# Лабораторная работа 3

**Средства создания и использования сертификатов в операционных системах и системах программирования**

**Содержание задания**

1. Создать проект приложения Microsoft Visual Studio для прототипа системы электронного документооборота с использованием сертификатов. Приложение должно обеспечивать обмен между пользователями подписанными текстовыми документами аналогично лабораторной работе 1. Приложение должно позволять редактировать текстовые документы, а также сохранять и загружать их вместе с электронной подписью. Для вычисления и проверки подписи используются закрытые ключи и сертификаты участников обмена сообщениями.
2. Примеры главной формы и интерфейса приложения приведены на рис. 1-3. Нередактируемая строка под надписью «Имя пользователя» предназначена для отображения имени владельца сертификата, используемого для вычисления и проверки электронной подписи (пока имя пользователя не задано, создание или редактирование подписанных документов невозможны). Поле с текстовым редактором под не редактируемой строкой предназначено для ввода и редактирования текстов подписанных документов. Кнопка «Выбрать сертификат» на главной форме используется для переключения на новый сертификат (нового пользователя). Эта кнопка дублирует команду «Выбрать» меню «Управление сертификатами». Кнопки «Загрузить документ» и «Сохранить документ» дублируют команды «Загрузить» и «Сохранить» меню «Файл».
3. Команда «Создать» меню «Файл» предназначена для создания нового документа (очищается содержимое поля с текстовым редактором, а в заголовок формы помещается текст «Подписанный документ»).

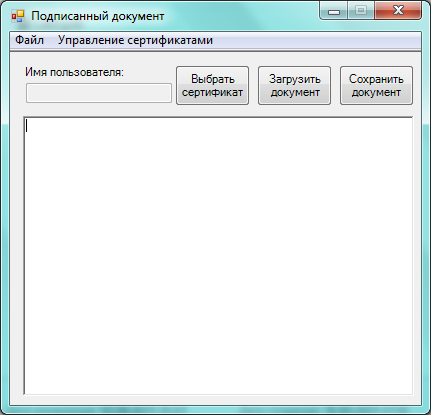


Рис. 1. Пример главной формы приложения.

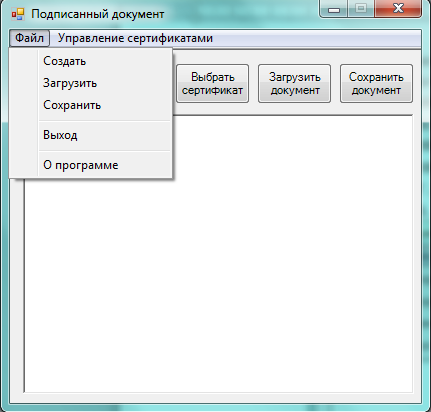


Рис. 2. Пример состава меню «Файл» приложения.

1. Команда «Сохранить» меню «Файл» предназначена для сохранения созданного (отредактированного) документа в виде массива байт вместе с вычисляемой электронной подписью и сертификатом автора документа в папке и файле (рекомендуется с расширением sd), выбираемыми пользователем. Структура подписанного документа приведена на рис. 4. Алгоритмы электронной подписи и хеширования массива байт документа выбираются из табл. 1 в соответствии с номером варианта студента. Если пользователь пытается подписать и сохранить документ, не выбрав предварительно сертификат, выводится соответствующее сообщение.
2. Команда «Загрузить» меню «Файл» предназначена для чтения подписанного документа из папки и файла, выбираемых пользователем, извлечения из документа массива байт с сертификатом его автора, построения цепочки сертификации для извлеченного сертификата и ее проверки, извлечения из сертификата имени его владельца (субъекта), извлечения и проверки подписи автора под документом и отображением текста проверенного документа в поле с текстовым редактором на главной форме. Имя автора проверенного документа должно отображаться в заголовке главной формы после слов «Подписанный документ». Если документ с заданным именем не найден, проверка не подтвердила подлинность сертификата автора документа, не подтверждается подпись под документом, приложение должно выводить соответствующие сообщения.
3. Команда «Выход» меню «Файл» предназначена для завершения работы приложения. Команда «О программе» меню «Файл» предназначена для вывода формы с информацией об авторе приложения (фамилии и инициалах студента, номере учебной группы и номере варианта).

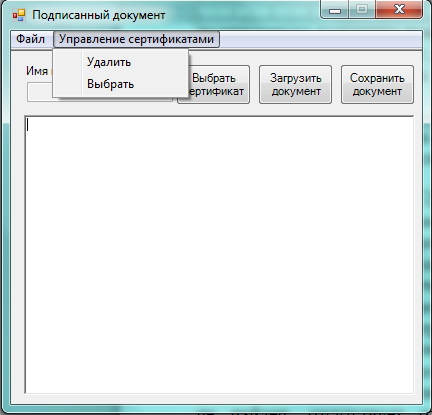


Рис. 3. Пример состава меню «Управление ключами» приложения.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Длина сертификата | Длина подписи | Сертификат  пользователя | Электронная подпись | Текст документа |

Рис. 4. Структура подписанного документа.

1. Команда «Удалить» меню «Управление сертификатами» предназначена для удаления сертификата и закрытого ключа пользователя из его хранилища личных сертификатов.
2. Команда «Выбрать» меню «Управление сертификатами» предназначена для отображения стандартного диалога выбора сертификата в хранилище личных сертификатов и, при выборе пользователем его сертификата, отображения имени субъекта в не редактируемой строке для имени пользователя.
3. После завершения разработки приложения провести его тестирование. Рекомендуемый порядок действий при тестировании:
   1. импортировать (при необходимости) сертификаты, созданные при выполнении лабораторной работы 2, из файлов формата PFX с помощью соответствующей команды контекстного меню Проводника Windows;
   2. создать небольшой текстовый документ и сохранить его;
   3. передать файл с подписанным документом другим пользователям;
   4. получить и импортировать сертификаты других пользователей;
   5. получить и прочитать документы других пользователей;
   6. удалить свой сертификат.
4. Предъявить проект с разработанным приложением преподавателю для проверки и защиты лабораторной работы.

Таблица 1

Варианты индивидуальных заданий

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| №№ вариантов | Алгоритм хеширования документа | Алгоритм подписи документа | №№ вариантов | Алгоритм хеширования документа | Алгоритм подписи документа |
| 1 | SHA1 | RSA | 26 | MD5 | RSA |
| 2 | RIPEMD | RSA | 27 | MD5 | RSA |
| 3 | SHA1 | RSA | 28 | SHA1 | DSA |
| 4 | RIPEMD | RSA | 29 | RIPEMD | DSA |
| 5 | MD5 | RSA | 30 | SHA1 | DSA |
| 6 | MD5 | RSA | 31 | MD5 | RSA |
| 7 | MD5 | RSA | 32 | SHA1 | DSA |
| 8 | MD5 | RSA | 33 | MD5 | RSA |
| 9 | SHA1 | DSA | 34 | SHA1 | DSA |
| 10 | RIPEMD | DSA | 35 | MD5 | RSA |
| 11 | SHA1 | DSA | 36 | SHA1 | RSA |
| 12 | SHA1 | RSA | 37 | RIPEMD | RSA |
| 13 | RIPEMD | RSA | 38 | RIPEMD | RSA |
| 14 | SHA1 | RSA | 39 | RIPEMD | RSA |
| 15 | RIPEMD | RSA | 40 | MD5 | RSA |
| 16 | MD5 | RSA | 41 | MD5 | RSA |
| 17 | MD5 | RSA | 42 | MD5 | RSA |
| 18 | MD5 | RSA | 43 | SHA1 | RSA |
| 19 | MD5 | RSA | 44 | SHA1 | DSA |
| 20 | SHA1 | RSA | 45 | SHA1 | DSA |
| 21 | RIPEMD | RSA | 46 | SHA1 | DSA |
| 22 | SHA1 | RSA | 47 | SHA1 | DSA |
| 23 | RIPEMD | RSA | 48 | MD5 | RSA |
| 24 | MD5 | RSA | 49 | SHA1 | RSA |
| 25 | MD5 | RSA | 50 | RIPEMD | RSA |

# Примечание. Если подписываемый или проверяемый документ хешируется по алгоритму RIPEMD, то после получения его хеша при вызове метода для вычисления или проверки электронной подписи должен указываться алгоритм SHA1.

# Рекомендуемые средства языка C#

1. Класс для хранилища сертификатов X509Store (пространство имен System.Security.Cryptography.X509Certificates). Конструктор:

X509Store ( string storeName, StoreLocation storeLocation); /\* создание объекта для хранилища storeName из расположения storeLocation: StoreLocation. CurrentUser для профиля текущего пользователя или StoreLocation. LocalMachine для профиля локального компьютера \*/

Методы:

void Open (OpenFlags flags); /\* открытие хранилища сертификатов; значением параметра flags может быть комбинация флагов перечисления OpenFlags: OpenExistingOnly – открытие только существующего хранилища, ReadOnly – открытие хранилища только для чтения \*/

void Close (); // закрытие хранилища

void Remove (X509Certificate2 certificate); /\* удаление сертификата certificate из хранилища \*/

Свойство для чтения:

X509Certificate2Collection Certificates // коллекция сертификатов из хранилища

1. Класс для коллекции сертификатов X509Certificate2Collection (пространство имен System.Security.Cryptography.X509Certificates). Свойство для чтения:

int Count // количество сертификатов в хранилище

Получение сертификата из коллекции производится с помощью операции индексации [номер сертификата в коллекции, начиная с 0].

Метод:

void Remove (X509Certificate2 certificate); /\* удаление первого вхождения сертификата certificate из коллекции \*/

1. Класс для выполнения стандартных диалогов Windows при работе с сертификатами X509Certificate2UI (пространство имен System.Security. Cryptography.X509Certificates). Метод:

static X509Certificate2Collection SelectFromCollection (X509Certificate2Collection certificates, string title, string message, X509SelectionFlag selectionFlag); /\* выполнение диалога для выбора сертификата из коллекции certificates с заголовком title и подсказкой для пользователя message; параметр selectionFlag определяет возможность выбора только одного сертификата – должно указываться значение X509SelectionFlag. SingleSelection; возвращается коллекция выбранных пользователем сертификатов \*/

1. Класс для сертификата X.509 X509Certificate2 (пространство имен System.Security. Cryptography.X509Certificates). Конструктор:

X509Certificate2 ( byte[] rawData); /\* создание объекта на основе информации из массива байт rawData \*/

Свойства:

AsymmetricAlgorithm PrivateKey /\* объект класса с реализацией асимметричного криптоалгоритма, содержащий связанный с сертификатом закрытый ключ \*/

PublicKey PublicKey /\* для получения объекта класса PublicKey, содержащего открытый ключ из сертификата \*/

Свойство класса PublicKey:

AsymmetricAlgorithm Key /\* для получения объекта класса с реализацией асимметричного криптоалгоритма, содержащего связанный с сертификатом закрытый ключ \*/

Свойства PrivateKey и PublicKey. Key используются для создания в приложении объектов классов с реализацией алгоритмов для вычисления и проверки электронной подписи, например:

X509Certificate2 Cert;

RSACryptoServiceProvider cspRSA = (RSACryptoServiceProvider) Cert.PrivateKey;

RSACryptoServiceProvider cspRSA = (RSACryptoServiceProvider) Cert.PublicKey. Key;

Методы класса X509Certificate2:

string GetNameInfo (X509NameType nameType, bool forIssuer); /\* получение имени субъекта (владельца сертификата); значение параметра nameType, равное X509NameType. SimpleName определяет получение простого имени субъекта; значение параметра forIssuer определяет необходимость включения в результат имени издателя сертификата \*/

bool Verify(); /\* построение цепочки сертификации и проверка ее действительности с использованием базовой политики проверки \*/

1. Класс для цепочки сертификации X509Chain (пространство имен System.Security.Cryptography.X509Certificates). Конструктор без параметров.

Свойство:

X509ChainPolicy ChainPolicy // политика создания цепочки сертификации

Свойство класса X509ChainPolicy:

X509RevocationMode RevocationMode /\* режим проверки статуса отзыва сертификатов в цепочке: при включении в проверяемую цепочку сертификатов, выданных тестовым удостоверяющим центром значением этого свойства должно быть X509RevocationMode. NoCheck – проверка статуса отзыва не производится \*/

Метод класса X509Chain:

bool Build ( X509Certificate2 certificate); /\* создание цепочки для сертификата certificate и проверка ее действительности с использованием политики, определенной в свойстве ChainPolicy \*/

**Рекомендуемые средства языка Java**

1. Класс для хранилища ключей и сертификатов java.security.KeyStore. Методы:

static KeyStore getInstance (String type); /\* создание хранилища ключей указанного в type типа: PKCS12 (совместимость с закрытыми ключами и сертификатами в формате PFX), JKS (специальный для Java тип), JCEKS (расширение JKS ч усиленной защитой закрытых ключей), PKCS11 (использование аппаратного устройства для хранилища ключей), DKS (коллекция хранилищ ключей), Windows-MY (системное хранилище операционной системы Windows для личных сертификатов), BKS (часто используется в мобильных приложениях) \*/

void load (null, null); /\* создание пустого хранилища ключей или его загрузка не из входного потока, а, например, из системного хранилища Windows \*/

void load(InputStream stream, char[] password); /\* загрузка хранилища ключей из входного потока stream с использованием пароля password для контроля целостности \*/

Enumeration aliases (); /\* получение списка всех псевдонимов, связанных с этим объектом хранилища ключей (для хранилища личных сертификатов в Windows псевдонимом является имя субъекта в сертификате) \*/

Класс Enumeration имеет методы

boolean hasMoreElements (); /\* проверка наличия в списке хотя бы одного элемента \*/

E nextElement (); /\* получение следующего элемента списка (E – тип элемента списка) \*/

boolean isCertificateEntry (String alias); /\* проверка, что элемент с псевдонимом alias хранилища ключей является сертификатом \*/

Certificate getCertificate (String alias); /\* получение сертификата в хранилище ключей по псевдониму alias \*/

Certificate[] getCertificateChain(String alias); /\* получение цепочки сертификатов для заданного псевдонима \*/

boolean isKeyEntry (String alias); /\* проверка связи с закрытым ключом элемента хранилища ключей с псевдонимом alias \*/

KeyStore.Entry getEntry (String alias, KeyStore.PasswordProtection protParam); /\* получение элемента хранилища ключей, связанного с закрытым ключом, по псевдониму alias и паролю protParam для доступа к элементу; результат должен быть приведен к типу KeyStore.PrivateKeyEntry \*/

В классе KeyStore.PasswordProtection (java.security.KeyStore.PasswordProtection) определен конструктор с параметром типа char[], задающим значение пароля для доступа к хранилищу ключей.

Класс KeyStore.PrivateKeyEntry содержит метод

PrivateKey getPrivateKey (); // создание объекта класса закрытого ключа

1. Класс java.security.cert.CertificateFactory для создания экземпляров сертификатов (объектов класса Certificate) из двоичных данных сертификатов с кодировками X.509. Методы:

certificateFactory getInstance ("X.509"); /\* создание экземпляра класса CertificateFactory \*/

Certificate generateCertificate (InputStream inStream); /\* генерация объекта сертификата и инициализация его данными, считанными из входного потока inStream \*/

CertPath generateCertPath(List<Certificate> certificates); /\* создание цепочки сертификатов из списка certificates \*/

Класс CertPath (java.security.cert.CertPath) представляет цепочку сертификатов.

Класс java.io. ByteArrayInputStream представляет входной поток, использующий в качестве источника данных массив байт. Конструкторы:

ByteArrayInputStream (byte[] buf); /\* buf – массив байт, из которого производится считывание данных \*/

ByteArrayInputStream (byte[] buf, int offset, int length); /\* данные длиной length считываются из массива buf со смещением offset \*/

1. Класс для сертификата X.509 java.security.cert.X509Certificate − подкласс класса Certificate. Методы:

byte[] getEncoded (); // получение сертификата в виде массива байт

PublicKey getPublicKey (); // получение открытого ключа из сертификата

X500Principal getSubjectX500Principal (); // получение имени субъекта

Класс X500Principal имеет метод

String getName (); // получение различимого имени субъекта

X500Principal getIssuerX500Principal (); // получение имени издателя сертификата

BigInteger getSerialNumber (); // получение серийного номера сертификата

1. Класс java.security.Signature представляет алгоритмы электронной подписи (ЭП) для двоичных данных. Методы:

static Signature getInstance (String algorithm); /\* создание объекта класса с реализацией криптоалгоритмов, заданных параметром algorithm: SHA1withDSA, SHA1withRSA, SHA256withRSA, SHA384withRSA, SHA512withRSA, MD2withRSA, MD5withRSA, SHA1withECDSA, SHA256withECDSA, SHA384withECDSA, SHA512withECDSA \*/

void initSign (PrivateKey privateKey); /\* инициализация объекта класса для вычисления ЭП (privateKey задает закрытый ключ ЭП) \*/

void initVerify (PublicKey publicKey); /\* инициализация объекта класса для проверки ЭП (publicKey задает открытый ключ ЭП) \*/

void update (byte[] data) ; /\* хеширование подписываемых (проверяемых) данных из буфера data \*/

byte[] sign (); // получение ЭП для хешированных данных

boolean verify (byte[] signature); /\* проверка ЭП signature для хешированных данных \*/

**Рекомендуемые средства языка Python**

1. Работа с сертификатами открытых ключей и закрытыми ключами в файл PFX.

import OpenSSL

p12 = OpenSSL.crypto.load\_pkcs12(pfx, password) # получение объекта  
# класса PKCS12 из данных pfx , прочитанных из файла PFX, с   
# расшифрованием закрытого ключа с помощью парольной фразы password

# (массива байт)

# извлечение сертификата, открытого и закрытого ключей  
cert = p12.get\_certificate() # возвращает объект класса X509, имеющего метод  
# get\_subject без параметров, который возвращает объект класса X509Name,  
# имеющего свойство commonName, хранящее имя владельца сертификата

publickey = OpenSSL.crypto.dump\_certificate(OpenSSL.crypto.FILETYPE\_ASN1, cert) # получение открытого ключа из сертификата

privatekey = p12.get\_privatekey() # получение закрытого ключа

sig = OpenSSL.crypto.sign(privatekey, bytes(message,'UTF-8'), fun) # получение  
# электронной подписи для сообщения message с помощью закрытого ключа  
# privatekey и хеш-функции с именем fun

OpenSSL.crypto.verify(cert, sig, bytes(message2,'UTF-8'), fun) # проверка  
# электронной подписи sig для сообщения message с помощью открытого   
# ключа из сертификата cert и хеш-функции с именем fun (при неудачной  
# проверке генерируется исключение)

1. Использование сертификатов из системного хранилища Windows (первый способ)

from win32 import win32crypt

from win32.lib import win32cryptcon

from Crypto.PublicKey import RSA

from Crypto.PublicKey import DSA

from Crypto.Signature import pkcs1\_15

from Crypto.Signature import DSS

from Crypto.Hash import MD5

from Crypto.Hash import SHA1

# получение доступа к хранилищу личных сертификатов пользователя   
# Windows

store2 = win32crypt.CertOpenSystemStore('MY', None)

# получение списка контекстов сертификатов в хранилище

store2.CertEnumCertificatesInStore():

# получение простого имени владельца сертификата (субъекта) по свойству   
# Subject контекста сертификата cert1

win32crypt.CertNameToStr(cert1.Subject,

win32cryptcon.CERT\_SIMPLE\_NAME\_STR)

# получение сертификата в кодировке DER

certDER = cert1.CertEncoded

# получение из контекста сертификата cert1кортежа:   
# (тип ключа, дескриптор криптопровайдера)

privkey = cert1.CryptAcquireCertificatePrivateKey(0)

# инициализация объекта хеширования (можно указать другой алгоритм  
# хеширования в виде CALG\_\*)

chash = privkey[1].CryptCreateHash(win32cryptcon.CALG\_MD5, None, 0)

# хеширование подписываемого сообщения

chash.CryptHashData(bytes(message,'UTF-8'), 0)

# вычисление подписи

sign = chash.CryptSignHash(privkey[0],0)[::-1]

# хеширование проверяемого сообщения (можно указать другой алгоритм  
# хеширования)

myhash = MD5.new(bytes(message2,'UTF-8'))

# проверка подписи по алгоритму RSA (открытый ключ импортируется из

# сертификата в кодировке DER)  
# при неудачной проверке генерируется исключение

pkcs1\_15.new(RSA.import\_key(certDER).verify(myhash, sign)

# для DSA проверка всегда неудачная, если используется  
# сертификат с ключом SHA1-DSA длиной 1024 бита, созданный с помощью  
# утилиты makecert

DSS.new(DSA.import\_key(certDER), 'fips-186-3').verify(myhash, sign)

1. Использование сертификатов из системного хранилища Windows (второй способ):

from ctypes import \*

import wincertstore

store = wincertstore.CertSystemStore("MY")

store.itercerts() # коллекция сертификатов в хранилище

cert.get\_name() # имя владельца сертификата

cert.get\_pem() # сертификат в кодировке PEM

# получение сертификата (объекта класса X509) из его представления   
# в кодировке PEM

cert2 = OpenSSL.crypto.load\_certificate(OpenSSL.crypto.FILETYPE\_PEM,

cert.get\_pem())

# вычисление и проверка электронной подписи с помощью CryptoAPI

# получение дескриптора криптопровайдера, поддерживающего контейнер  
# ключей, по контексту сертификата

lib = cdll.crypt32 # доступ к библиотеке CryptoAPI v2

phCryptProv = c\_void\_p() # буфер для приема дескриптора криптопровайдера

pdwKeySpec = c\_uint() # буфер для приема спецификации пары ключей

pfCallerFreeProvOrNCryptKey = c\_bool() # буфер для приема результата

lib.CryptAcquireCertificatePrivateKey(

byref(cert),0,0,byref(phCryptProv),byref(pdwKeySpec),

byref(pfCallerFreeProvOrNCryptKey))

# создание хеш-значения подписываемых данных

lib2 = cdll.advapi32 # доступ к библиотеке CryptoAPI v1

hashobject = c\_void\_p() # буфер для приема дескриптора хеш-значения  
# 0x00008003 – код функции MD5 (другие значения – 0x00008001 MD2,  
# 0x00008002 MD4, 0x00008004 SHA1)

lib2.CryptCreateHash(c\_void\_p(phCryptProv.value),

0x00008003,None,None,byref(hashobject))

# хеширование подписываемых данных

lib2.CryptHashData(c\_void\_p(hashobject.value),

bytesmessage,len(bytesmessage),0)

# получение дескриптора пары ключей подписи

phUserKey = c\_void\_p() # буфер для приема дескриптора пары ключей

lib2.CryptGetUserKey(c\_void\_p(phCryptProv.value),

pdwKeySpec.value,byref(phUserKey))

# получение длины электронной подписи

pdwSigLen = c\_uint() # буфер для приема длины подписи

lib2.CryptSignHashA(c\_void\_p(hashobject.value),

pdwKeySpec.value,None,0,None,byref(pdwSigLen))

# получение электронной подписи

length = pdwSigLen.value # сохранение длины подписи

signature = create\_string\_buffer(b'0',length) # буфер для подписи

lib2.CryptSignHashA(c\_void\_p(hashobject.value),

pdwKeySpec.value,None,0,byref(signature),byref(pdwSigLen))

# проверка подписи

lib2.CryptCreateHash(c\_void\_p(phCryptProv.value),

0x00008003,None,None,byref(hashobject))

bytesmessage2 = bytes(message2,'UTF-8')

lib2.CryptHashData(c\_void\_p(hashobject.value),

bytesmessage2,len(bytesmessage2),0)

# функция CryptVerifySignatureA возвращает True, если подпись верна

lib2.CryptVerifySignatureA(c\_void\_p(hashobject.value),

byref(signature),pdwSigLen.value,c\_void\_p(phUserKey.value),None,0)

# освобождение дескрипторов системных объектов

lib2.CryptDestroyHash(c\_void\_p(hashobject.value))

lib2.CryptDestroyKey(c\_void\_p(phUserKey.value))

lib2.CryptReleaseContext(c\_void\_p(phCryptProv.value),0)

Используемые функции из набора CryptoAPI возвращают False в случае ошибки при их выполнении.

# использование криптографии нового поколения (CNG)

# константы для функций Windows API

CRYPT\_ACQUIRE\_ONLY\_NCRYPT\_KEY\_FLAG = 0x00040000

CERT\_NCRYPT\_KEY\_SPEC = 0xFFFFFFFF

BCRYPT\_PAD\_PKCS1 = 0x00000002

hashvaluesize = 16 # длина хеш-значения в байтах

# работа с функциями Windows API

lib = cdll.crypt32 # доступ к функциям CryptoAPI v2

lib3 = cdll.Bcrypt # доступ к функциям CNG

ib4 = cdll.Ncrypt # доступ к функциям CNG

phCryptProv = c\_void\_p() # буфер для приема дескриптора ключа CNG

pdwKeySpec = c\_ulong() # буфер для приема спецификации пары ключей

pfCallerFreeProvOrNCryptKey = c\_bool() # буфер для приема результата

# получение дескриптора ключа (CNG) по контексту сертификата

# (функция возвращает True в случае успеха)

res = c\_bool(lib.CryptAcquireCertificatePrivateKey(byref(cert),

CRYPT\_ACQUIRE\_ONLY\_NCRYPT\_KEY\_FLAG,

0,byref(phCryptProv),

byref(pdwKeySpec), byref(pfCallerFreeProvOrNCryptKey)))

# проверка получения дескриптора ключа для CNG

if bool(res) & (int(pdwKeySpec.value) == int(CERT\_NCRYPT\_KEY\_SPEC)):

# инициализация поставщика CNG

phAlgorithm = c\_void\_p() # буфер для дескриптора поставщика CNG

lib3.BCryptOpenAlgorithmProvider(

byref(phAlgorithm), "MD5", None, 0)

# хеширование подписываемых данных message

bytesmessage = bytes(message,'UTF-8')

# буфер для хеш-значения

hashvalue = create\_string\_buffer(b'0',hashvaluesize)

lib3.BCryptHash(c\_void\_p(phAlgorithm.value),

None, 0, bytesmessage,len(bytesmessage), byref(hashvalue),

hashvaluesize)

# получение длины электронной подписи

# структура для передачи в функцию CNG

class BCRYPT\_PKCS1\_PADDING\_INFO(Structure):

\_fields\_ = [("pszAlgId", c\_wchar\_p)]

hashalg = None # задание значения полю структуры

# создание экземпляра структуры (для алгоритма подписи RSA)

algid = BCRYPT\_PKCS1\_PADDING\_INFO(c\_wchar\_p(hashalg))

pdwSigLen = c\_uint() # буфер для приема длины подписи

lib4.NCryptSignHash(c\_void\_p(phCryptProv.value),

None, hashvalue, len(hashvalue),

None, 0, byref(pdwSigLen), 0)

# получение электронной подписи

signature = create\_string\_buffer(pdwSigLen.value) # буфер для подписи

lib4.NCryptSignHash(c\_void\_p(phCryptProv.value),

byref(algid), byref(hashvalue), hashvaluesize,

byref(signature), pdwSigLen.value, byref(pdwSigLen),

BCRYPT\_PAD\_PKCS1)

# хеширование проверяемых данных message2

bytesmessage = bytes(message2,'UTF-8')

# буфер для хеш-значения

hashvalue = create\_string\_buffer(b'0',hashvaluesize)

lib3.BCryptHash(c\_void\_p(phAlgorithm.value),

None, 0, bytesmessage,len(bytesmessage), byref(hashvalue),

hashvaluesize)

# проверка электронной подписи

lib4.NCryptVerifySignature(c\_void\_p(phCryptProv.value),

byref(algid), byref(hashvalue), hashvaluesize,

byref(signature), pdwSigLen.value, BCRYPT\_PAD\_PKCS1)

# функция возвращает 0 при успешной проверке подписи

# освобождение при необходимости дескриптора ключа CNG

if bool(c\_bool(pfCallerFreeProvOrNCryptKey.value)):

lib4.NCryptFreeObject(c\_void\_p(phCryptProv.value))