORE store) {
	TCHAR certName[255] = TEXT("");
	printf("Введите имя сертификата %s: ", subject);
	_tscanf(TEXT("%s"), certName);
	PCCERT_CONTEXT result = CertFindCertificateInStore(store, MY_ENCODING_TYPE, 0, CERT_FIND_SUBJECT_STR, certName, NULL);
	if (result != NULL) {
		*pccert_ctx = result;
		return TRUE;
	}
	return FALSE;
}

BOOL CertValidate(HCERTSTORE store, PCCERT_CONTEXT pccert_ctx) {
	DWORD checkFlags = CERT_STORE_TIME_VALIDITY_FLAG | CERT_STORE_SIGNATURE_FLAG;

	if (!(CertGetIssuerCertificateFromStore(
		store,
		pccert_ctx,
		NULL,
		&checkFlags
		))) {
			return FALSE;
	}

	if (checkFlags == 0) {
		return TRUE;
	}

	return FALSE;
}
// ----------------FILES----------------
void EnterFileNames(char *inFileName, char *outFileName) {
	printf("\nEnter input file name: ");
	scanf("%s", inFileName);
	printf("\nEnter output file name: ");
	scanf("%s", outFileName);
}

BOOL OpenFiles(FILE **inFile, FILE **outFile, char *inFileName,
	char *outFileName, int *filesize) {
	*inFile = fopen(inFileName, "rb");
	if (*inFile == NULL) {
		printf("Input file not existing\n");
		return FALSE;
	}

	fseek(*inFile, 0, SEEK_END);
	*filesize = ftell(*inFile);
	rewind(*inFile);

	*outFile = fopen(outFileName, "wb");
	if (*outFile == NULL) {
		fclose(*inFile);
		*inFile = NULL;
		printf("Error create output file\n");
		return FALSE;
	}

	return TRUE;
}

void CloseFiles(FILE **inFile, FILE **outFile) {
	fclose(*inFile);
	fclose(*outFile);

	*inFile = NULL;
	*outFile = NULL;
}
```