УДК 511.3

© 2017 г. **К. М.** **Стуков**

(Тихоокеанский государственный университет, Хабаровск)

**РАЗРАБОТКА ПРИЛОЖЕНИЯ С ИСПОЛЬЗОВАНИЕМ БАЗОВЫХ ФУНКЦИЙ**

**ИНТЕРФЕЙСА CRYPTOAPI**

В статье рассматривается приложение, выполняющее функции шифрования и дешифрования данных, создания цифровой подписи и ее верификации. Приведено подробное описание использованных функций библиотеки CryptoAPI.

***Ключевые слова:*** CryptoAPI, криптопровайдер, шифрование, дешифрование, цифровая подпись, защита.

**K. M. Stukov**

**DEVELOPMENT OF AN APPLICATION USING THE BASIC FUNCTIONS OF THE CRYPTOAPI INTERFACE**

The article describes the application that performs the functions of data encryption and decryption, creation of a digital signature and its verification. The detailed description of the functions of the CryptoAPI library is given.

***Keywords:*** CryptoAPI, crypto provider, encryption, decryption, digital signature, protection.

**Введение**

Криптография достаточно актуальная вещь в современном мире. Она позволяет передавать информацию в защищенной форме, обеспечивая безопасность, конфиденциальность и целостность данных. При защите конфиденциальной информации криптография способствует высокому уровню безопасности персональных данных людей.

На сегодняшний день одной из самых популярных операционных систем является Microsoft Windows. И Microsoft предоставляет встроенные средства шифрования данных, начиная с Windows 95 OSR2. Этот блок API (Application Programming Interface) носит название Microsoft CryptoAPI.

CryptoAPI – это интерфейс прикладного программирования, предоставляющий службы шифрования для операционной системы и приложений, работающих под ее управлением. Он содержит ряд функций, позволяющих приложениям шифровать данные и ставить цифровую подпись различными способами, обеспечивая защиту личных ключей. Однако сами функции CryptoAPI не выполняют никаких криптографических действий, а служат лишь посредниками между прикладной программой и CSP (Cryptography Service Provider – поставщик службы шифрования) – криптопровайдером [1].

Криптопровайдер – это независимый модуль, содержащий библиотеку криптографических функций со стандартизованным интерфейсом. Криптопровайдер отвечает за реализацию функций интерфейса, а также играет роль хранилища для ключей всех типов. Подобная архитектура позволяет переходить от одного провайдера к другому с минимальными изменениями исходного кода, так как интерфейс не меняется.

Программная часть криптопровайдера представляет собой dll-файл с функциями поддержки шифрования. Криптопровайдеры отличаются друг от друга составом функций, требованиями к оборудованию, алгоритмами, осуществляющими базовые действия (создание ключей, хеширование и пр.) [2].

Нашей задачей являлась разработка приложения с использованием базовых функций интерфейса CryptoAPI, а именно шифрование данных и создание цифровой подписи.

**Описание приложения**

Приложение реализовано на языке С++ в среде разработки Qt с использованием библиотеки CryptoAPI. Приложение выполняет функции шифрования, дешифрования данных, создания цифровой подписи и ее верификации.

Для шифрования и дешифрования пользователю необходимо выбрать файл, данные которого следует обработать, а также файл, в который будут записаны модифицированные данные. Кроме этого, при шифровании данных можно задать пароль, на основе которого будет генерироваться сессионный ключ (рис 1).

Если для шифрования данных используется пароль, этот же пароль необходимо использовать и для дешифрования. В случае шифрования без использования пароля, сессионный ключ генерируется случайным образом и записывается в конечный файл. При дешифровании ключ считывается из файла, и уже с его помощью сообщение расшифровывается.

В CryptoAPI для шифрования и дешифрования используются и симметричный, и асимметричный алгоритмы. Симметричный алгоритм менее надежен, но работает намного быстрее, чем асимметричный. Поэтому в CryptoAPI применяется комбинация алгоритмов. Данные шифруются с помощью симметричного алгоритма с сессионным ключом, а сам сессионный ключ шифруется по асимметричному алгоритму открытым ключом получателя. Дешифрование происходит в обратном порядке: сначала закрытым ключом получателя дешифруется сессионный ключ, а затем этим сессионным ключом дешифруются сами данные.

В разработанном приложении процесс шифрования представлен следующим образом. Сначала открываются файлы для чтения и записи с помощью функции *CreateFile*. Далее необходимо получить доступ к криптопровайдеру, для этого используется функция *CryptAcquireContext*. В нее передается имя провайдера и его тип. В данной программе используется *Microsoft Enhanced Cryptographic Provider (MS\_ENHANCED\_PROV)*, тип – *PROV\_RSA\_FULL*. Провайдеры такого типа поддерживают как цифровые подписи, так и шифрование данных. Как видно из названия, для обмена ключами и создания цифровой подписи используется алгоритм RSA.

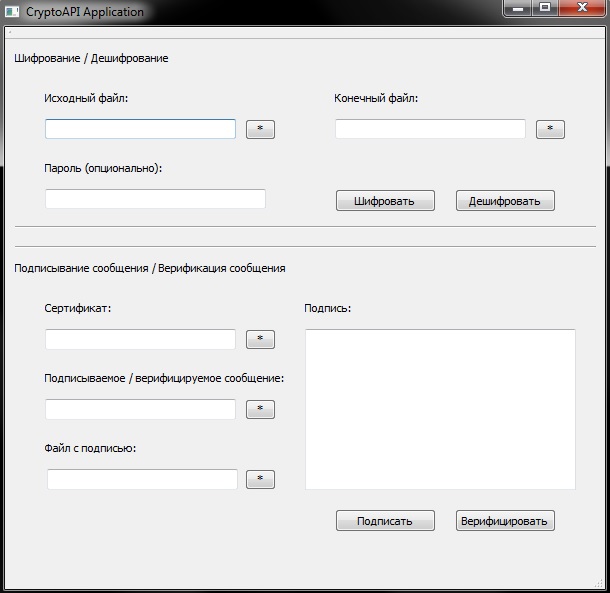


Рис. 1. Окно разработанного приложения.

После получения доступа к криптопровайдеру происходит генерирование сессионного ключа. Если для создания ключа пользователем был введен пароль, то сначала этот пароль хешируется, а затем уже на основе полученного хеша создается сессионный ключ. Для хеширования пароля используются две функции: *CryptCreateHash* и *CryptHashData*. Первая функция создает хеш-объект. Она принимает дескриптор криптопровайдера и идентификатор алгоритма хеширования. Провайдеры *PROV\_RSA\_FULL* поддерживают два алгоритма хеширования – MD5 и SHA. В программе используется MD5. Вторая функция заполняет хеш-объект паролем, приведенным к двоичному виду. Генерирование сессионного ключа осуществляется функцией *CryptDeriveKey*. Одним из параметров данной функции является идентификатор алгоритма симметричного шифрования. Криптопровайдер предоставляет на выбор два алгоритма: RC2 и RC4, в данном случае используется последний.

Если пользователь не вводил пароль, то ключ генерируется случайным образом функцией *CryptGenKey*. Она очень схожа с *CryptDeriveKey*. В функцию передается флаг *CRYPT\_EXPORTABLE*, он предоставляет возможность экспорта полученного ключа из криптопровайдера. После этого забирается дескриптор ключа пользователя с помощью *CryptGetUserKey*. Далее *CryptExportKey* экспортирует ключ из криптопровайдера в двоичный массив данных BLOB. Функция вызывается два раза – в первый раз для определения размера BLOB-массива, во второй – для экспорта ключа в данный массив. После этого массив записывается в файл функцией *WriteFile*.

Когда ключ создан и, если была необходимость, записан в файл, осуществляется шифрование с использованием этого ключа. Операция выглядит следующим образом. Из файла считывается блок данных фиксированного размера с помощью функции *ReadFile*, этот блок шифруется с помощью *CryptEncrypt* и затем записывается в конечный файл. Операция повторяется, пока не будет достигнут конец файла. В *CryptEncrypt* передаются дескриптор ключа, блок с данными, метка, определяющая, последний блок или нет, а также размеры буферов для хранения данных. Зашифрованный блок данных не возвращается отдельным параметром, он перезаписывает блок исходных данных. После окончания операции все дескрипторы уничтожаются с помощью *CryptDestroyXXX*, файлы закрываются функциями *CloseHandle* и закрывается доступ к криптопровайдеру *CryptReleaseContext* [3]. На рис. 2 представлен алгоритм шифрования данных.

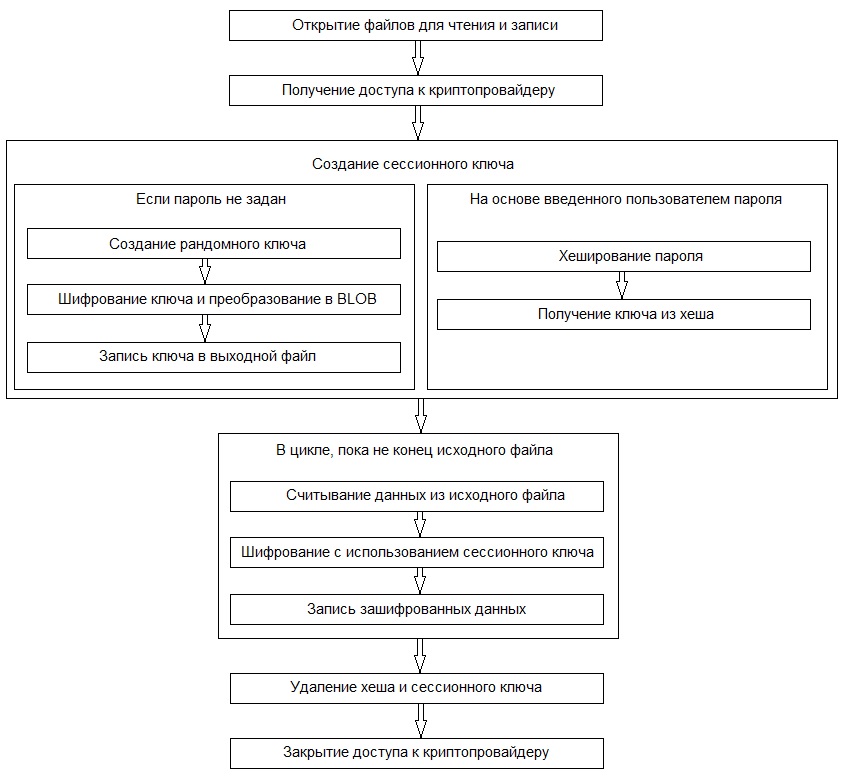


Рис. 2. Алгоритм шифрования данных.

Процесс дешифрования начинается с открытия файлов для чтения и записи, получения доступа к провайдеру. Далее проверяется, ввел пользователь пароль или нет. Если да, то пароль так же, как и при шифровании, хешируется, и генерируется сессионный ключ. Если нет, ключ считывается из файла.

Функция *ReadFile* вызывается дважды – для определения размера буфера, в который будет записан ключ, и для считывания в данный буфер. После этого вызывается *CryptImportKey*. Функция выполняет операцию по импорту ключа из BLOB в криптопровайдер. Далее выполняется уже само дешифрование. Операция аналогична шифрованию, только вместо функции *CryptEncrypt* используется *CryptDecrypt*. В конце происходит закрытие доступов и уничтожение дескрипторов. На рис. 3 представлен алгоритм дешифрования данных.

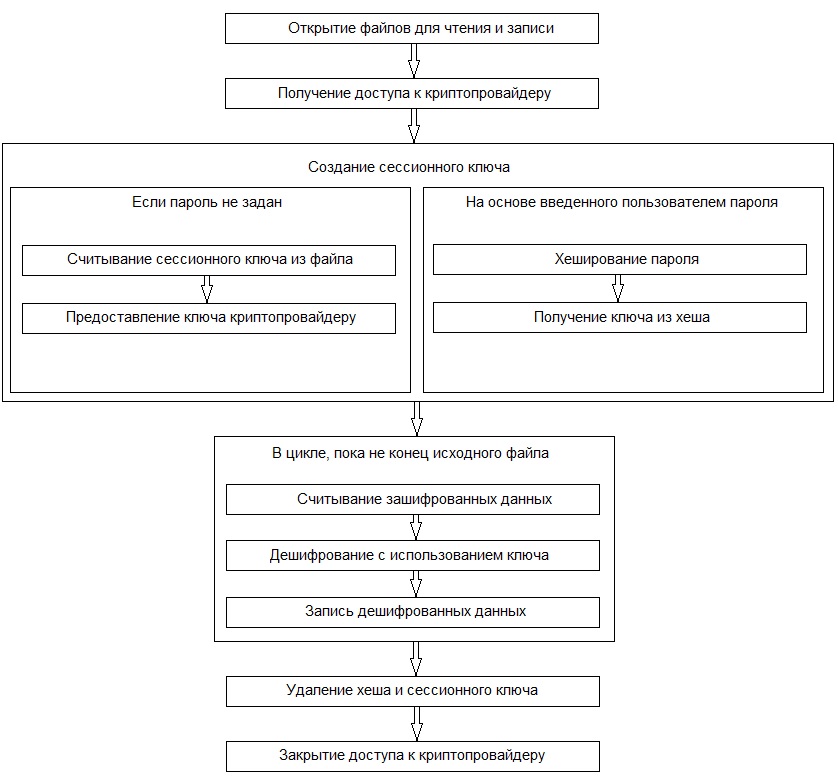


Рис. 3. Алгоритм дешифрования данных.

Для создания цифровой подписи пользователю необходимо выбрать подписываемый файл и сертификат, с помощью которого будет создана цифровая подпись. После нажатия на кнопку «Подписать» подпись генерируется и записывается в файл с расширением .detsf. Его можно найти в той же папке, что и подписываемый файл. Кроме этого, в пустом поле появляется подпись в шестнадцатеричном формате (рис. 1).

Формирование подписи начинается с открытия подписываемого файла для чтения и создания файла для хранения цифровой подписи с помощью *CreateFile*. После открытия исходного файла определяется размер его содержимого функцией *GetFileSize*, и выделяется память под считываемые данные. Копирование данных из файла в буфер осуществляется с помощью описанной ранее функции *ReadFile*. Далее открывается хранилище сертификатов. Для этого используется функция *CertOpenStore*. В данной программе открывается доступ к хранилищу с личными сертификатами пользователя. После открытия хранилища осуществляется поиск сертификата по его имени. Если указанный пользователем сертификат отсутствует, будет выдано сообщение об ошибке.

Поиск сертификата выполняет функция *CertFindCertificateInStore*. Кроме имени сертификата ей передаются идентификаторы возможных форматов подписей – PKCS #7 и X509. После нахождения сертификата формируется структура цифровой подписи *CRYPT\_SIGN\_MESSAGE\_PARA*. У нее довольно много параметров. Некоторые из них: формат цифровой подписи, имя сертификата, размер структуры в байтах, алгоритм хеширования данных, количество сертификатов, включаемых в подписываемое сообщение. Сформированная структура передается в *CryptSignMessage* – функцию, которая создает хеш из считанных из файла данных, подписывает его и кодирует. Данная функция вызывается два раза – для определения размера хеша и для его подписи. Стоит отметить, что функцию следует вызывать со вторым параметром, установленным в TRUE, это позволяет создать обособленную подпись – подпись, не включающую в себя само подписываемое сообщение.

Для верификации пользователь должен указать сертификат, верифицируемый файл и файл с подписью, после чего нажать на кнопку «Верифицировать». Операция верификации очень схожа с операцией по созданию цифровой подписи. Цифровая подпись и верифицируемые данные считываются из файлов, открывается доступ к хранилищу сертификатов и осуществляется проверка на наличие указанного сертификата. Если сертификат найден, то формируется структура *CRYPT\_VERIFY\_MESSAGE\_PARA*. Эта структура передается в функцию *CryptVerifyDetachedMessageSignature*, которая используется для верификации данных с помощью обособленной подписи [4]. Функция принимает указатель на структуру для верификации, индекс цифровой подписи, указатель на буфер с подписью, указатель на буфер с верифицируемыми данными и указатель на сертификат. Индекс цифровой подписи в данной программе равен нулю, так как предполагается, что для подписи использовался только один сертификат. На рис. 4 представлен алгоритм создания цифровой подписи.

.

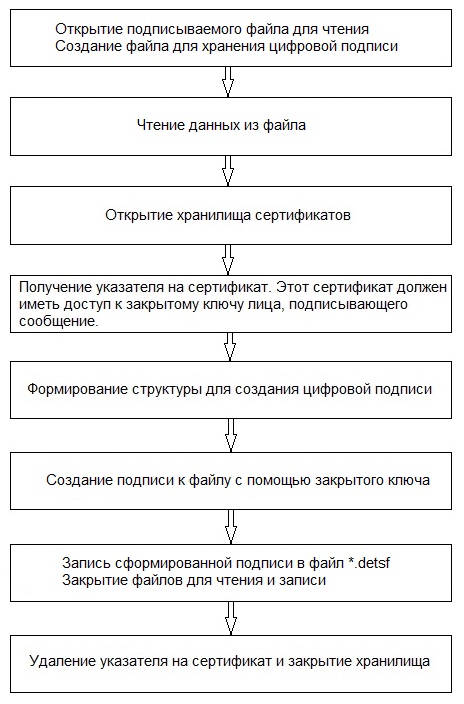


Рис. 4. Алгоритм создания цифровой подписи.

**Заключение**

В ходе работы был рассмотрен интерфейс CryptoAPI с рядом его функций, позволяющих приложениям шифровать данные и ставить цифровую подпись. Результатом работы является разработанное приложение на основе CryptoAPI.

**Список литературы**

1. Лясин Д. Н., Саньков С. Г. Методы и средства защиты компьютерной информации – Волгоград: ВолгГТУ, 2005.
2. Национальная библиотека им. Н. Э. Баумана, Windows CryptoAPI.
3. Камышников И. Г. Обзор Microsoft CryptoAPI – Кафедра радиотехники, московский физико-технический институт (МФТИ ГУ).
4. Щербаков Л. Ю. Домашев А. В. Прикладная криптография. Использование и синтез криптографических интерфейсов. – М: Издательско-торговый дом «Русская Редакция», 2003.