ЛАБОРАТОРНАЯ РАБОТА № 1

СИММЕТРИЧНОЕ ШИФРОВАНИЕ ДАННЫХ С ИСПОЛЬЗОВАНИЕМ КРИПТОГРАФИЧЕСКИХ ИНТЕРФЕЙСОВ MICROSOFT CRYPTOAPI И CRYPTOGRAPHY API: NEXT GENERATION

Цель работы: ознакомиться с понятием криптопровайдера в интерфейсе CryptoAPI и провайдера алгоритма в интерфейсе Cryptography API: Next Generation, способами подключения к ним, получить навыки выполнения базовых криптографических преобразований: хэширования и симметричного шифрования.

ОСНОВНЫЕ ПОНЯТИЯ

Интерфейс CryptoAPI

В операционных системах (ОС) компании Microsoft, начиная с Windows 95, обеспечивается реализация шифрования, генерации ключей, создания и проверки цифровых подписей и других криптографических преобразований. Эти функции необходимы для работы операционной системы, однако ими может воспользоваться и любая прикладная программа. Для этого используется интерфейс прикладного программирования *CryptoAPI*.

Все современные операционные системы Windows поддерживают криптографический интерфейс CryptoAPI 2.0. Он содержит функции, осуществляющие базовые криптографические преобразования, а также дополнительные средства, такие как функции для работы с сертификатами X.509. Набор функций для осуществления базовых криптографических преобразований также называют CryptoAPI 1.0. Ниже будут рассмотрены некоторые функции именно этого интерфейса.

Все функции интерфейса CryptoAPI 1.0 содержатся в библиотеке *advapi32.dll*. Однако они выполняют лишь ряд вспомогательных операций и вызывают библиотеку, в которой непосредственно реализованы соответствующие криптографические преобразования. Такие библиотеки называются *криптопровайдерами* (*Cryptographic Service Providers, CSP*). Криптопровайдеры имеют стандартный набор из 23 обязательных и 2 необязательных функций. Программная часть криптопровайдера представляет собой *dll*-файл, подписанный

Microsoft; периодически Windows проверяет цифровую подпись, что исключает возможность подмены криптопровайдера. В составе Windows могут быть установлены криптопровайдеры разработанные не только Microsoft, но и сторонними производителями. Сведения обо всех установленных криптопровайдерах содержатся в системном реестре (*HKLM\SOFTWARE\Microsoft\Cryptography\Defaults\Provider*).

Каждый криптопровайдер поддерживает свою ключевую базу, которая представляется набором ключевых контейнеров. Каждый контейнер имеет имя, которое присваивается ему при создании, а затем используется для работы с ним. В ключевом контейнере сохраняется долговременная ключевая информация, например пары закрытый/открытый ключ для создания и проверки электронной подписи (ЭП). Сеансовые ключи для блочного или потокового шифрования в контейнере не хранятся.

Каждый криптопровайдер характеризуется собственным именем и типом. Имя это строка символов, по которой система распознает криптопровайдера. Например, базовый криптопровайдер, имеющийся в составе ОС Windows любой версии называется «Microsoft Base Cryptographic Provider v1.0». Тип криптопровайдера это целое число (тип *DWORD*), значение которого идентифицирует набор поддерживаемых алгоритмов электронной подписи и обмена сеансовыми ключами. Криптопровайдер «Microsoft Base Cryptographic Provider v1.0» имеет тип 1, что соответствует символической константе *PROV_RSA_FULL*. В криптопровайдерах этого типа используется алгоритм RSA и для создания электронной подписи и для обмена сеансовыми ключами. В системе могут быть установлены криптопровайдеры разных типов. Также может быть установлено несколько криптопровайдеров одного типа.

В системном реестре содержится информация не только об именах криптопровайдеров и их типах, но и о том какой криптопровайдер использовать, если пользователь при вызове соответствующей функции не определил конкретное имя, а указал только требуемый ему тип. Такие криптопровайдеры называются *используемыми по умолчанию*. Например, для типа 1, в ОС, предшествующих Windows 2000, криптопровайдером по умолчанию являлся «Microsoft Base Cryptographic Provider v1.0». Начиная с Windows 2000, в состав ОС включены еще 2 криптопровайдера того же типа: «Microsoft Enhanced Cryptographic Provider» и «Microsoft Strong Cryptographic Provider», один из которых назначается криптопровайдером типа 1 по умолчанию (обычно «Microsoft Strong Cryptographic Provider»).

Криптопровайдеры Microsoft 1-го типа поддерживают уже устаревшие на данный момент алгоритмы шифрования RC2, RC4, DES, хэширования MD5, SHA-1. Современные алгоритмы: шифрования AES (с длинами ключа 128, 192, 256 бит) и хэширования SHA-2 (с длинами хэш-кода 256, 384, 512 бит) поддерживаются только криптопровайдерами типа 24 (*PROV_RSA_AES*). Начиная с версии Windows XP SP3, в состав операционных систем Microsoft включается единственный криптопровайдер этого типа «Microsoft Enhanced RSA and AES Cryptographic Provider». Он, также как и криптопровайдеры типа 1, использует алгоритм RSA для обмена ключами и создания ЭП. Расположены при этом все вышеперечисленные криптопровайдеры в составе одного файла – *rsaenh.dll*.

Для использования функции CryptoAPI в прикладной программе, разработчик должен объявить ее в программе как внешнюю, с указанием файла, в котором она находится. Чтобы облегчить процесс разработки, объявления необходимых типов, символических констант (таких как PROV_RSA_AES) и прототипов функций объединили в заголовочном файле *wincrypt.h*, который вошел в состав пакета Platform (Windows) SDK. По умолчанию пакет устанавливается в составе таких сред программирования, как C++ Builder и Visual Studio. Также он доступен для бесплатной загрузки на сайте компании Microsoft. Для использования в программе функций CryptoAPI необходимо включить в ее текст директивой *#include* содержимое файла *wincrypt.h*.

Рассмотрим функции CryptoAPI, необходимые для выполнения задания. Все описания и примеры будут приведены для языка программирования C/C++, применительно к консольному приложению (в оконных приложениях отличия только в способе вывода диагностических сообщений). Применяемые типы данных определены в файлах *wincrypt.h* и *windef.h*. Последний традиционно включается в текст в числе многих файлов, необходимых для разработки приложения Win32 API, при указании директивы *#include <windows.h>*.

В рассматриваемых далее примерах будут использоваться переменные, описание которых приведено ниже:

```
HCRYPTPROV hProv; //дескриптор криптопровайдера
HCRYPTKEY hKeyH; //дескриптор ключа, созданного из хэш-кода
HCRYPTHASH hHash; //дескриптор хэш-объекта
BYTE aBuf[512]; //буфер для данных (размер произвольный, кратный 16)
DWORD dwBuflen; //длина буфера
TCHAR szPass[100]; //строка для пароля
```

Назначение переменных будет пояснено по мере рассмотрения тех функций CryptoAPI, которые понадобятся при выполнении данной лабораторной работы.

Полное описание этих и других функций можно найти в разделе MSDN, посвященном использованию CryptoAPI (https://msdn.microsoft.com/en-us/library/windows/desktop/aa380256%28v=vs.85%29.aspx).

CryptAcquireContext

Эта функция выполняет подключение к криптопровайдеру. Ее прототип имеет вид:

```
BOOL WINAPI CryptAcquireContext(HCRYPTPROV *phProv,
                                LPCTSTR pszContainer,
                                LPCTSTR pszProvider,
                                DWORD dwProvType,
                                DWORD dwFlags);
```

В Win32 API (в том числе и в CryptoAPI) традиционно библиотечные функции, имеющие в качестве параметров строки, представлены в двух вариантах: с однобайтными ANSI-строками и «широкими» строками UNICODE. Здесь показан обобщенный прототип функции (макрос), который после обработки программы препроцессором будет заменен, в соответствии с настройками проекта, на нужный вариант (с буквой «A» или «W» в конце).

Функция возвращает дескриптор криптопровайдера в параметре *phProv*. В дальнейшем этот параметр будет передаваться другим функциям.

Параметр *pszContainer* содержит имя ключевого контейнера. Если приложение не будет использовать ключевой контейнер, то параметр задается как *NULL*. В частности, этот параметр должен быть нулевым, если приложение не будет производить экспорт/импорт сеансовых ключей. В противном случае имя контейнера необходимо задавать.

Параметр *pszProvider* содержит имя того криптопровайдера, к которому производится подключение. Параметр *dwProvType* определяет тип криптопровайдера. При вызове функции вместо него можно указывать символические константы, например, *PROV_RSA_AES*. Если в качестве параметра *pszProvider* указать *NULL*, то будет осуществлено подключение к криптопровайдеру, используемому по умолчанию для типа, указанного в параметре *dwProvType*.

Параметр *dwFlags* содержит значения флага (здесь и при описании некоторых других функций будут приведены только некоторые

значения флагов; полное описание функций можно найти в соответствующей литературе или библиотеке MSDN). Если задать его равным константе CRYPT_VERIFYCONTEXT, то получение контекста произойдет без подключения к контейнеру ключей. Для создания нового контейнера ключей с именем, заданным в параметре *pszContainer* используется флаг CRYPT_NEWKEYSET. Нулевое значение флага используется для подключению к уже существующему контейнеру.

Все функции CryptoAPI имеют логический тип (*BOOL*) и возвращают *TRUE* в случае успешного завершения и *FALSE* в противном случае. Поэтому при каждом вызове необходимо проверять возвращаемое функцией значение. В случае ошибки ее код можно узнать с помощью функции ***GetLastError***. Один из возможных вариантов обработки ошибки выполнения функции, получающей контекст без подключений к ключевому контейнеру, приведен ниже.

```
if (!CryptAcquireContext(&hProv, NULL, NULL,
    PROV_RSA_AES, CRYPT_VERIFYCONTEXT))
    {
       _tprintf(TEXT("Ошибка подключения к\
                криптопровайдеру. Код ошибки:0x%X"),
                GetLastError());
    }
```

Данный вариант вызова функции ***CryptAcquireContext*** выполнит подключение к криптопровайдеру типа 24 по умолчанию (то есть «Microsoft Enhanced RSA and AES Cryptographic Provider»).

Другой вариант вызова функции производит подключение к контейнеру ключей «my_cont», а в случае отсутствия создает его заново:

```
TCHAR szBuf[100];
if (!CryptAcquireContext(&hProv,TEXT("my_cont"),
                         NULL, PROV_RSA_AES, 0))
{
    DWORD err=GetLastError();
    if(err!=NTE_BAD_KEYSET)// Прочая ошибка
    {
         _tprintf(TEXT("Ошибка подключения к\
                  криптопровайдеру. Код ошибки:0x%X"),
                  err);
    }
    else // Создаем контейнер ключей «my_cont»
```

```
    if (!CryptAcquireContext(&hProv, TEXT("my_cont"),
                             NULL, PROV_RSA_AES,
                             CRYPT_NEWKEYSET))
    {
         _tprintf(TEXT("Ошибка создания контейнера ключей.\
                  Код ошибки:0x%X"), GetLastError());
    }
}
```

После того, как произведено подключение к нужному криптопровайдеру, для выполнения задачи симметричного шифрования необходимо получить сессионный ключ. Он может быть импортирован или сгенерирован. В свою очередь генерация ключа может быть произведена двумя способами: с помощью генератора псевдослучайных чисел или на основе хэшированной парольной фразы. Рассмотрим более подробно второй способ.

Однонаправленные хэш-функции выполняют свертку входных данных произвольной длины в битовую последовательность фиксированной длины, называемую хэш-кодом или дайджестом. При этом определение сообщения (прообраза), которое приводит к созданию данного хэш-кода, является трудновыполнимой (с точки зрения необходимых вычислительных ресурсов) задачей. А изменение всего лишь одного бита исходного сообщения приводит к кардинальным изменениям в его хэш-коде. Таким образом, пользователь, придумав парольную фразу и получив ее дайджест, использует его в качестве ключа симметричного шифрования.

В криптопровайдре «Microsoft Enhanced RSA and AES Cryptographic Provider» реализована поддержка алгоритма хэширования SHA-2 с переменной длиной дайджеста: 256, 384, или 512 бит. Для создания ключа с помощью хэш-кода используется функция ***CryptDeriveKey***. Однако перед ее вызовом необходимо, чтобы в памяти был создан хэш-объект, содержащий какой-либо хэш-код. Поэтому вначале рассмотрим функции для работы с хэш-объектом.

CryptCreateHash

Эта функция создает в памяти хэш-объект и имеет следующий прототип:

```
BOOL WINAPI CryptCreateHash(HCRYPTPROV hProv,
                            ALG_ID Algid,
                            HCRYPTKEY hKey,
                            DWORD dwFlags,
                            HCRYPTHASH *phHash);
```

Через параметр *hProv* в функцию передается дескриптор криптопровайдера. Параметр *Algid* представляет собой идентификатор используемого алгоритма хэширования. При использовании криптопровайдера типа 24 целесообразно использовать один из следующих алгоритмов: SHA-256, SHA-384, SHA-512. Соответственно определены следующие константы: *CALG_SHA_256*, *CALG_SHA_384*, *CALG_SHA_512*. Именно их можно указывать при вызове функции в качестве второго параметра. Однако их описание может отсутствовать в файле *wincrypt.h*, если он входит в состав Platform SDK устаревшей версии. В этом случае константы можно описать вручную:

```
#define CALG_SHA_256 0x800C
#define CALG_SHA_384 0x800D
#define CALG_SHA_512 0x800E
```

Параметр *hKey* содержит дескриптор ключа и используется в том случае, если должен быть создан код аутентичности сообщения (HMAC). В нашем случае этот параметр должен быть равен нулю. Параметр *dwFlags* содержит флаги и должен быть равен нулю. Функция копирует дескриптор созданного хэш-объекта по адресу, переданному через параметр *phHash*.

Приведем пример вызова функции. В этом примере и во всех последующих для краткости будут приведены только сами вызовы функций без проверки успешности их завершения. Однако в реальном приложении эти проверки необходимо осуществлять всегда, аналогично варианту, рассмотренному для функции ***CryptAcquireContext***.

```
CryptCreateHash(hProv, CALG_SHA_256, 0, 0, &hHash);
```

Функция ***CryptCreateHash*** создает пустой хэш-объект. Подать данные на вход хэш-объекта и собственно вычислить хэш-код позволяет функция ***CryptHashData***.

CryptHashData

Функция позволяет добавлять данные к объекту хэш-функции. Она может вызываться несколько раз, если данные, от которых вычисляется хэш, разбиты на блоки. Прототип функции имеет вид:

```
BOOL WINAPI CryptHashData(HCRYPTHASH hHash,
                          BYTE *pbData,
                          DWORD dwDataLen,
                          DWORD dwFlags);
```

В качестве параметра *hHash* передается дескриптор хэш-объекта, созданный функцией ***CryptCreateHash***. Параметр *pbData* содержит указатель на хэшируемые данные, а *dwDataLen* содержит размер этих данных в байтах. Параметр *dwFlags* должен быть равен нулю.

Приведем пример вызова функции, осуществляющего хэширование некоторого пароля, хранящегося в строке *szPass* (описание см. выше).

```
.....................................................
//Получение каким-либо образом пароля и занесение его в szPass.
.....................................................
CryptHashData(hHash, (PBYTE)szPass,
              _tcslen(szPass)*sizeof(TCHAR), 0);
```

После того, как произведено хэширование парольной фразы, необходимо вызвать функцию ***CryptDeriveKey*** (описание см. ниже). После формирования ключа хэш-объект должен быть уничтожен функцией ***CryptDestroyHash***.

CryptDestroyHash

Функция уничтожает хэш-объект, ранее созданный функцией ***CryptCreateHash***. Ее прототип имеет вид:

```
BOOL WINAPI CryptDestroyHash(HCRYPTHASH hHash);
```

Единственный параметр представляет собой дескриптор уничтожаемого объекта. Пример вызова:

```
CryptDestroyHash(hHash);
```

CryptDeriveKey

Функция создает сеансовый ключ на базе хэшированного пароля. Прототип функции имеет вид:

```
BOOL WINAPI CryptDeriveKey(HCRYPTPROV hProv,
                           ALG_ID Algid,
                           HCRYPTHASH hBaseData,
                           DWORD dwFlags,
                           HCRYPTKEY *phKey);
```

Параметр *hProv* содержит дескриптор криптопровайдера, полученный с помощью функции ***CryptAcquireContext***. Параметр *Algid* содержит идентификатор того алгоритма шифрования, для которого создается ключ. При использовании криптопровайдера типа 24 целесообразно использовать следующие варианты: AES-128, AES-192, AES-256. Соответственно определены константы: *CALG_AES_128*, *CALG_AES_192*, *CALG_AES_256*.

Параметр *hBaseData* содержит дескриптор хэш-объекта, содержащего хэшированный пароль. Причем после создания сеансового ключа в хэш-объект запрещено добавлять новые данные Параметр *dwFlags*, содержащий значения флагов, можно установить равным нулю. Функция копирует дескриптор созданного ключа по адресу, переданному через параметр *phKey*. Пример вызова:

```
CryptDeriveKey(hProv, CALG_AES_128, hHash, 0, &hKeyH);
```

Данный вызов создаст сессионный ключ, представляющий собой объект в защищенной области памяти, доступ к которому осуществляется через дескриптор. При этом объект содержит в себе не только сам ключ, но и параметры шифрования: алгоритм, режим (по умолчанию CBC), вектор инициализации или синхропосылку (по умолчанию нулевой), размер блока и т.д. Непосредственно в открытом виде ключ приложению недоступен.

После того, как создан сеансовый ключ, можно осуществлять зашифрование или расшифрование с помощью функций ***CryptEncrypt*** или ***CryptDecrypt*** (описание см. ниже), передавая им в качестве одного из параметров полученный дескриптор. После шифрования ключ необходимо уничтожить функцией ***CryptDestroyKey***. В дальнейшем такой ключ может быть воссоздан, если предоставить для хэширования первоначальную парольную фразу.

CryptDestroyKey

Функция освобождает ранее определенный дескриптор ключа, созданного любым из двух способов или дескриптор ключевой пары. Ее прототип имеет вид:

```
BOOL WINAPI CryptDestroyKey(HCRYPTKEY hKey);
```

Если параметром функции является дескриптор сессионного ключа, то при вызове уничтожается сам ключ. Если дескриптор ключевой пары, то происходит только его освобождение, а сама пара остается в контейнере.

Пример вызова функции:

```
CryptDestroyKey(hKeyH);
```

CryptEncrypt

Функция осуществляет зашифрование данных. Прототип имеет вид:

```
BOOL WINAPI CryptEncrypt(HCRYPTKEY hKey,
                         HCRYPTHASH hHash,
                         BOOL Final,
                         DWORD dwFlags,
                         BYTE *pbData,
                         DWORD *pdwDataLen,
                         DWORD dwBufLen);
```

В параметре *hKey* передается дескриптор ключа, необходимый для шифрования. Этот ключ также определяет алгоритм шифрования. Параметр *hHash* используется, если исходные данные одновременно шифруются и хэшируются. В нашем случае этот параметр следует установить равным нулю. Параметр *Final* следует установить в *TRUE*, если переданный в функцию блок данных является единственным или последним. В этом случае, он будет дополнен до необходимого размера. В противном случае передается значение *FALSE*. Параметр *dwFlags* не используется, и на его месте следует указать ноль. Параметр *pbData* – указатель на буфер, в котором содержатся данные для зашифрования. Зашифрованные данные по завершении работы функции помещаются в тот же буфер. Через параметр *pdwDataLen* в функцию при вызове передается размер исходных данных, а по завершении возвращается размер зашифрованных. *dwBufLen* – размер выходного буфера, для блочных шифров может быть больше, чем *pdwDataLen*.

Если используется блочный шифр AES, то максимальное значение параметра *pdwDataLen* при вызове функции составляет (*dwBufLen* - 16). Это связано с тем, что шифрование по умолчанию ведется в режиме CBC. Размер блока в данной реализации алгоритма всегда равен 128 битам. При зашифровании последний блок (*Final* устанавливается в *TRUE*) всегда конкатенируется с дополнением, размер которого в данном случае может составлять от 1 до 16 байт. Реальный размер зашифрованного блока, используемый для его записи в выходной файл, возвращается также через параметр *pdwDataLen*. Если размер исходных данных при вызове превышает 128 бит, то функция самостоятельно делит их на блоки требуемого алгоритмом размера. При этом размер любого входного блока, кроме последнего, должен быть кратен 16 байтам.

Пример вызова функции для зашифрования не последнего блока:

```
dwBuflen = 496;
CryptEncrypt(hKeyH, 0, FALSE,0, aBuf, &dwBuflen, 512);
```

CryptDecrypt

Функция осуществляет расшифрование данных. Прототип имеет вид:

```
BOOL WINAPI CryptDecrypt(HCRYPTKEY hKey,
                         HCRYPTHASH hHash,
                         BOOL Final,
                         DWORD dwFlags,
                         BYTE *pbData,
                         DWORD *pdwDataLen);
```

Все параметры функции имеют тоже значение, что и для функции ***CryptEncrypt***. Шифртекст передается через параметр *pbData*, а его длина через *pdwDataLen*. Функция возвращает реальную длину расшифрованного блока также через параметр *pdwDataLen*. Размер выходных данных, в отличие от шифрующей функции, не указывается. Это связано с тем, что размер расшифрованных данных может только уменьшаться и отведенного при вызове размера буфера будет вполне достаточно.

Пример вызова функции для расшифрования не последнего блока. Переменная *dwBuflen* должна содержать размер передаваемых через буфер *aBuf* данных:

```
CryptDecrypt(hKeyH, 0, FALSE, 0, aBuf, &dwBuflen);
```

CryptReleaseContext

После завершения работы с функциями CryptoAPI, необходимо освободить контекст криптопровайдера, полученный ранее с помощью функции ***CryptAcquireContext***. Для этого вызывается функция ***CryptReleaseContext***. Ее прототип имеет вид:

```
BOOL WINAPI CryptReleaseContext(HCRYPTPROV hProv,
                                DWORD dwFlags);
```

Параметр *hProv* – дескриптор освобождаемого контекста криптопровайдера. Параметр *dwFlags* должен равняться нулю. Пример вызова функции:

```
CryptReleaseContext(hProv, 0);
```

Интерфейс Cryptography API: Next Generation

Криптографический интерфейс CryptoAPI, частично описанный выше, поддерживался и продолжает поддерживаться всеми версиями операционных систем семейства Windows. Однако, начиная с версии Windows Vista среди настольных ОС и Windows 2008 Server среди

серверных, появилась поддержка так называемого криптографического интерфейса следующего поколения: *Cryptography API: Next Generation* (сокращенно *CNG*).

Сравнивая интерфейс CNG с предшественником можно сказать, что по многим параметрам они схожи, но есть и различия. В CryptoAPI 1.0 центральным понятием является криптопровайдер (CSP). Существует относительно большое число типов криптопровайдеров, распространяемых для ОС Windows. Пользователю необходимо получить контекст одного из криптопровайдеров нужного типа и использовать возвращенный ему дескриптор для получения доступа к его функциям.

В CNG также имеется понятие провайдера. Однако порядок действий с ними несколько иной. Пользователь непосредственно может получать дескриптор нужного ему криптоалгоритма, не задумываясь к какому типу относится предоставляющий его провайдер. По сути, в CNG осталось всего четыре типа провайдеров. В стандартную поставку Windows входят обычно четыре CNG-провайдера, каждый из которых относится к соответствующему типу:

- «Microsoft Primitive Provider»;
- «Microsoft Smart Card Key Storage Provider»;
- «Microsoft Software Key Storage Provider»;
- «Microsoft SSL Protocol Provider».

Программы, которые используют средства первого и последнего из перечисленных провайдеров могут работать как в режиме ядра, так и пользовательском режиме. Использование остальных провайдеров возможно только в пользовательском режиме. Сведения об имеющихся провайдерах CNG можно получить в ветви реестра *HKLM\SYSTEM\CurrentControlSet\Control\Cryptography*. В частности, там содержится информация о том, какой из провайдеров CNG поддерживает выполнение того или иного криптоалгоритма, а также информация о месторасположении провайдеров.

Из названий вышеперечисленных провайдеров становится ясным их назначение. «Microsoft Primitive Provider» поддерживает выполнение функций так называемых криптографических примитивов: генерации ключей и случайных чисел, шифрования, хэширования, экспорта и импорта ключей. Провайдер «Microsoft Software Key Storage Provider» отвечает за хранение ключей асимметричного шифрования и цифровой подписи. Если такие ключи должны храниться на смарт-картах, то используется «Microsoft Smart Card Key Storage Provider». Ну и, наконец, провайдер «Microsoft SSL

Protocol Provider» применяется в приложениях, реализующих протоколы безопасной передачи данных SSL и TLS.

Далее в данной работе будут рассматриваться особенности применения криптографических примитивов для симметричного шифрования данных. Все содержимое «Microsoft Primitive Provider» для работы в пользовательском режиме находится в файле *bcryptprimitives.dll*. По сути это набор провайдеров отдельных алгоритмов. «Microsoft Primitive Provider» поддерживает работу всех тех же алгоритмов, которые реализованы в криптопровайдерах CryptoAPI от Microsoft. Кроме того, добавлена поддержка реализации алгоритмов Диффи-Хеллмана и DSA с использованием эллиптических кривых (ECDH, ECDSA), а также новых алгоритмов генерации псевдослучайных чисел.

Для использования криптографических примитивов пользователь напрямую работает не с указанной выше библиотекой, а с файлом так называемого *маршрутизатора примитивов* (primitive router). Этот файл содержит функции, которые получают доступ к одному из криптографических интерфейсов: асимметричного, симметричного шифрования, хэширования и других, реализованных в библиотеке *bcryptprimitives.dll*. В качестве маршрутизатора при работе с примитивами в пользовательском режиме используется библиотека *bcrypt.dll*.

Соответственно, все функции, реализованные в данном файле, имеют префикс *BCrypt* (буква «B» означает «базовый). Например, ***BCryptOpenAlgorithmProvider*** или ***BCryptEncrypt***. Для доступа к сервисам других провайдеров в основном используются функции, реализованные в файле *ncrypt.dll*, и имеющие в своем названии префикс *NCrypt* (буква «N» означает «новый»). Также в данном файле реализованы функции для работы с «Microsoft SSL Protocol Provider» и имеющие, соответственно, префикс *Ssl*.

Рассмотрим более подробно, как выполняется симметричное шифрование данных с использованием криптографических примитивов CNG. Для этого необходимо выполнить следующие действия:

1. Загрузить и инициализировать провайдер нужного алгоритма шифрования.
2. Если необходимо, установить требуемые параметры алгоритма.
3. Определить размер ключевого объекта и выделить для него память.
4. Сгенерировать (или импортировать) сеансовый ключ.
5. Если необходимо, установить требуемые параметры ключа.

6. Зашифровать/расшифровать данные.
7. Экспортировать (если необходимо) и уничтожить ключ.
8. Закрыть провайдер алгоритма шифрования.

Генерацию ключа, также как и в CryptoAPI, можно выполнять с помощью генератора случайных чисел или на основе хэшированной парольной фразы. В данной лабораторной работе не рассматриваются вопросы экспорта и импорта ключей. Поэтому для генерации сеансового ключа будет применяться второй способ, из чего следует, что перед выполнением пункта 1, дополнительно должны быть выполнены следующие действия:

1. Загрузить и инициализировать провайдер нужного алгоритма хэширования.
2. Определить размер будущего хэш-объекта и выделить для него память.
3. Создать хэш-объект.
4. Поместить в хэш-объект парольную фразу.
5. Извлечь вычисленный хэш-код, который в дальнейшем будет передан функции, генерирующей сеансовый ключ.
6. Уничтожить хэш-объект.
7. Закрыть провайдер алгоритма хэширования.

Далее будут рассмотрены функции, которые потребуются для реализации приведенных выше алгоритмов. Все прототипы функций будут приведены на языке C/C++, как они и указаны в документации. В проектах Win32 для получения доступа к функциям работы с примитивами необходимо загрузить библиотеку *bcrypt.dll*. Для этого можно использовать библиотеку импорта *bcrypt.lib*, указав этот файл в свойствах проекта (*Проект→ Свойства → Свойства конфигурации → Компоновщик → Ввод → Дополнительные зависимости*) или указав строку в тексте программы:

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

В файле исходного кода необходимо включить в текст директивой *#include* содержимое файлов *windows.h* и *bcrypt.h*. Причем включать файлы нужно именно в этом порядке, так как *bcrypt.h*, содержащий прототипы функций, константы, макросы, типы данных модуля BCrypt, использует в свою очередь некоторые типы данных, определенные в файлах, включение которых производится с помощью *windows.h*.

Все функции CNG, оперирующие строковыми параметрами, существуют только в одном варианте: с Unicode-строками. Поэтому строковые переменные нужно определять типом *WCHAR* (или *TCHAR*

и указывать в свойствах проекта, что используется Unicode), а перед константами ставить префикс “*L”* (или использовать макрос *TEXT*). Однако необходимость непосредственного оперирования ими возникает достаточно редко, так как для всех используемых функциями строковых констант в *bcrypt.h* определены символические константы.

Что касается специальных типов данных, определенных в *bcrypt.h*, то для описания приведенных ниже функций мы будем использовать переменные всего трех из них:

```
BCRYPT_ALG_HANDLE  hAlg;
BCRYPT_KEY_HANDLE hKey;
BCRYPT_HASH_HANDLE hHash;
```

Переменная *hAlg* будет использоваться в качестве дескриптора провайдера алгоритма, *hKey* в качестве дескриптора ключа, а *hHash* в качестве дескриптора хэш-объекта. Причем все три типа определены в файле *bcrypt.h* одинаково, как указатели на тип *VOID*. Что касается остальных типов параметров описываемых функций, то они являются стандартными типами Win32 API: *DWORD, PBYTE*.

Все функции CNG возвращают 32-битное значение типа *NTSTATUS*, которое представляет собой код ошибки ядра. Возможные значения определены в файле *ntstatus.h*. По структуре этот код схож со значением, возвращаемым функцией ***GetLastError***. Одним из вариантов определения успешности выполнения функции, является использование макроса *BCRYPT_SUCCESS*, определенного в файле *bcrypt.h*:

```
#define BCRYPT_SUCCESS(Status) (((NTSTATUS)(Status)) >= 0)
```

Если значение, возвращаемое макросом *BCRYPT_SUCCESS* истинно, то функция считается выполненной успешно. Проверка на неотрицательность в макросе производится потому, что только при ошибке старший разряд кода устанавливается в единицу. А поскольку тип *NTSTATUS* определен как тип *LONG* (знаковый), то значение с единичным старшим битом будет восприниматься как отрицательное.

Описание большинства из представленных ниже функций является неполным, но достаточным для выполнения данной лабораторной работы. Полное описание этих и других функций можно найти в разделе MSDN, посвященном использованию CNG (http://msdn.microsoft.com/en-us/library/windows/desktop/aa376210%28v=vs.85%29.aspx).

BCryptOpenAlgorithmProvider

Данная функция загружает и инициализирует провайдер конкретного алгоритма. Ее прототип имеет вид:

```
NTSTATUS WINAPI BCryptOpenAlgorithmProvider(
                BCRYPT_ALG_HANDLE *phAlgorithm,
                LPCWSTR pszAlgId,
                LPCWSTR pszImplementation,
                DWORD dwFlags);
```

Параметр *phAlgorithm* представляет собой указатель на переменную, принимающую возвращаемый дескриптор провайдера указанного алгоритма. Идентификатор запрашиваемого алгоритма задает второй параметр *pszAlgId*. Он представляет собой указатель на Unicode-строку с нуль-символом в конце, содержащую зарегистрированное имя одного из алгоритмов. Для стандартного CNG-провайдера «Microsoft Primitive Provider» в файле *bcrypt.h* определены строковые константы с именами зарегистрированных алгоритмов и соответствующие им символические константы, которые, как правило, и указываются в качестве второго параметра. При выполнении данной работы для шифрования будет использоваться алгоритм AES, а для хэширования – SHA-256. Поэтому из всех определенных идентификаторов нам понадобятся только два. Их описание в файле *bcrypt.h* приведено ниже:

```
#define  BCRYPT_AES_ALGORITHM          L"AES"
#define  BCRYPT_SHA256_ALGORITHM       L"SHA256"
```

Параметр *pszImplementation* является указателем на Unicode-строку, содержащую зарегистрированное имя (псевдоним) одного из провайдеров криптографических примитивов. Если в качестве этого параметра указать константу *NULL*, то будет использован провайдер, определенный в реестре для данного алгоритма по умолчанию («Microsoft Primitive Provider»).

Параметр *dwFlags* для реализации обычного шифрования или хэширования должен быть равен нулю.

Ниже показан пример вызова функции для подключения к провайдеру алгоритма AES и один из возможных вариантов обработки возвращаемого функцией значения:

```
NTSTATUS status = STATUS_UNSUCCESSFUL;
if(!BCRYPT_SUCCESS(status = BCryptOpenAlgorithmProvider(
                            &hAlg,
                            BCRYPT_AES_ALGORITHM,
                            NULL, 0)))
```

```
{
     _tprintf(TEXT("Ошибка 0x%x получения\
              дескриптора алгоритма"), status);
}
```

Если необходимо подключиться к провайдеру алгоритма хэширования SHA-256, то используется идентификатор *BCRYPT_SHA256_ALGORITHM*.

Полученный через параметр *phAlgorithm* дескриптор алгоритма, в зависимости от его типа, в дальнейшем используется для создания либо ключевого объекта, либо хэш-объекта.

BCryptGetProperty

Функция позволяет получить значение одного из свойств объекта CNG (алгоритма, ключевого объекта, хэш-объекта) и имеет следующий прототип:

```
NTSTATUS WINAPI BCryptGetProperty(
                BCRYPT_HANDLE   hObject,
                LPCWSTR pszProperty,
                PUCHAR   pbOutput,
                ULONG   cbOutput,
                ULONG   *pcbResult,
                ULONG   dwFlags);
```

В качестве параметра *hObject* в функцию может передаваться дескриптор любого из объектов CNG, указанных выше. Параметр *pszProperty* является указателем на Unicode-строку, содержащую имя запрашиваемого свойства. Значения свойств могут быть разных типов, но все они возвращаются через параметр *pbOutput*. По сути функции передается адрес буфера в памяти, куда передается значение свойства. Поэтому передаваемый функции параметр должен приводиться к типу *PUCHAR* или *PBYTE*. Размер буфера задается параметром *cbOutput*, а параметр *pcbResult* задает адрес переменной, куда возвращается размер данных в байтах, реально скопированных в буфер *pbOutput*. Параметр *dwFlags* должен быть равен нулю.

Данная функция совместно с функцией ***BCryptSetProperty*** в качестве идентификаторов свойств может использовать символические константы, определенные в файле *bcrypt.h*. Некоторые из них приведены ниже:

– *BCRYPT_ALGORITHM_NAME*. Позволяет определить имя используемого алгоритма;

- *BCRYPT_BLOCK_LENGTH*. Позволяет получить размер блока алгоритма шифрования в байтах;
- *BCRYPT_CHAINING_MODE*. Позволяет получить название установленного режима шифрования. Для основных режимов блочного шифрования определены символические константы: *BCRYPT_CHAIN_MODE_CBC*, *BCRYPT_CHAIN_MODE_CFB*, *BCRYPT_CHAIN_MODE_ECB*, которые соответствуют строковым константам, содержащим названия режимов;
- *BCRYPT_KEY_LENGTH*. Позволяет получить размер ключа в битах;
- *BCRYPT_HASH_LENGTH*. Позволяет получить размер хэш-кода загруженного алгоритма хэширования;
- *BCRYPT_OBJECT_LENGTH*. Используется для получения размера в байтах объектов провайдера некоторых типов.

Что касается последнего идентификатора, то он используется при вызове функции, в частности, если нужно определить размер ключевого объекта перед созданием сеансового ключа или размер хэш-объекта. В приведенном ниже примере определяется размер объекта, тип которого зависит от типа загруженного алгоритма. Если переменная *hAlg* содержит дескриптор алгоритма шифрования, то через параметр *pbOutput* возвратится размер ключевого объекта, а если загружен алгоритм хэширования, то вернется размер хэш-объекта. В приведенном ниже примере и во всех последующих для краткости не записывается «обвязка» вызова, проверяющая возвращаемое функцией значение.

```
DWORD cbObject=0, cbData=0;
PBYTE pbObject=NULL;
BCryptGetProperty(hAlg, BCRYPT_OBJECT_LENGTH,
                  (PBYTE)&cbObject, sizeof(DWORD),
                  &cbData, 0);
//выделение памяти для объекта:
pbObject=(PBYTE)HeapAlloc(GetProcessHeap(), 0,
                          cbObject);
//выполнение каких-либо операций с объектом
.....................................
//освобождение  объекта
HeapFree(GetProcessHeap(), 0, pbObject);
```

Далее созданный в памяти буфер может быть передан функциям, которые разместят в нем, соответственно, или ключевой объект, или хэш-объект. В переменную *cbData* в этом случае будет записан размер в байтах переданных в *cbObject* данных, то есть 4.

BCryptSetProperty

Функция позволяет установить значение одного из именованных свойств объекта CNG (алгоритма, ключевого объекта, хэш-объекта). Ее прототип во многом повторяет прототип предыдущей функции:

```
NTSTATUS WINAPI BCryptSetProperty(
                BCRYPT_HANDLE   hObject,
                LPCWSTR pszProperty,
                PUCHAR   pbInput,
                ULONG   cbInput,
                ULONG   dwFlags);
```

Разница между этой функцией и предыдущей заключается в том, что через параметр *pbInput* передается новое значение свойства, заданного параметром *pszProperty*. Через параметр *cbInput* передается размер этого значения. Параметр dwFlags также должен иметь значение нуля.

Формально функция ***BCryptSetProperty*** может использовать те же идентификаторы, что и функция ***BCryptGetProperty***. Однако из всех перечисленных в описании предыдущей функции идентификаторов, функция ***BCryptSetProperty*** может использовать только один: *BCRYPT_CHAINING_MODE*. Другие рассмотренные идентификаторы определяют свойства, доступные только для чтения. Ниже показан пример вызова функции, устанавливающий для полученного ранее дескриптора алгоритма симметричного шифрования режим CFB:

```
BCryptSetProperty(hAlg, BCRYPT_CHAINING_MODE,
                  (PBYTE)BCRYPT_CHAIN_MODE_CFB,
                  sizeof(BCRYPT_CHAIN_MODE_CFB), 0);
```

BCryptCreateHash

Функция создает хэш-объект и имеет следующий прототип:

```
NTSTATUS WINAPI BCryptCreateHash(
                BCRYPT_ALG_HANDLE hAlgorithm,
                BCRYPT_HASH_HANDLE *phHash,
                PUCHAR pbHashObject,
                ULONG cbHashObject,
                PUCHAR pbSecret,
                ULONG cbSecret,
                ULONG dwFlags);
```

Параметр *hAlgorithm* представляет полученный ранее с помощью функции ***BCryptOpenAlgorithmProvider*** дескриптор алгоритма хэширования. Параметр *phHash* является указателем на переменную типа *BCRYPT_HASH_HANDLE*, которая будет содержать дескриптор

создаваемого хэш-объекта. В дальнейшем этот дескриптор будет передаваться функциям, которые непосредственно выполняют хэширование данных и выгрузку хэш-кода.

Параметр *pbHashObject* содержит указатель на буфер, в котором размещается созданный хэш-объект. Способ получения этого буфера был описан ранее. Параметр *cbHashObject* содержит размер этого буфера. Параметр *pbSecret* передает адрес буфера, содержащего ключ, используемый для создания кода аутентичности сообщения (MAC). Поскольку мы будем выполнять обычное хэширование, то вместо этого параметра нужно указывать константу *NULL*, а в качестве значения параметра *cbSecret* – 0. Параметр *dwFlags* в нашем случае тоже должен быть равен нулю.

Ниже показан пример вызова этой функции. Предполагается, что буфер, адресуемый указателем *pbHashObject* и его размер *cbHashObject* получены заранее:

```
BCryptCreateHash(hAlg, &hHash, pbHashObject,
                 cbHashObject, NULL, 0, 0);
```

BCryptHashData

Функция непосредственно выполняет хэширование передаваемых ей данных, используя созданный ранее с помощью функции ***BCryptCreateHash*** хэш-объект. Прототип функции имеет следующий вид:

```
NTSTATUS WINAPI BCryptHashData(
                BCRYPT_HASH_HANDLE hHash,
                PUCHAR pbInput,
                ULONG cbInput,
                ULONG dwFlags);
```

Параметр *hHash* представляет собой дескриптор существующего хэш-объекта. Параметр *pbInput* содержит адрес буфера с входными данными, а параметр *cbInput* его размер. Параметр *dwFlags* должен быть равен нулю. Функция может вызываться многократно для одного и того же хэш-объекта, добавляя в него новые данные для хэширования.

Ниже показан пример хэширования пароля, который находится в строке *szPass*.

```
TCHAR szPass[100];
.............................
//Получение каким-либо образом пароля и занесение его в szPass.
.............................
BCryptHashData(hHash,(PBYTE)szPass,
               _tcslen(szPass)*sizeof(TCHAR),0);
```

BCryptFinishHash

Функция извлекает из хэш-объекта окончательный хэш-код входных данных, полученный с помощью одного или нескольких вызовов функции ***BCryptHashData.*** Прототип функции имеет вид:

```
NTSTATUS WINAPI BCryptFinishHash(
                BCRYPT_HASH_HANDLE hHash,
                PUCHAR pbOutput,
                ULONG cbOutput,
                ULONG dwFlags);
```

Параметр *hHash* является дескриптором хэш-объекта. Буфер, заданный указателем *pbOutput*, будет принимать выгружаемый хэш-код и должен быть создан заранее. Параметр *cbOutput* содержит размер этого буфера. Параметр *dwFlags* должен быть равен нулю.

Ниже показан пример выгрузки хэш-кода:

```
PBYTE pbHash= NULL;
DWORD cbData= 0, cbHash= 0;
BCryptGetProperty(hAlg, BCRYPT_HASH_LENGTH,
                  (PBYTE)&cbHash, sizeof(DWORD),
                  &cbData, 0);
pbHash= (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHash);
BCryptFinishHash(hHash, pbHash, cbHash, 0);
```

BCryptGenerateSymmetricKey

Функция создает ключевой объект и помещает в него ключ симметричного шифрования загруженного ранее алгоритма. Ключ создается на основе ключевого материала, передаваемого функции. Ее прототип имеет вид:

```
NTSTATUS WINAPI BCryptGenerateSymmetricKey(
                BCRYPT_ALG_HANDLE hAlgorithm,
                BCRYPT_KEY_HANDLE *phKey,
                PUCHAR pbKeyObject,
                ULONG cbKeyObject,
                PUCHAR pbSecret,
                ULONG cbSecret,
                ULONG dwFlags);
```

Параметр *hAlgorithm* представляет полученный ранее с помощью функции ***BCryptOpenAlgorithmProvider*** дескриптор алгоритма шифрования. Параметр *phKey* является указателем на переменную типа *BCRYPT_KEY_HANDLE*, которая будет содержать дескриптор создаваемого ключевого объекта. Параметр *pbKeyObject* содержит указатель на буфер, в котором размещается созданный ключевой

объект. Способ получения этого буфера был описан ранее. Параметр *cbKeyObject* содержит размер данного буфера в байтах.

Параметр *pbSecret* является указателем на буфер, который содержит ключевой материал. По сути, он и будет являться ключом. Получить ключевой материал можно двумя способами: генерацией псевдослучайного числа требуемого размера или созданием хэш-кода парольной фразы. Поэтому в нашем случае в качестве этого параметра функции будет передаваться указатель на буфер с хэш-кодом, извлеченным ранее функцией ***BCryptFinishHash***. Через параметр *cbSecret* передается размер в байтах буфера с ключевым материалом.

Если вспомнить генерацию сеансового ключа AES в CryptoAPI, то там использовались отдельные идентификаторы алгоритма для разных длин ключа. Здесь же длина генерируемого ключа фактически определяется значением, передаваемым через параметр *cbSecret*. Если, к примеру, через параметр *pbSecret* передается адрес буфера, содержащего 32-байтный (256-битный) хэш-код, то задавая через параметр *cbSecret* значения 16, 24 или 32 мы получим ключ длиной, соответственно, 128, 192 или 256 бит.

Параметр *dwFlags* должен быть равен нулю. Ниже показан пример вызова функции для создания 128-битного ключа. Предполагается, что на момент вызова функции уже определены значения параметров *pbKeyObject* (содержит адрес буфера для размещения ключевого объекта), *cbKeyObject* (содержит размер буфера в байтах) и *pbHash* (содержит адрес буфера с хэш-кодом).

```
BCryptGenerateSymmetricKey(hAlg, &hKey, pbKeyObject,
                           cbKeyObject, (PBYTE)pbHash,
                           16, 0);
```

Для созданного после вызова функции дескриптора ключа можно определять и устанавливать некоторые параметры с помощью функций ***BCryptGetProperty*** и ***BCryptSetProperty***. Например, если установлен режим шифрования CFB, то по умолчанию он используется в 8-битном варианте. Для того чтобы установить иной размер обратной связи (в байтах) можно использовать функцию ***BCryptSetProperty*** с идентификатором *BCRYPT_MESSAGE_BLOCK_LENGTH*.

После создания ключа и установления его параметров (если это необходимо) можно передавать его дескриптор функциям зашифрования или расшифрования.

BCryptEncrypt

Функция зашифровывает переданный ей блок данных произвольного размера. Ее прототип имеет вид:

```
NTSTATUS WINAPI BCryptEncrypt(
                BCRYPT_KEY_HANDLE hKey,
                PUCHAR pbInput,
                ULONG cbInput,
                VOID *pPaddingInfo,
                PUCHAR pbIV,
                ULONG cbIV,
                PUCHAR pbOutput,
                ULONG cbOutput,
                ULONG *pcbResult,
                ULONG dwFlags);
```

Через параметр *hKey* передается дескриптор созданного ранее ключа. Параметр *pbInput* содержит адрес буфера с входными данными (открытым текстом). Параметр *cbInput* содержит размер входных данных в байтах. Параметр *pPaddingInfo* может содержать указатель на одну из двух возможных структур, определяющих параметры дополнения блоков при шифровании. В случае обычного симметричного шифрования этот параметр не используется и должен быть равен *NULL*.

Через параметр *pbIV* можно передавать адрес буфера, содержащего вектор инициализации (синхропосылку). Обычно он получается с помощью генератора псевдослучайных чисел. Размер вектора в байтах передается через параметр *cbIV* и он должен совпадать с размером блока. То есть если размер блока алгоритма AES равен 16 байтам (128 бит), то вектор инициализации должен иметь такой же размер. Также нужно учесть, что в процессе зашифрования функция изменяет содержимое буфера с вектором. Поэтому если в дальнейшем вновь понадобится использовать его значение, то перед вызовом функции необходимо создать копию вектора. Если при зашифровании используется нулевой вектор инициализации, то параметры *pbIV* и *cbIV* должны быть заданы, соответственно, как *NULL* и 0.

Параметр *pbOutput* содержит адрес буфера, в который будет помещаться шифртекст. Размер буфера в байтах передается через параметр *cbOutput*. Если в качестве параметра *pbOutput* передать *NULL*, то через параметр *pcbResult* будет возвращен размер выходного буфера, необходимого для размещения шифртекста, который получится из указанных входных данных. Если же параметр *pbOutput*

при вызове содержит адрес выходного буфера, то через *pcbResult* возвращается количество байт, реально помещенных в этот буфер.

Параметр *dwFlags* в случае симметричного шифрования может быть равен нулю или константе *BCRYPT_BLOCK_PADDING*, которая определяет необходимость дополнения последнего блока. Флаг устанавливается в ноль, если четко известно, что размер входных данных кратен размеру блока алгоритма симметричного шифрования. Если же при нулевом флаге размер открытого текста не кратен размеру блока, то восстановить его в дальнейшем может не получиться, вне зависимости от выбранного режима шифрования. Поэтому при шифровании файлов флаг всегда нужно устанавливать равным *BCRYPT_BLOCK_PADDING*, независимо от выбранного режима шифрования (в том числе и при выборе режима CFB). Следовательно, размер выходных данных будет всегда больше (для AES на величину от 1 до 16 байт), чем входных.

Отсюда следует еще один вывод. Параметры *pbInput* и *pbOutput* могут содержать адрес одного и того же буфера. Однако этот буфер нельзя полностью заполнять входными данными, так как заменяющие их по мере зашифрования выходные данные будут иметь больший размер (для AES максимально на 16 байт).

Ниже показан пример зашифрования массива однобайтных величин с нулевым вектором инициализации и предварительным определением размера буфера для размещения шифртекста:

```
BYTE Plaintext[]={1, 2, 3, 4, 5, 6, 7, 8, 9, 10};
DWORD cbOutput=0, cbCiphertext=0;
PBYTE pbOutput=NULL;
BCryptEncrypt(hKey, Plaintext, sizeof(Plaintext),
              NULL, NULL, 0, NULL, 0, &cbOutput,
              BCRYPT_BLOCK_PADDING);
pbOutput=(PBYTE)HeapAlloc(GetProcessHeap(), 0,
                          cbOutput);
BCryptEncrypt(hKey, Plaintext, sizeof(Plaintext),
              NULL, NULL, 0, pbOutput, cbOutput,
              &cbCiphertext, BCRYPT_BLOCK_PADDING);
```

BCryptDecrypt

Функция расшифровывает переданный ей блок данных произвольного размера. Ее прототип имеет вид:

```
NTSTATUS WINAPI BCryptDecrypt(
                BCRYPT_KEY_HANDLE hKey,
                PUCHAR pbInput,
                ULONG cbInput,
                VOID *pPaddingInfo,
```

```
                PUCHAR pbIV,
                ULONG cbIV,
                PUCHAR pbOutput,
                ULONG cbOutput,
                ULONG *pcbResult,
                ULONG dwFlags);
```

Все параметры этой функции по назначению аналогичны параметрам функции ***BCryptEncrypt***. Отличие состоит лишь в том, что через параметр *pbInput* передается адрес буфера с шифртекстом, а параметр *pbOutput* указывает на буфер, в котором будет размещен восстановленный открытый текст. Рассмотренный ранее способ определения размера выходного буфера может применяться и для этой функции. Параметр *dwFlags* также должен иметь значение *BCRYPT_BLOCK_PADDING*.

BCryptDestroyHash

Функция уничтожает созданный ранее хэш-объект и имеет следующий прототип:

```
NTSTATUS WINAPI BCryptDestroyHash(
                BCRYPT_HASH_HANDLE hHash);
```

Единственный параметр представляет собой дескриптор уничтожаемого хэш-объекта.

BCryptDestroyKey

Функция уничтожает созданный ранее ключевой объект и имеет следующий прототип:

```
NTSTATUS WINAPI BCryptDestroyKey(
                BCRYPT_KEY_HANDLE hKey);
```

В качестве параметра функция принимает дескриптор уничтожаемого ключа.

BCryptCloseAlgorithmProvider

Функция выгружает загруженный ранее провайдер алгоритма и имеет следующий прототип:

```
NTSTATUS WINAPI BCryptCloseAlgorithmProvider(
                BCRYPT_ALG_HANDLE hAlgorithm,
                ULONG dwFlags);
```

Параметрами функции являются дескриптор провайдера и флаг, который должен быть равен нулю.

СОДЕРЖАНИЕ РАБОТЫ

Разработать на языке программирования C/C++ консольное или оконное приложение, выполняющее зашифрование и расшифрование файла произвольного формата с помощью алгоритма AES-128, AES-192 или AES-256 по выбору пользователя. Программа должна генерировать сеансовый ключ на основании хэшированного пароля, который запрашивается у пользователя. Использовать хэш-функцию SHA-256.

Приложение должно по выбору пользователя использовать функционал двух криптографических интерфейсов:

- CryptoAPI (в этом случае режим блочного шифрования – устанавливаемый по умолчанию CBC с нулевым вектором инициализации, используемый криптопровайдер – «Microsoft Enhanced RSA and AES Cryptographic Provider»).
- Cryptography API: Next Generation (дополнительно предоставляется выбор режим блочного шифрования: ECB, CBC или CFB, причем в последних двух также используется нулевой вектор инициализации).

КОНТРОЛЬНЫЕ ВОПРОСЫ

1. Что такое CryptoAPI? В чем заключается различие между CryptoAPI 1.0 и CryptoAPI 2.0?
2. Что такое криптопровайдер? Как можно подключиться к криптопровайдеру?
3. Какое количество функций должен поддерживать криптопровайдер?
4. Как создать контейнер ключей? Какие типы ключей в нем будут храниться?
5. Какие типы криптопровайдеров вы знаете? Чем они различаются?
6. Как можно выполнить генерацию ключа симметричного шифрования?
7. Какой режим шифрования устанавливается при генерации ключа по умолчанию?
8. Что такое хэш-объект? Какие функции для работы с хэш-объектами вы знаете?
9. Какие функции CryptoAPI выполняют зашифрование и расшифрование данных? Какие они имеют параметры?

10. Что такое Cryptography API: Next Generation? В чем заключаются его различия с CryptoAPI?
11. Какие типы провайдеров CNG доступны в операционных системах Windows? Как можно узнать, какие конкретно провайдеры установлены в системе?
12. Средства каких провайдеров CNG можно использовать в режиме ядра?
13. Какие алгоритмы поддерживает провайдер криптографических примитивов Microsoft?
14. Что такое маршрутизатор примитивов?
15. Как определить успешность вызова функции CNG?
16. Как загрузить и выгрузить провайдер нужного алгоритма?
17. Как создается хэш-объект? Какие функции CNG для работы с хэш-объектами вы знаете?
18. Как определить размер буфера при выгрузке хэш-кода?
19. Как сгенерировать ключ симметричного шифрования и установить его параметры?
20. Какие функции CNG выполняют зашифрование и расшифрование данных? Какие они имеют параметры?