МИНОБРНАУКИ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение высшего образования

«ПЕНЗЕНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»

Политехнический институт	Кафедра
Факультет информационных технологий и электроники	«Информационная безопасность систем и технологий»
Специальность 10.05.03 –	Информационная безопасность автоматизированных систем
Специализация –	Защищенные автоматизированные системы управления

ДИПЛОМНЫЙ ПРОЕКТ

на тему

ПРАКТИКУМ ПО ИСПОЛЬЗОВАНИЮ КРИПТОПРОВАЙДЕРОВ ОПЕРАЦИОННЫХ СИСТЕМ СЕМЕЙСТВА WINDOWS

Студент 9.06.2022 (подпись, дата) Корнев Григорий Евгеньевич (ФИО полностью)

Руководитель 9.06.2022 (подпись, дата) Липилин О. В. (фамилия, инициалы)

Нормоконтролёр _____ (подпись, дата) Егорова Н. А. (фамилия, инициалы)

Рецензент Зам. ген. дир. АО „ПНИЭИ" (должность, место работы) _____ (подпись, дата) Семкин С. А. (фамилия, инициалы)

Работа допущена к защите (протокол заседания кафедры от 10.06.22г. № 9) 10 баллов

Заведующий кафедрой _____ (подпись) Зефиров С. Л. (фамилия, инициалы)

Работа защищена с отметкой отлично (протокол заседания ГЭК от 23.06.22г. № 2)

Секретарь ГЭК _____ (подпись) Иванов А. П. (фамилия, инициалы)

Пенза 2022

Оглавление

Введение 4
1 Обзор криптопровайдеров операционных систем семейства Windows 6
1.1 Определение и назначение криптопровайдеров 6
1.2 Криптопровайдеры операционной системы Windows 10 6
2 Обзор возможностей криптографических провайдеров 11
2.1 Возможности криптопровайдеров для обеспечения свойств информации 11
2.2 Возможности криптопровайдеров по управлению ключами и сертификатами 12
2.3 Защита криптопровайдера от внешних воздействий 12
3 Разработка структуры практикума 15
4 Разработка порядка проведения практикума 17
5 Разработка примеров реализации практикума 21
5.1 Алгоритмы реализации функций по использованию криптопровайдеров 21
5.2 Проверка работоспособности примеров реализации функций по использованию криптопровайдеров 30
5.2.1 Проверка работоспособности примера реализации функций по использованию криптопровайдеров для обеспечения конфиденциальности и для управления ключами 30
5.2.2 Проверка работоспособности примера реализации функций по использованию криптопровайдеров для обеспечения целостности 31
5.2.3 Проверка работоспособности примера реализации функций по использованию криптопровайдеров для реализации электронной подписи и для управления сертификатами 32
6 Рекомендации для проведения практикума 34
6.1 Рекомендации по разработке приложения для зашифрования 34

6.2 Рекомендации по разработке приложения для вычисления хэш-кода.... 40
6.3 Рекомендации по разработке приложения для формирования электронной подписи 41
7 Разработка приложения для проверки результатов выполнения практикума. 48
7.1 Описание приложения для проверки результатов выполнения практикума 48
7.2 Проверка работоспособности приложения для проверки результатов выполнения практикума 50
7.2.1 Проверка правильности зашифрования 50
7.2.2 Проверка правильности вычисления хэш-кода 52
7.2.3 Проверка правильности формирования электронной подписи 53
8 Вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации 56
Заключение 58
Список использованных источников 59
Приложение А. Обязательное. Примеры реализации практикума 62
Приложение Б. Обязательное. Текст программы для проверки результатов выполнения практикума 77

Введение

Люди постоянно обмениваются информацией. Различные задачи требуют обеспечения её конфиденциальности, целостности, доступности и других свойств. Подготовка специалистов, разрабатывающих средства криптографической защиты информации, обеспечивающие часть требуемых свойств, занимает много времени ввиду сложности используемых алгоритмов. Использование криптопровайдеров упрощает эту задачу и позволяет уменьшить количество уязвимостей в разрабатываемых приложениях, потому что реализация алгоритмов в этом случае обеспечивается криптопровайдерами.

Для выпускной квалификационной работы была выбрана тема: «Практикум по использованию криптопровайдеров операционных систем семейства Windows».

Данная тема является актуальной, поскольку студентам необходимо получать навыки практической реализации средств криптографической защиты информации с использованием возможностей современных операционных систем. Кроме того, использование криптопровайдеров позволяет выполнять разработку приложений без необходимости получения лицензии на осуществление деятельности, связанной с разработкой криптографических средств, т. к. разработка выполняется с использованием компонентов программных операционных систем, криптографические возможности которых не могут быть изменены пользователем [1].

Ранее в различных работах были рассмотрены следующие варианты применения криптопровайдеров:

– защита информации при предоставлении государственных электронных услуг [2];
– защита систем IP-телефонии [3];
– идентификация пользователя беспроводной сети [4];
– защита документов Microsoft Office посредством пароля [5].

Цель дипломного проекта: разработка практикума по использованию криптопровайдеров операционных систем семейства Windows.

В ходе ВКР решаются задачи, связанные с обеспечением некоторых свойств информации с помощью криптопровайдеров ОС Windows.

Разработанные программы осуществляют зашифрование/расшифрование файлов, формирование хэша и проверку с его помощью целостности файла, формирование и проверку электронной подписи файла, импорт/экспорт ключа, установку сертификатов.

1 Обзор криптопровайдеров операционных систем семейства Windows

1.1 Определение и назначение криптопровайдеров

Криптопровайдер (Cryptography Service Provider, CSP) ОС Windows – это независимый модуль, позволяющий осуществлять криптографические операции в операционных системах Microsoft, управление которым происходит с помощью специальных API (способов, которыми одна компьютерная программа может взаимодействовать с другой программой [6]) [7].

Криптопровайдеры могут использоваться различными приложениями для зашифровывания и расшифровывания данных, обмена ключами, формирования подписи или хэша.

Для взаимодействия с криптопровайдерами ОС Windows могут использоваться различные интерфейсы: CryptoAPI и Cryptography API: Next Generation.

1.2 Криптопровайдеры операционной системы Windows 10

Криптопровайдеры CryptoAPI подразделяются на типы в зависимости от используемых алгоритмов. Типы криптопровайдеров приведены в таблице 1 [8].

Таблица 1 – Типы криптопровайдеров CryptoAPI

Номер	Название	Алгоритмы обмена ключами	Алгоритмы цифровой подписи	Алгоритмы шифрования	Алгоритмы хеширования
1	PROV_RSA_FULL	RSA	RSA	RC2, RC4	MD5, SHA

Продолжение таблицы 1

Номер	Название	Алгоритмы обмена ключами	Алгоритмы цифровой подписи	Алгоритмы шифрования	Алгоритмы хеширования
2	PROV_RSA_SIG	–	RSA	–	MD5, SHA
3	PROV_DSS	–	DSS	–	MD5, SHA
4	PROV_FORTEZZA	KEA	DSS	Skipjack	SHA
5	PROV_MS_EXCHANGE	RSA	RSA	CAST	MD5
6	PROV_SSL	RSA	RSA	Разные	Разные
12	PROV_RSA_SCHANNEL	RSA	RSA	RC4, DES, Triple DES	MD5, SHA
13	PROV_DSS_DH	DH	DSS	CYLINK_MEK	MD5, SHA
18	PROV_DH_SCHANNEL	DHE	DSS	DES, Triple DES	MD5, SHA
24	PROV_RSA_AES	RSA	RSA	RC2, RC4, AES	MD5, SHA

В ОС Windows 10 присутствуют криптопровайдеры CryptoAPI, представленные в таблице 2.

Таблица 2 – Криптопровайдеры CryptoAPI

Тип	Название
1	Microsoft Base Cryptographic Provider
13	Microsoft Base DSS and Diffie-Hellman Cryptographic Provider
3	Microsoft Base DSS Cryptographic Provider

Продолжение таблицы 2

Тип	Название
1	Microsoft Base Smart Card Crypto Provider
18	Microsoft DH/SChannel Cryptographic Provider
1	Microsoft Enhanced Cryptographic Provider
13	Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider
24	Microsoft Enhanced RSA and AES Cryptographic Provider
12	Microsoft RSA/SChannel Cryptographic Provider
1	Microsoft Strong Cryptographic Provider

Microsoft Base Cryptographic Provider – универсальный криптопровайдер (поддерживающий и формирование подписей, и шифрование данных), который использует для всех операций с открытым ключом алгоритм RSA.

Microsoft Enhanced Cryptographic Provider поддерживает те же возможности, что и Microsoft Base Cryptographic Provider, но обеспечивает лучшую защиту за счёт увеличенной длины ключей и дополнительных алгоритмов.

Microsoft Strong Cryptographic Provider используется по умолчанию при выборе типа PROV_RSA_FULL. Он выполняет такие же операции, как Microsoft Enhanced Cryptographic Provider.

Microsoft Enhanced RSA and AES Cryptographic Provider отличается от Microsoft Strong Cryptographic Provider возможностью использования для шифрования алгоритма AES.

Microsoft Base DSS Cryptographic Provider поддерживает формирование хэша, формирование и проверку подписей с помощью алгоритмов SHA, DSS и MD5.

Microsoft Base DSS and Diffie-Hellman Cryptographic Provider поддерживает обмен ключами с помощью протокола Диффи — Хеллмана, формирование хэша с помощью SHA или MD5, формирование и проверку подписи данных с помощью DSS.

Microsoft DH/SChannel Cryptographic Provider поддерживает формирование хэша, подписи; создание, обмен и экспорт ключей с помощью протокола Диффи – Хеллмана. Он может формировать ключи для протоколов SSL3 и TLS1.

Microsoft RSA/SChannel Cryptographic Provider поддерживает формирование хэша, формирование и проверку подписи. Он может использоваться для аутентификации клиента с помощью протоколов SSL 3.0 и TLS 1.0. Может формировать ключ для протоколов SSL2, PCT1, SSL3 и TLS1. Хэш формируется путём конкатенации хэша, полученного с помощью алгоритма MD5, с хэшем, полученным с помощью SHA, затем хэш подписывается с помощью закрытого ключа RSA.

Microsoft Base Smart Card Crypto Provider имеет те же возможности, что и Microsoft Strong Cryptographic Provider, и используется для взаимодействия со смарт-картами.

Microsoft Enhanced DSS and Diffie-Hellman Cryptographic Provider поддерживает обмен ключами с помощью протокола Диффи — Хеллмана, алгоритм SHA для хеширования, DSA для подписи данных и алгоритм симметричного шифрования RC4.

Список криптопровайдеров Cryptography API: Next Generation и их назначение приведены в таблице 3.

Таблица 3 – Криптопровайдеры Cryptography API: Next Generation

Название	Назначение
Microsoft Primitive Provider	Выполнение функций криптографических примитивов
Microsoft Smart Card Key Storage Provider	Хранение ключей асимметричного шифрования и цифровой подписи на смарт-картах
Microsoft Software Key Storage Provider	Хранение ключей асимметричного шифрования и цифровой подписи

Продолжение таблицы 3

Название	Назначение
Microsoft SSL Protocol Provider	Протоколы SSL и TLS

2 Обзор возможностей криптографических провайдеров

2.1 Возможности криптопровайдеров для обеспечения свойств информации

Криптопровайдеры используются для обеспечения различных свойств безопасности информации.

Для обеспечения конфиденциальности информации используется шифрование. Криптопровайдер позволяет разработчику разрабатывать приложения для зашифровывания информации, практически ничего не зная о том, каким образом это зашифровывание осуществляется. Для шифрования данных криптопровайдер использует симметричные ключи. Для передачи симметричный ключ зашифровывается с помощью асимметричного алгоритма. Такая схема используется для обеспечения оптимального быстродействия и защиты. Кроме того, для обмена ключами может быть использован протокол Диффи – Хеллмана [9].

Для обеспечения целостности данных может быть использована цифровая подпись и хэш. Сначала криптопровайдер формирует хэш данных, а затем зашифровывает его с помощью закрытого ключа. Получив данные с такой подписью, получатель может расшифровать подпись с помощью открытого ключа отправителя (получив таким образом хэш, сформированный отправителем), самостоятельно сформировать хэш и сравнить оба хэша. Если они одинаковые, можно сделать вывод о целостности данных и их источнике. Иногда подписанное сообщение требует подписи другой стороны (например, для соглашения с условиями, указанными в сообщении). В этом случае получатель может подписать уже подписанное отправителем сообщение и отправить его обратно. Тогда у отправителя будет доказательство соглашения получателя с сообщением.

Для аутентификации источника данных используются цифровые сертификаты X.509. В сертификате, выдаваемом источнику данных центром сертификации, содержится открытый ключ источника данных. С помощью этого

ключа получатель может проверить подпись, полученную вместе с данными, чтобы убедиться, что их источник – тот, за кого себя выдаёт. Каждый сертификат подписывается закрытым ключом центра сертификации, что позволяет получателю убедиться в подлинности самого сертификата.

2.2 Возможности криптопровайдеров по управлению ключами и сертификатами

Каждый пользователь обычно имеет две пары открытого и закрытого ключей – для шифрования ключей сеанса (симметричных ключей, используемых для шифрования передаваемых данных) и для формирования цифровых подписей. Криптопровайдер не предоставляет способ передачи ключа другому лицу, вместо этого предлагается экспортировать ключ с помощью криптопровайдера, затем любым способом передать его, после чего импортировать на другой стороне с помощью криптопровайдера.

Для передачи данных криптопровайдер может кодировать и декодировать их методом ASN.1 по набору правил DER.

С помощью криптопровайдера можно также управлять сертификатами, установленными в ОС. Существуют функции, управляющие хранилищами сертификатов, и функции, работающие с сертификатами, списками отозванных сертификатов (CRL) и списками доверенных сертификатов (CTL), находящимися в этих хранилищах.

2.3 Защита криптопровайдера от внешних воздействий

Для защиты криптопровайдера от внешних воздействий применяются следующие правила [10]:

– приложения, взаимодействующие с криптопровайдером, не имеют прямого доступа к ключам;

– приложения не могут влиять на реализацию криптографического алгоритма, они могут только выбрать тот или иной алгоритм и его параметры;

– приложения не обрабатывают учётные данные пользователя, это выполняет криптопровайдер;

– криптопровайдер имеет цифровую подпись, проверяемую API при его использовании, что затрудняет изменение или подмену криптопровайдера.

На рисунке 1 приведены варианты использования криптопровайдеров.

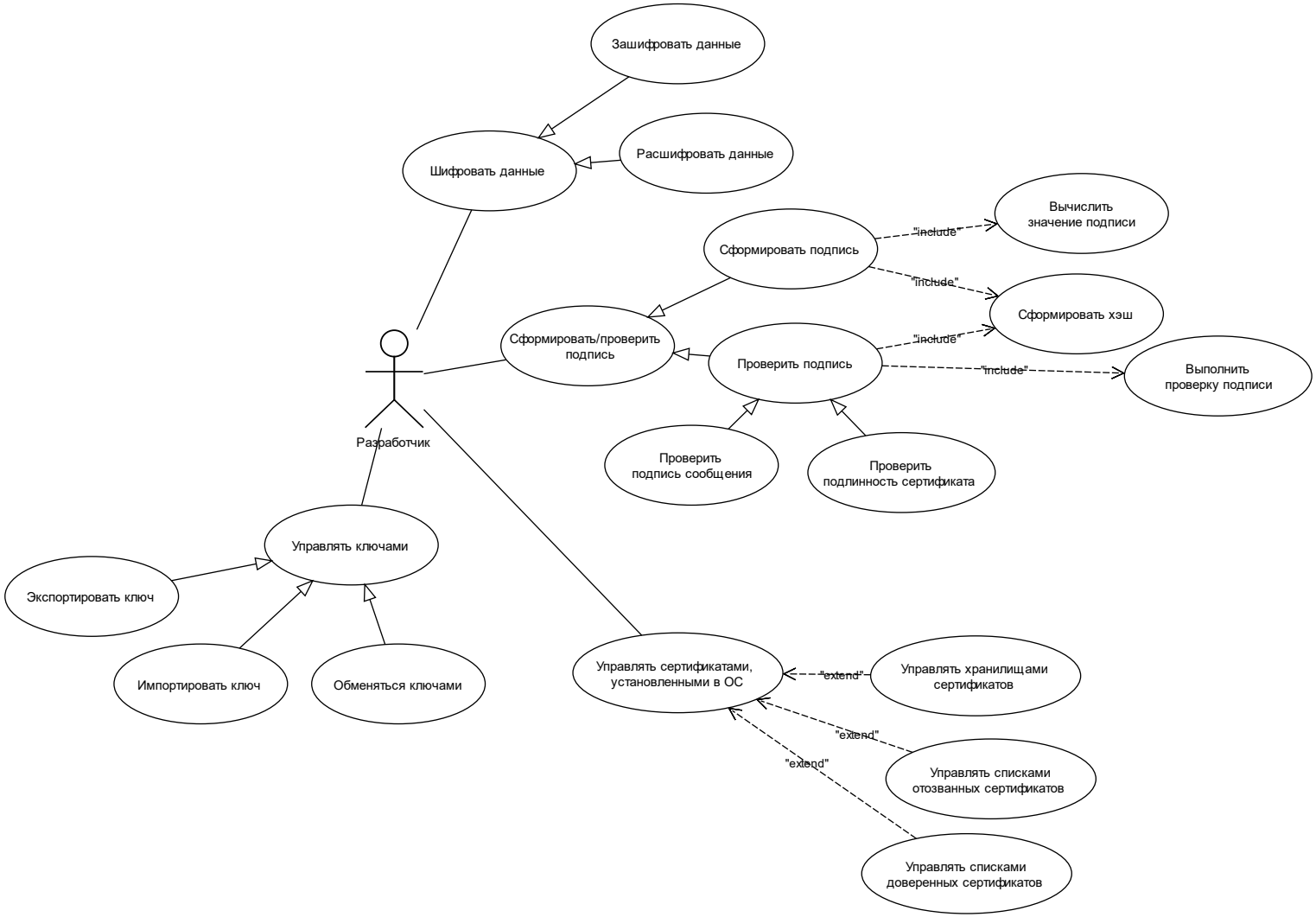

Рисунок 1 – Диаграмма вариантов использования криптопровайдеров

3 Разработка структуры практикума

Практикум предполагает разработку обучаемыми приложений, вычисляющих на основе предоставленной им информации (открытого текста) некой новой информации – закрытого текста, хэш-кода или электронной цифровой подписи. Полученная новая информация передаётся преподавателю для проверки. Проверка выполняется с использованием специализированного приложения.

Для подготовки обучаемых используются алгоритмы реализации функций по применению криптопровайдеров, рассмотренные в разделе 4, а также примеры реализации функций по использованию криптопровайдеров, приведённые в приложении А.

Все криптографические преобразования, выполняемые приложениями, являющимися компонентами практикума или разработанными в ходе его выполнения, выполняются с помощью криптопровайдеров. Обращения к криптопровайдерам совершаются с помощью CryptoAPI или Cryptography API: Next Generation.

Диаграмма компонентов практикума приведена на рисунке 2.

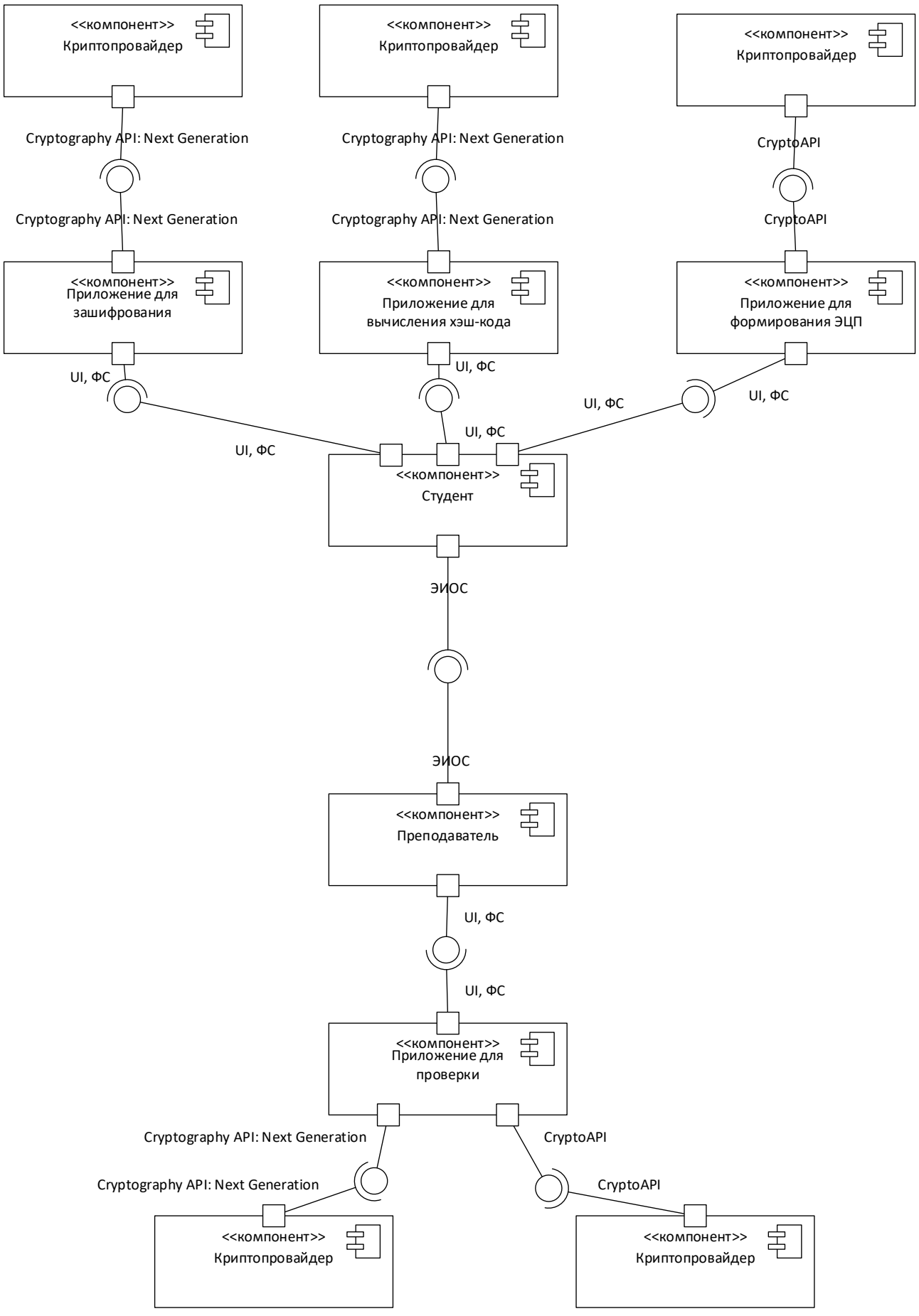

Рисунок 2 – Диаграмма компонентов практикума

4 Разработка порядка проведения практикума

Для проведения практикума обучаемому предоставляется файл для зашифрования (открытый текст) и парольная фраза, необходимая для экспорта/импорта ключа. В начале практикума рассматриваются темы использования криптопровайдеров для обеспечения конфиденциальности данных и управления ключами. Обучаемый должен разработать приложение, выполняющее зашифрование предоставленного ему файла и экспорт ключа, с помощью которого это зашифрование было выполнено. Экспортируемый ключ должен быть зашифрован с помощью ключа, сгенерированного на основе предоставленной обучаемому парольной фразы с использованием алгоритма хэширования SHA256. Шифрование должно осуществляться по алгоритму симметричного шифрования AES с помощью криптопровайдера Microsoft Primitive Provider. После выполнения задания результаты в виде отчёта, шифртекста и ключа должны быть переданы преподавателю для проверки, проводимой с использованием соответствующего приложения.

Следующая тема – использование криптопровайдеров для обеспечения целостности данных. При её изучении обучаемый разрабатывает приложение для вычисления хэш-кода файла с использованием алгоритма SHA256 криптопровайдера Microsoft Primitive Provider и передаёт преподавателю получившийся хэш-код для проверки.

В последнем задании рассматриваются темы использования криптопровайдеров для реализации электронной подписи и для управления сертификатами. Обучаемому необходимо самостоятельно создать сертификаты центра сертификации и пользователя, разработать приложение, а затем с их помощью сформировать подпись для файла. При формировании подписи для хэширования должен использоваться алгоритм MD5, поддерживаемый криптопровайдером Microsoft Base Cryptographic Provider. Для проверки необходимо передать сформированные сертификаты и подпись преподавателю.

Диаграммы последовательности действий выполнения практикума представлены на рисунках 3 и 4.

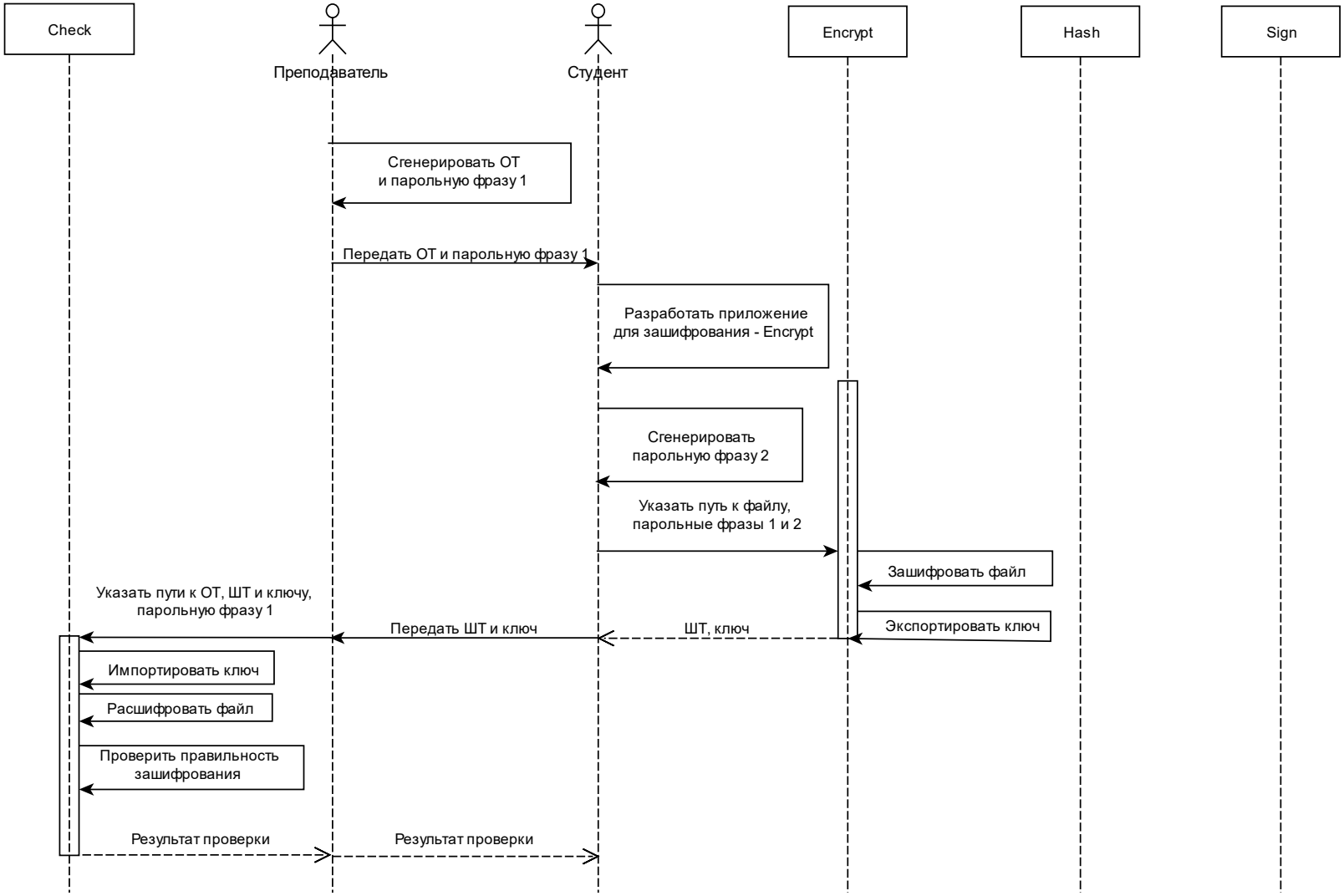

Рисунок 3 – Диаграмма последовательности действий выполнения практикума 1

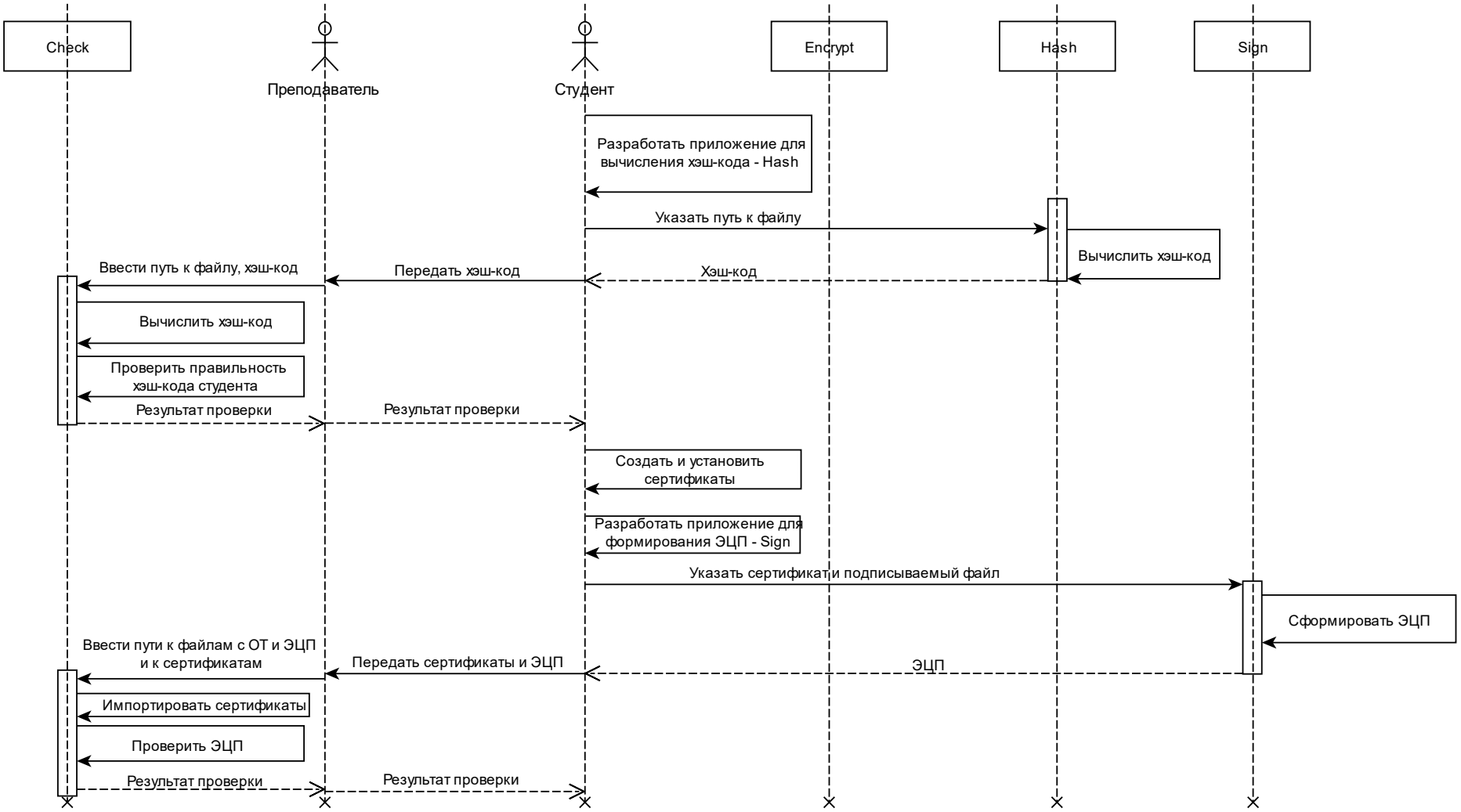

Рисунок 4 – Диаграмма последовательности действий выполнения практикума 2

5 Разработка примеров реализации практикума

5.1 Алгоритмы реализации функций по использованию криптопровайдеров

Были разработаны примеры реализации функций по использованию криптопровайдеров. На рисунке 5 приведён алгоритм реализации функции по использованию криптопровайдеров для зашифрования. Примеры реализации практикума приведены в приложении А.

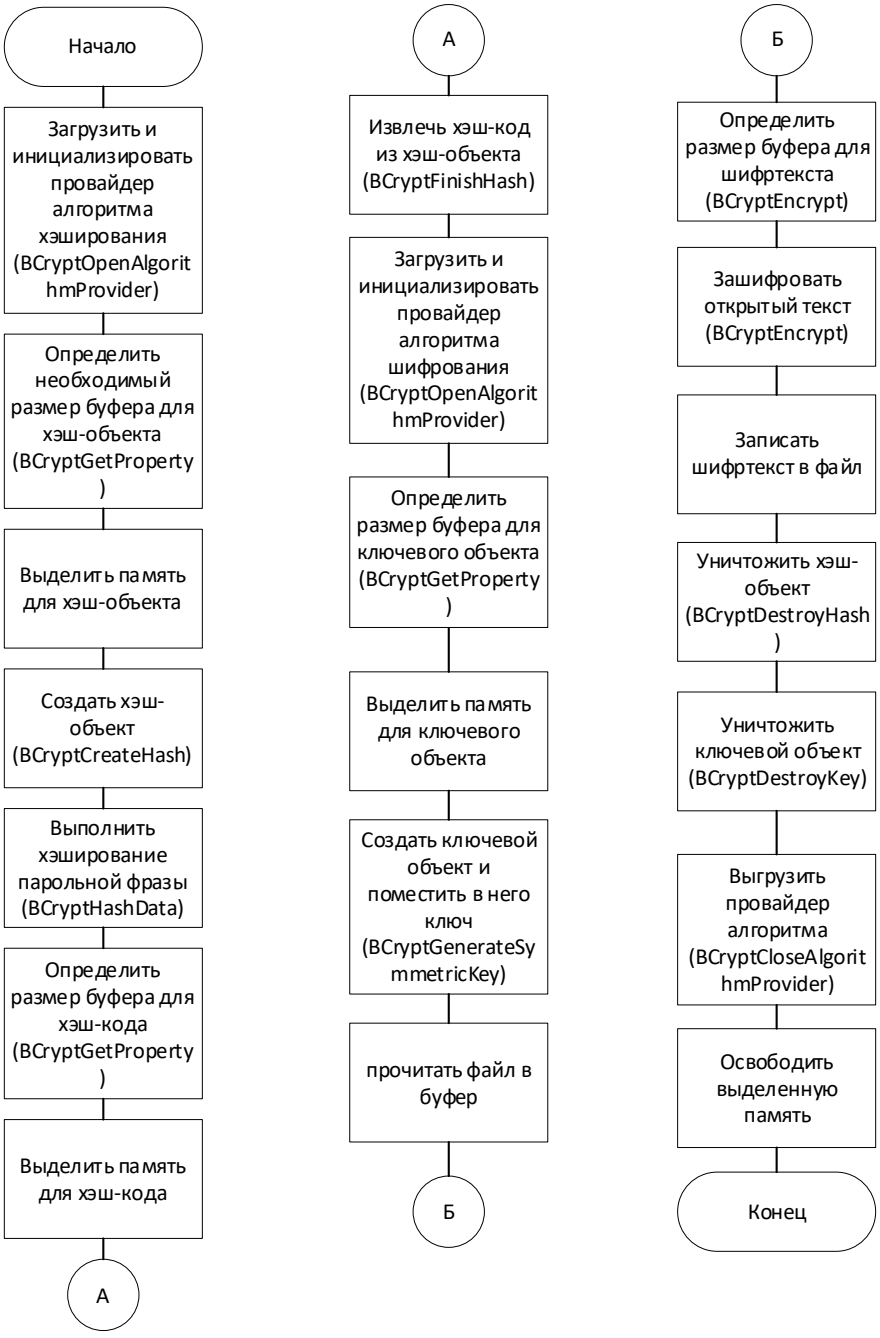

Рисунок 5 – Алгоритм зашифрования

На рисунке 6 приведён алгоритм реализации функции по использованию криптопровайдеров для расшифрования.

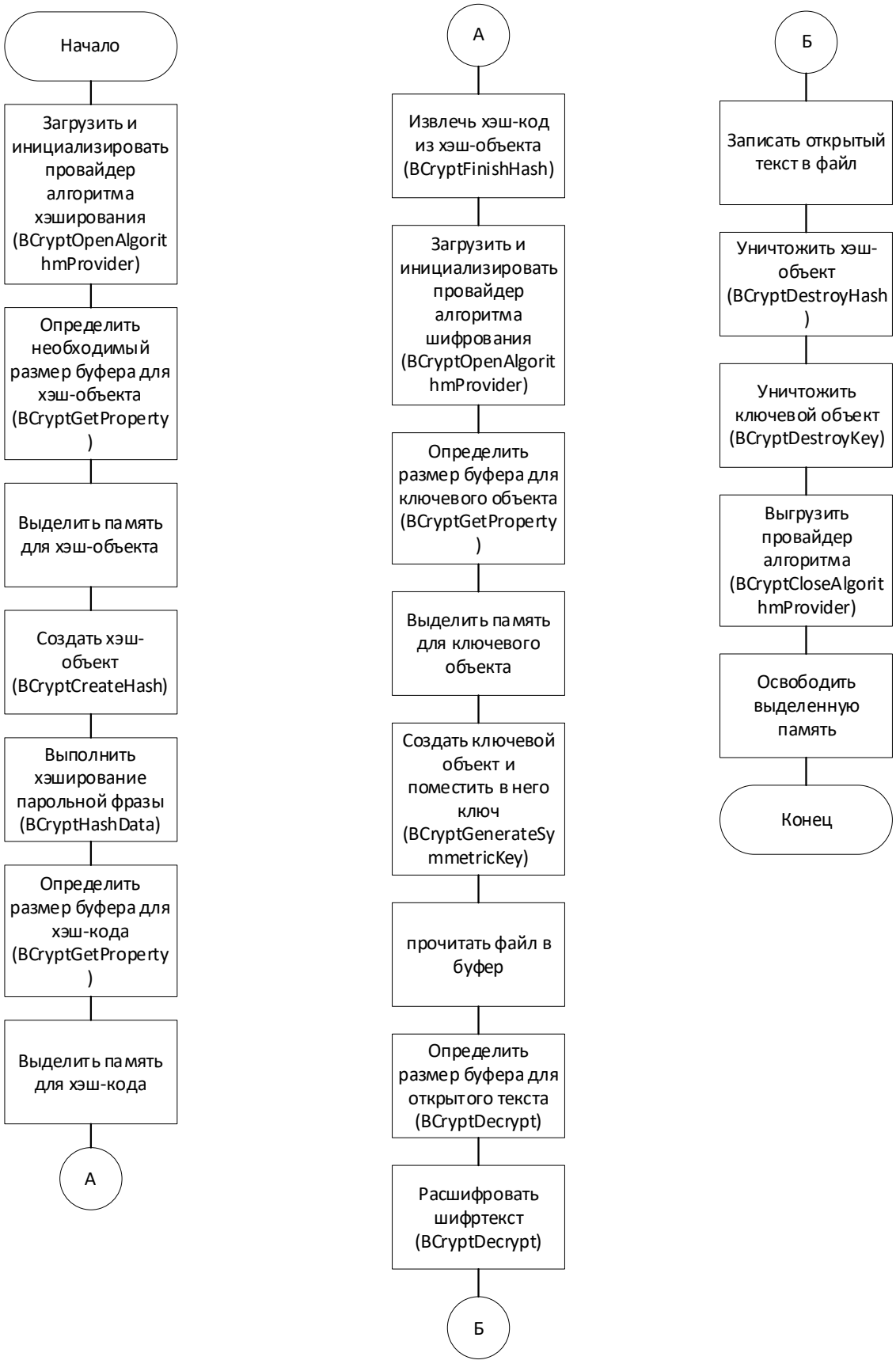

Рисунок 6 – Алгоритм расшифрования

На рисунке 7 приведён алгоритм реализации функции по использованию криптопровайдера для хэширования.

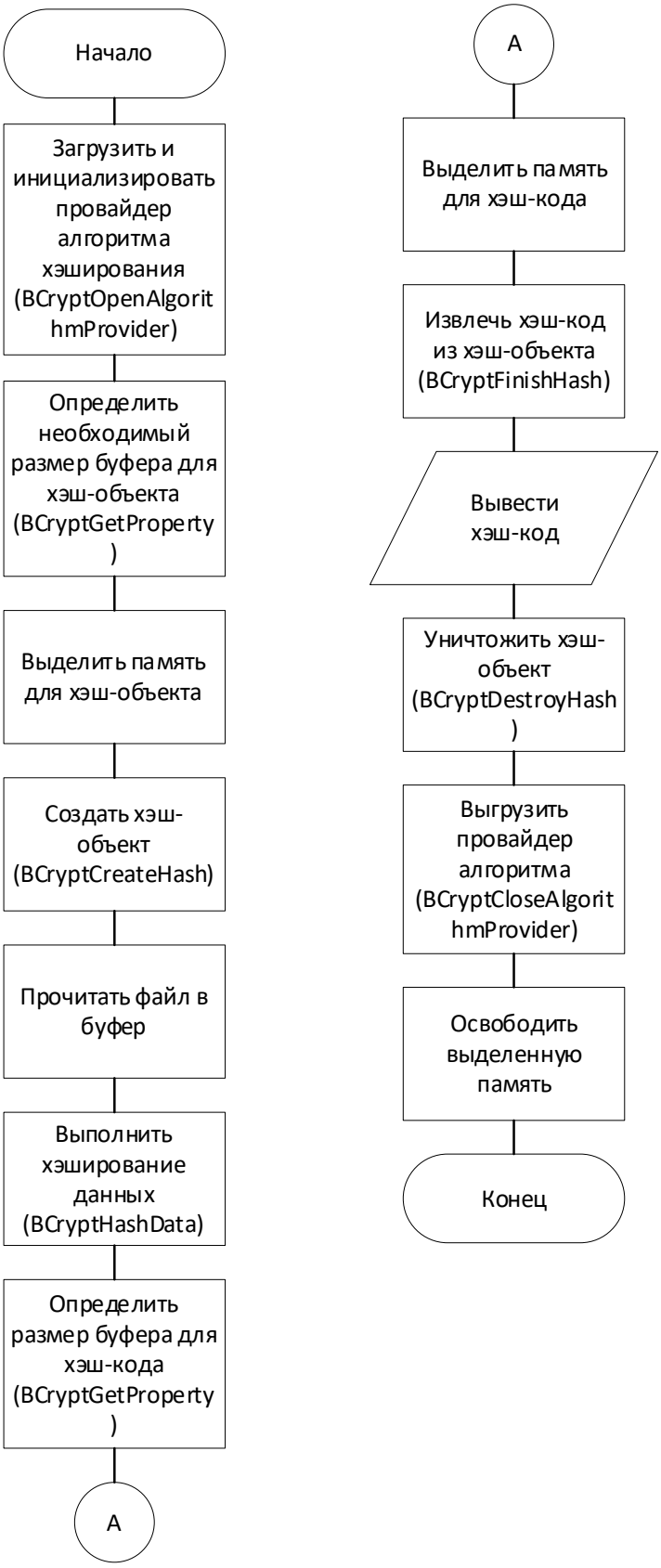

Рисунок 7 – Алгоритм хэширования

На рисунке 8 приведён алгоритм реализации функции по использованию криптопровайдера для проверки целостности файла путём сравнения хэш-кодов.

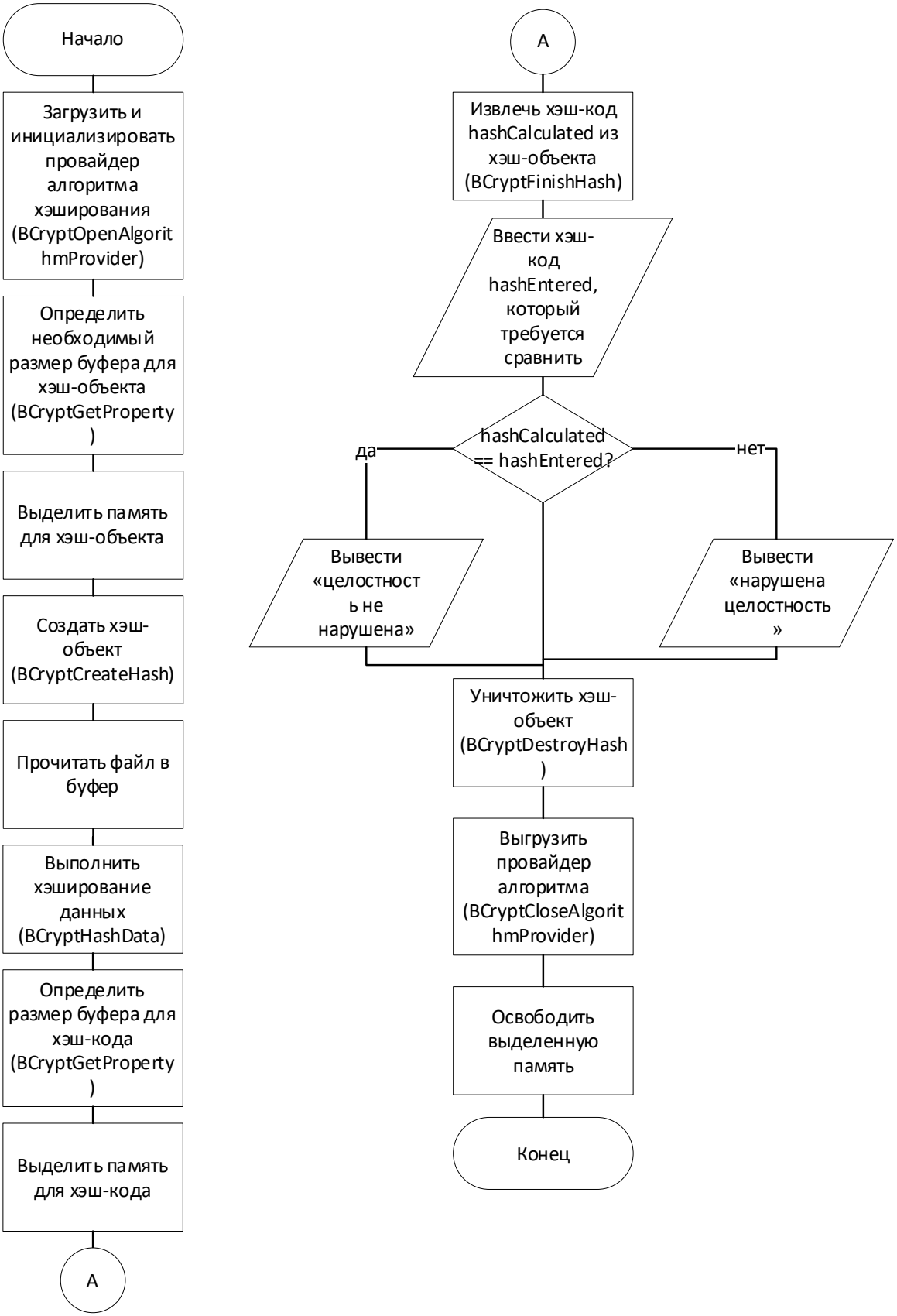

Рисунок 8 – Алгоритм проверки целостности

На рисунке 9 приведён алгоритм реализации функции по использованию криптопровайдера для формирования электронной подписи.

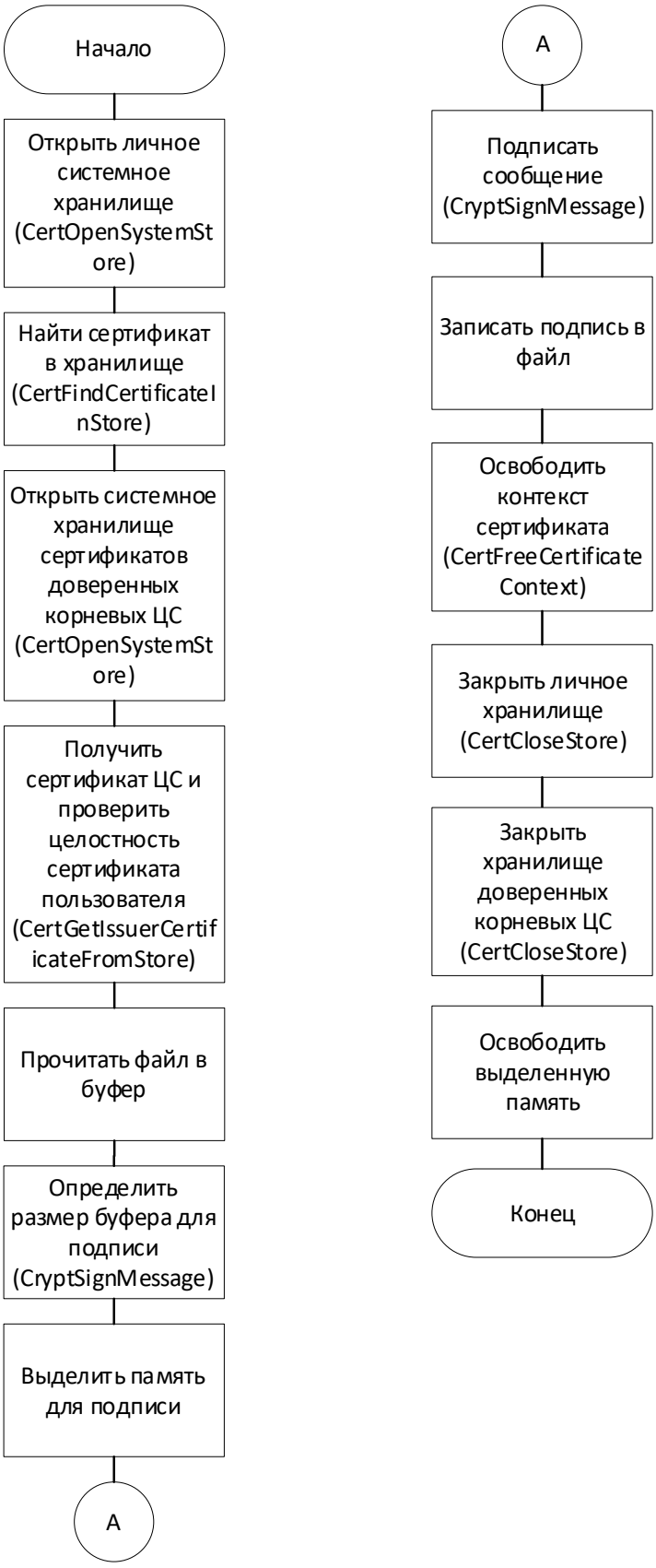

Рисунок 9 – Алгоритм формирования электронной подписи

На рисунке 10 представлен алгоритм реализации функции по использованию криптопровайдера для проверки электронной подписи.

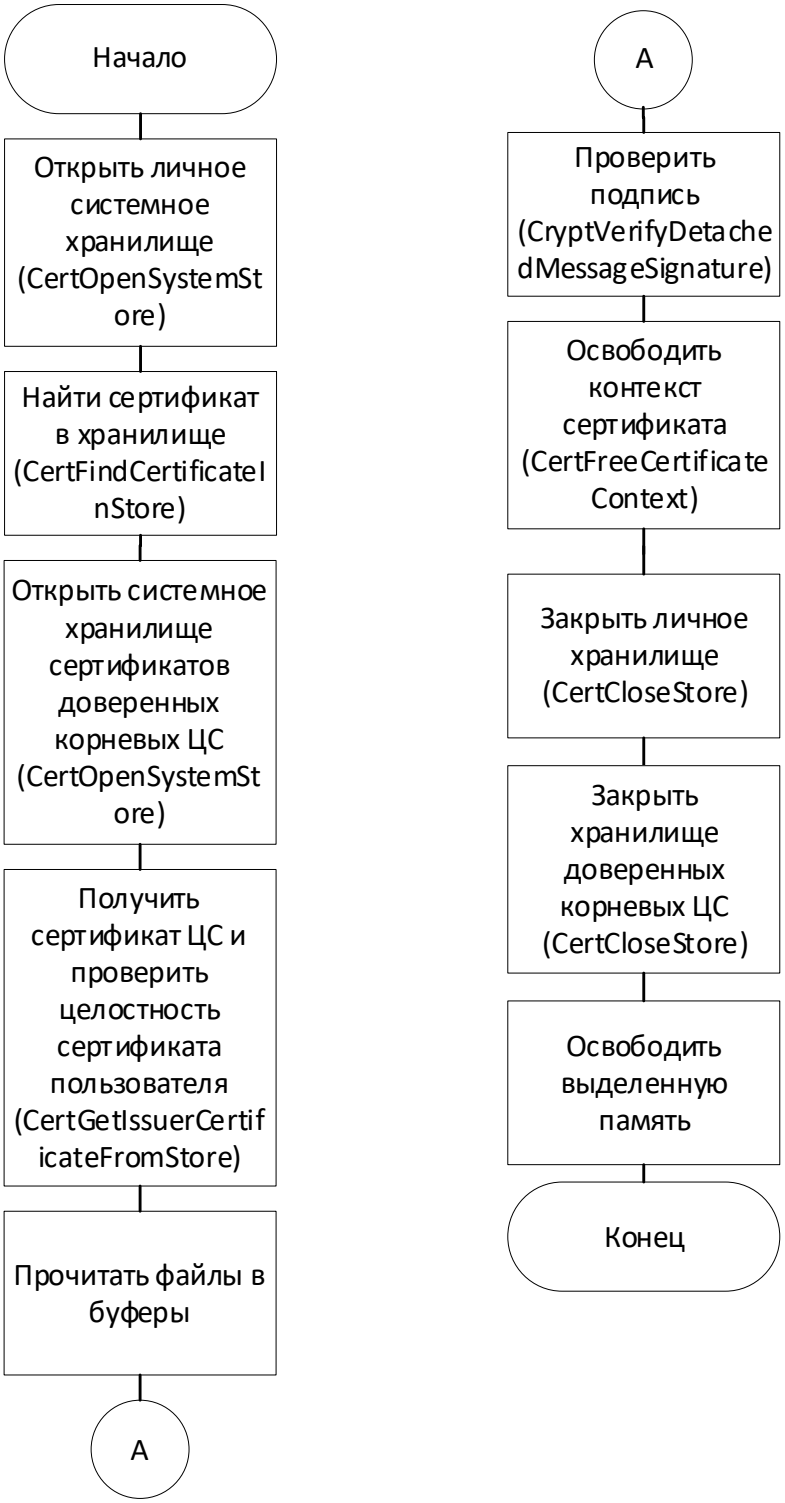

Рисунок 10 – Алгоритм проверки электронной подписи

На рисунке 11 представлен алгоритм реализации функции по использованию криптопровайдера для импорта сертификата. Импортируемый сертификат должен быть в формате DER.

Рисунок 11 – Алгоритм импорта сертификата

На рисунке 12 представлен алгоритм реализации функции по использованию криптопровайдера для экспорта ключа.

На рисунке 13 представлен алгоритм реализации функции по использованию криптопровайдера для импорта ключа.

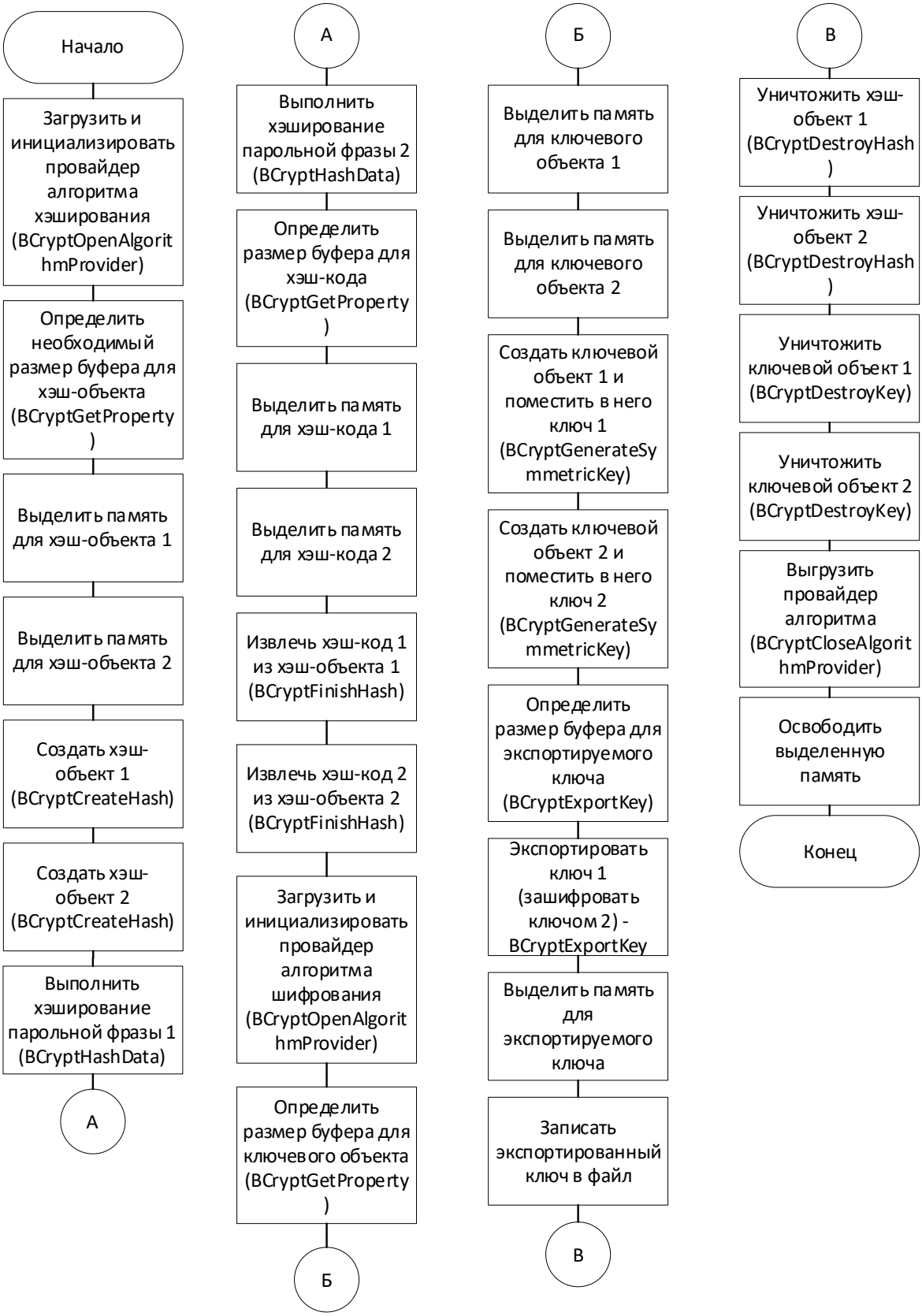

Рисунок 12 – Алгоритм экспорта ключа

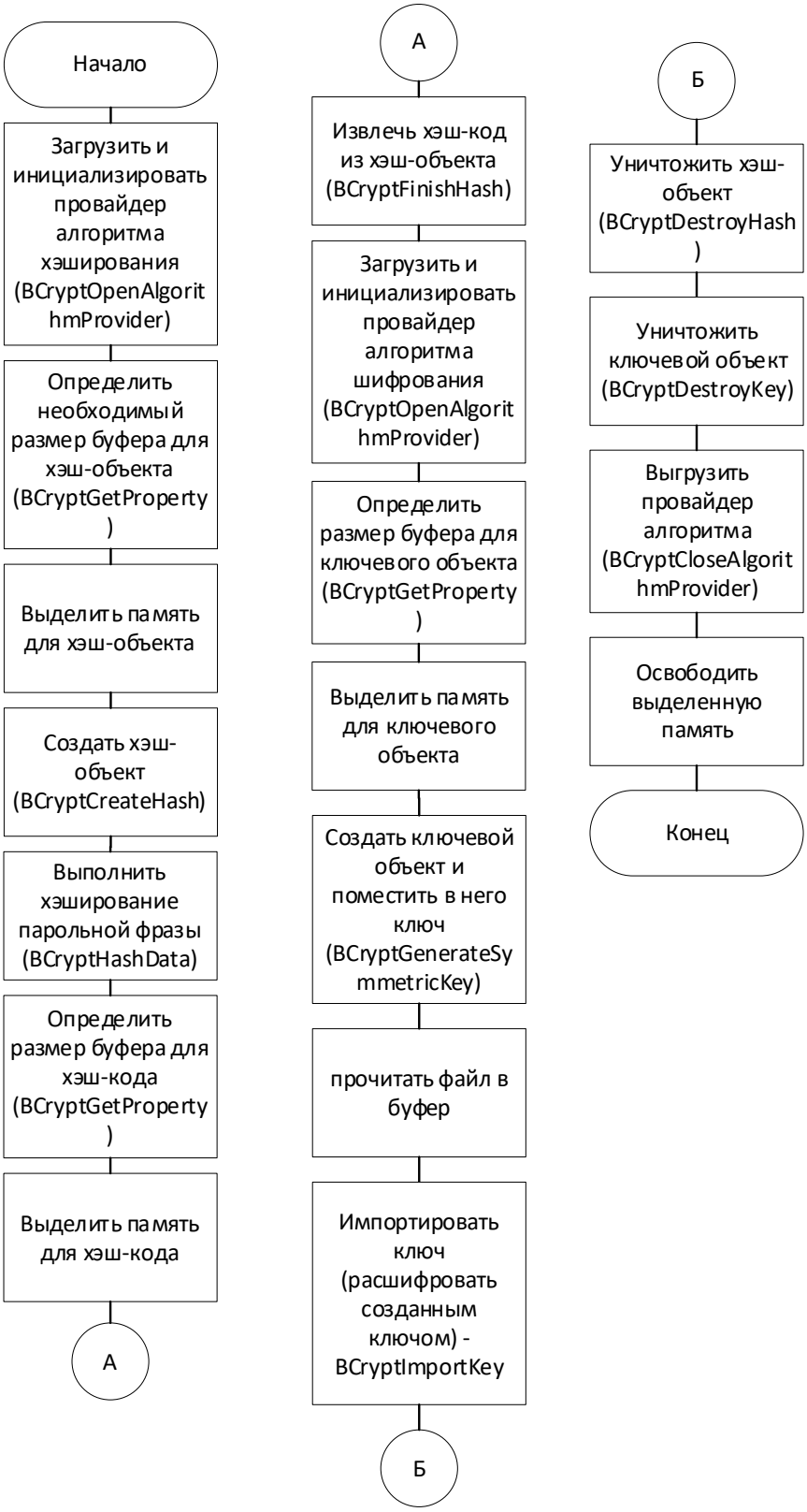

Рисунок 13 – Алгоритм импорта ключа

5.2 Проверка работоспособности примеров реализации функций по использованию криптопровайдеров

5.2.1 Проверка работоспособности примера реализации функций по использованию криптопровайдеров для обеспечения конфиденциальности и для управления ключами

Для проверки работоспособности примера реализации приложения для зашифрования был создан текстовый файл `1.txt`, содержащий строку:

```
один два три четыре пять шесть семь восемь девять десять
```

После запуска программы были введены с клавиатуры пути к файлам:

– `1.txt`, содержащему открытый текст;

– `2.txt`, предназначенному для хранения зашифрованного текста;

– `3.txt`, предназначенному для хранения расшифрованного текста (для проверки правильности зашифрования);

– `1.key`, предназначенному для хранения зашифрованного ключа:

```
plaintext: 1.txt
ciphertext: 2.txt
decrypted: 3.txt
export key to: 1.key
```

Было выполнено зашифрование файла `1.txt`. Результат был записан в файл `2.txt`:

```
Уа(_р9К»ЦQ—d@1Љ>пФ_№с«kUЯ°…6|^————————————————————EgHTE{У‰u%Ud
бФЙ”>пе†-t)±аЗЪЮZwкjHmC¦Ќс-ф±Ґ0>ZЦzвsщrv{BeIфVVMPd-ьђ!HT,uЯ`Ёz
```

Затем содержимое файла `2.txt` было расшифровано и записано в файл `3.txt`:

```
один два три четыре пять шесть семь восемь девять десять
```

Ключ был экспортирован в файл `1.key`. Для зашифрования ключа использовался ключ, сформированный на основе пароля `forExport`.

Так как содержимое файлов `1.txt` и `2.txt` одинаковое, можно сделать вывод о правильном функционировании разработанного примера.

5.2.2 Проверка работоспособности примера реализации функций по использованию криптопровайдеров для обеспечения целостности

Для проверки работоспособности примера реализации приложения для хэширования оно было запущено и был введён путь к файлу `1.txt`. На экран было выведено значение хэша. Затем было предложено ввести хэш-код для сравнения:

```
file: 1.txt
hash: a9275e4a4f2ab88016ce686b9a5c4b996169adfeca4f475a6a2cdbe1d753305c
verification hash:
```

Сначала был введён хэш-код, вычисленный приложением:

```
file: 1.txt
hash: a9275e4a4f2ab88016ce686b9a5c4b996169adfeca4f475a6a2cdbe1d753305c
verification hash:
a9275e4a4f2ab88016ce686b9a5c4b996169adfeca4f475a6a2cdbe1d753305c
==
```

На экран было выведено «==», означающее, что хэш-коды совпадают.

Затем приложение было запущено заново, вычислен хэш-код для файла `1.txt`, но при вводе хэш-кода для сравнения намеренно была допущена ошибка, имитирующая нарушение целостности файла:

```
file: 1.txt
hash: a9275e4a4f2ab88016ce686b9a5c4b996169adfeca4f475a6a2cdbe1d753305c
verification hash:
a9275e4a4f2ab88016c3686b9a5c4b996169adfeca4f475a6a2cdbe1d753305c
!=
```

На экран было выведено «!=», означающее, что целостность файла была нарушена.

5.2.3 Проверка работоспособности примера реализации функций по использованию криптопровайдеров для реализации электронной подписи и для управления сертификатами

Для проверки работоспособности примера реализации приложения для формирования электронной подписи приложение было запущено, были введены пути к файлам `1.txt` и `2.txt`. В файле `1.txt` содержалась строка:

```
один два три четыре пять шесть семь восемь девять десять
```

Файл `2.txt` был предназначен для сохранения сформированной электронной подписи и для чтения из него электронной подписи для проверки:

```
file: 1.txt
sign out: 2.txt
sign  in: 2.txt
881818D5-53FD-40B3-80D7-161D1EC19132
User_A
User_A
User_B
signature verified
```

На экран были выведены названия сертификатов, установленных в ОС, затем было выведено сообщение «signature verified», означающее, что проверка электронной подписи прошла успешно.

После этого была сформирована подпись для другого файла `n.txt`, содержащего строку:

```
0123456789
```

Сформированная подпись была записана в файл `3.txt`, а для проверки подписи была использована подпись файла `1.txt`, содержащаяся в файле `2.txt`:

```
file: n.txt
sign out: 3.txt
sign  in: 2.txt
881818D5-53FD-40B3-80D7-161D1EC19132
User_A
User_A
User_B
invalid signature 80090006
```

Было выведено сообщение о неверной подписи, сопровождающееся кодом ошибки.

На основании полученных данных можно сделать вывод, что пример приложения для формирования электронной подписи работает верно.

6 Рекомендации для проведения практикума

6.1 Рекомендации по разработке приложения для зашифрования

В приведённом алгоритме зашифрования используется криптографический интерфейс Cryptography API: Next Generation (CNG). Для его использования при разработке приложения необходимо подключить библиотеку `bcrypt.dll`. Для этого можно указать в первой строке программы:

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

Также необходимо включить заголовочные файлы `windows.h` и `bcrypt.h`:

```
#include <windows.h>
#include <bcrypt.h>
```

Для использования строк, используемых функциями CNG, нужно включить файл `<tchar.h>`.

Инициализировать такую строку можно следующим образом:

```
TCHAR szPass[] = TEXT("password");
```

В функциях CNG используются специальные типы данных:

```
BCRYPT_ALG_HANDLE hAlg; // дескриптор провайдера алгоритма
BCRYPT_KEY_HANDLE hKey; // дескриптор ключа
BCRYPT_HASH_HANDLE hHash; // дескриптор хэш-объекта
```

Функции CNG возвращают код ошибки типа `NTSTATUS`. Для обработки ошибок можно завести специальную переменную:

```
NTSTATUS status = 0;
```

и использовать её при вызове функций CNG:

```
if (!BCRYPT_SUCCESS(status = BCryptHashData( // функция CNG
      hHash,
      (PBYTE)szPass,
      _tcslen(szPass) * sizeof(TCHAR),
      0)))
{
      cout << "BCryptHashData error 0x" << hex << status;
}
```

Далее приведено описание функций, используемых в алгоритме, представленном на рисунке 5 (алгоритме зашифрования).

Функция **`BCryptOpenAlgorithmProvider`** загружает и инициализирует провайдер алгоритма:

```
NTSTATUS WINAPI BCryptOpenAlgorithmProvider(
```

```
    BCRYPT_ALG_HANDLE *phAlgorithm, // сюда запишется дескриптор → 
провайдера алгоритма
    LPCWSTR pszAlgId, // идентификатор (имя) алгоритма
    LPCWSTR pszImplementation, // имя провайдера криптографических → 
примитивов
    DWORD dwFlags); //
```

Для предлагаемого зашифрования используются такие идентификаторы алгоритма:

– `BCRYPT_AES_ALGORITHM` (для зашифрования);

– `BCRYPT_SHA256_ALGORITHM` (для хэширования).

В качестве имени провайдера криптографических примитивов можно указать `NULL` – тогда будет использоваться провайдер по умолчанию.

Параметр `dwFlags` определяет флаги, влияющие на поведение функции. В данном случае и далее во всех функциях данный параметр должен быть равен `NULL` или `0`, если не указано иное.

Затем необходимо определить размер буфера, необходимый для размещения хэш-объекта. Это делается с помощью функции **`BCryptGetProperty`**, позволяющей получить свойство того или иного объекта CNG. Пример вызова этой функции для получения длины объекта алгоритма хэширования:

```
    DWORD cbHashObject = 0, cbData = 0;
    if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject → 
дескриптор объекта, значение свойства которого необходимо получить
        BCRYPT_OBJECT_LENGTH, // pszProperty имя свойства
        (PBYTE)&cbHashObject, // pbOutput сюда передастся значение → 
свойства
        sizeof(DWORD), // cbOutput размер буфера pbOutput
        &cbData, // *pcbResult адрес переменной, куда возвращается → 
размер данных, реально скопированных в pbOutput
        0))) // dwFlags 0
    {
        cout << "hash object buffer creation error 0x" << hex << 
status;
    }
```

Выделить память для хэш-объекта можно с помощью команды:

```
PBYTE pbHashObject = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHashObject);
```

Создаётся хэш-объект с помощью функции **`BCryptCreateHash`**:

```
NTSTATUS WINAPI BCryptCreateHash(
    BCRYPT_ALG_HANDLE hAlgorithm, // дескриптор алгоритма хэширования
    BCRYPT_HASH_HANDLE *phHash, // указатель на дескриптор → 
создаваемого хэш-объекта
```

```
	PUCHAR pbHashObject, // указатель на буфер, в котором размещается → создаваемый хэш-объект
	ULONG cbHashObject, // размер буфера pbHashObject
	PUCHAR pbSecret, // NULL
	ULONG cbSecret, // 0
	ULONG dwFlags); // 0
```

Хэширование парольной фразы выполняет функция **BCryptHashData**:

```
	if (!BCRYPT_SUCCESS(status = BCryptHashData(hHash, // hHash → дескриптор хэш-объекта
		(PBYTE)szPass, // pbInput адрес буфера с входными данными
		_tcslen(szPass) * sizeof(TCHAR), // cbInput размер pbInput
		0))) // dwFlags
	{
		cout << "BCryptHashData error 0x" << hex << status;
	}
```

Определить размер буфера и выделить память для хэш-кода можно аналогично определению размера буфера и выделению памяти для хэш-объекта.

Для извлечения хэш-кода из хэш-объекта используется функция **BCryptFinishHash**:

```
NTSTATUS WINAPI BCryptFinishHash(
	BCRYPT_HASH_HANDLE hHash, // дескриптор хэш-объекта
	PUCHAR pbOutput, // буфер для принятия хэш-кода
	ULONG cbOutput, // размер буфера pbOutput
	ULONG dwFlags); // 0
```

Загрузить и инициализировать провайдер алгоритма шифрования можно с помощью функции **BCryptOpenAlgorithmProvider**, рассмотренной ранее.

Размер буфера для ключевого объекта определяется с помощью функции **BCryptGetProperty**, рассмотренной ранее.

После выделения памяти для ключевого объекта аналогично выделению памяти для хэш-кода и хэш-объекта необходимо создать ключевой объект и поместить в него ключ с помощью функции **BCryptGenerateSymmetricKey**. Пример вызова:

```
BCRYPT_KEY_HANDLE hKey;
if (!BCRYPT_SUCCESS(status = BCryptGenerateSymmetricKey(hAlg, // → hAlgorithm полученный с помощью BCryptOpenAlgorithmProvider дескриптор → алгоритма шифрования
		&hKey, // *phKey дескриптор создаваемого ключевого объекта
		pbKeyObject, // pbKeyObject указатель на буфер, в котором → размещается созданный ключевой объект
		cbKeyObject, // cbKeyObject размер буфера pbKeyObject
```

```
            (PBYTE)pbHash, // pbSecret указатель на буфер, который →
содержит ключевой материал (буфер с хэш-кодом)
            16, // cbSecret размер буфера bpSecret
            0))) // dwFlags
      {
            cout << "BCryptGenerateSymmetricKey error 0x" << hex <<
status;
      }
```

Прочитать файл в буфер можно следующим образом:

```
int length = 0; // длина файла
tail = 16; //
BYTE *buffer;
ifstream iF(plainTextFilePath, ios_base::in | ios_base::binary);

iF.seekg(0, iF.end); //
length = iF.tellg(); // определение длины файла
iF.seekg(0, iF.beg); //

buffer = new BYTE [length + tail];

iF.read((char*)buffer, length); // прочитать файл в буфер
```

Определить размер буфера для шифртекста можно с помощью функции, выполняющей зашифрование – **BcryptEncrypt**:

```
NTSTATUS WINAPI BCryptEncrypt(
      BCRYPT_KEY_HANDLE hKey, // дескриптор созданного ранее ключа
      PUCHAR pbInput, // адрес буфера с открытым текстом
      ULONG cbInput, // размер буфера с открытым текстом
      VOID *pPaddingInfo, // указатель на структуру, определяющую →
параметры дополнения блоков
      PUCHAR pbIV, // адрес буфера, содержащего вектор инициализации
      ULONG cbIV, // размер вектора инициализации
      PUCHAR pbOutput, // адрес буфера, в который будет помещаться →
шифртекст
      ULONG cbOutput, // размер буфера для шифртекста
      ULONG *pcbResult, // размер получившегося шифртекста
      ULONG dwFlags); // BCRYPT_BLOCK_PADDING
```

Параметр `pPaddingInfo` в данном случае (обычного симметричного шифрования) должен быть равен NULL.

Параметр `pbIV` должен быть равен `NULL`, а `cbIV` – `0`, т. к. в данном случае используется нулевой вектор инициализации.

В `pbOutput` при первом вызове (для определения размера буфера) необходимо указать `NULL`, а в `cbOutput` – `0`. При повторном вызове (для

зашифрования) в `cbOutput` нужно указать `*pcbResult`, а в `pbOutput` – адрес буфера.

Параметр `dwFlags` должен быть равен `BCRYPT_BLOCK_PADDING`, так как неизвестно, будет ли размер входных данных кратен размеру блока алгоритма симметричного шифрования.

Записать шифртекст в файл можно так:

```
ofstream oF(cipherTextFilePath, ios_base::out | ios_base::binary);
oF.write((char*)buffer, *pcbResult); // записать зашифрованные данные в → файл
```

Уничтожение хэш-объекта выполняется функцией **`BCryptDestroyHash`**:

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

Уничтожение ключевого объекта – функция **`BCryptDestroyKey`**:

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

Выгрузка провайдера алгоритма – функция **`BCryptCloseAlgorithmProvider`**:

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

Освобождение выделенной памяти:

```
HeapFree(GetProcessHeap(), 0, pbHashObject); // освобождение буфера хэш-объекта
```

При расшифровании используются уже рассмотренные функции, а также функция **`BCryptDecrypt`**:

```
NTSTATUS WINAPI BCryptDecrypt(
	BCRYPT_KEY_HANDLE hKey, // дескриптор созданного ранее ключа
	PUCHAR pbInput, // адрес буфера с шифртекстом
	ULONG cbInput, // размер буфера с шифртекстом
	VOID *pPaddingInfo, // указатель на структуру, определяющую → параметры дополнения блоков
	PUCHAR pbIV, // адрес буфера, содержащего вектор инициализации
	ULONG cbIV, // размер вектора инициализации
	PUCHAR pbOutput, // адрес буфера, в который будет помещаться → расшифрованный текст
	ULONG cbOutput, // размер буфера для расшифрованного текста
	ULONG *pcbResult, // размер получившегося расшифрованного текста
	ULONG dwFlags); // BCRYPT_BLOCK_PADDING
```

Эта функция также вызывается два раза: первый – для определения размера буфера, второй – для расшифрования.

Для экспорта ключа используется функция **`BCryptExportKey`**:

```
NTSTATUS BCryptExportKey(
BCRYPT_KEY_HANDLE hKey, // дескриптор экспортируемого ключа
BCRYPT_KEY_HANDLE hExportKey, // дескриптор ключа, используемого для →
зашифрования hKey
LPCWSTR pszBlobType, // тип BLOB-объекта
PUCHAR pbOutput, // буфер для результата
ULONG cbOutput, // размер pbOutput
ULONG *pcbResult, // количество скопированных байт
ULONG dwFlags // NULL
);
```

Сначала параметры `pbOutput` и `cbOutput` можно установить равными `NULL` (при определении необходимого размера буфера для экспортируемого ключа). Затем (при экспорте ключа) нужно заполнить их в соответствии с полученным значением `*pcbResult`. Пример вызова функции:

```
if (!BCRYPT_SUCCESS(status = BCryptExportKey( // экспорт ключа
      hKey, // hKey дескриптор экспортируемого ключа
      hKeyExp, // hExportKey дескриптор ключа, используемого для
зашифрования hKey
      BCRYPT_AES_WRAP_KEY_BLOB, // pszBlobType тип BLOB-объекта,
      expKeyBuf, // pbOutput буфер для результата
      cbOutputExp, // cbOutput размер pbOutput
      &cbOutputExp, // *pcbResult количество скопированных байт
      NULL))) // dwFlags
{
      cout << "BCryptExportKey error 0x" << hex << status << endl;
}
```

Импорт ключа выполняет функция **`BCryptImportKey`**:

```
NTSTATUS BCryptImportKey(
BCRYPT_ALG_HANDLE hAlgorithm, // дескриптор алгоритма
BCRYPT_KEY_HANDLE hImportKey, // дескриптор ключа, используемого для →
расшифрования импортируемого ключа
LPCWSTR pszBlobType, // тип BLOB-объекта
BCRYPT_KEY_HANDLE *phKey, // сюда запишется импортированный ключ
PUCHAR pbKeyObject, // NULL
ULONG cbKeyObject, // размер pbKeyObject
PUCHAR pbInput, // буфер с импортируемым ключом
ULONG cbInput, // размер pbInput
ULONG dwFlags // NULL
);
```

Пример вызова функции:

```
if (!BCRYPT_SUCCESS(status = BCryptImportKey( // импорт ключа
      hAlg, // hAlgorithm дескриптор алгоритма
      hKey, // hImportKey дескриптор ключа, используемого для
расшифрования импортируемого ключа
      BCRYPT_AES_WRAP_KEY_BLOB, // pszBlobType тип BLOB-объекта,
      &hKeyImp, // *phKey сюда запишется импортированный ключ
```

```
        NULL, // pbKeyObject
        0, // cbKeyObject размер pbKeyObject
        impKeyBuf, // pbInput буфер с импортируемым ключом
        length, // cbInput размер pbInput
        NULL))) // dwFlags
    {
        cout << "BCryptImportKey error 0x" << hex << status << endl;
    }
```

6.2 Рекомендации по разработке приложения для вычисления хэш-кода

Хэширование выполняется аналогично хэшированию парольной фразы для генерации ключа при зашифровании. Нужно использовать алгоритм SHA-256. Вывод хэш-кода на экран можно реализовать следующим образом (`cbHash` – размер буфера с хэш-кодом, `pbHash` – буфер с хэш-кодом):

```
#include <iostream>
#include <sstream>
ostringstream oStrHash;
    cout << "hash: ";
    for (int i = 0; i < cbHash; i++)
    {
        oStrHash << hex << (int)pbHash[i];
    }
    cout << oStrHash.str() << endl;
```

Сравнить хэш-коды при проверке целостности можно так:

```
    cout << "verification hash: ";
    cin >> hash;
    if (oStrHash.str() == hash)
    {
        cout << "==" << endl;
    }
    else
    {
        cout << "!=" << endl;
    }
```

6.3 Рекомендации по разработке приложения для формирования электронной подписи

Далее будут рассмотрены функции, используемые для формирования электронной подписи. Они используют интерфейс CryptoAPI. Для их использования необходимо подключить библиотеку `crypt32.dll`. Для этого можно указать в первой строке программы:

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

Также необходимо включить заголовочные файлы:

```
#include <windows.h>
#include <wincrypt.h>
```

Все функции CryptoAPI возвращают `TRUE` в случае успеха и `FALSE` в случае ошибки. При возникновении ошибки её номер можно узнать с помощью функции **`GetLastError`**, например:

```
	if (!CryptSignMessage(&SignPara, TRUE, 1, &buffers, &tbsLength, 
NULL, &cbOutput))
	{
		cout << "CryptSignMessage buffer size define error 0x" << hex 
<< GetLastError();
	}
```

Для открытия личного системного хранилища используется функция **`CertOpenSystemStore`**:

```
HCERTSTORE WINAPI CertOpenSystemStore(
	HCRYPTPROV_LEGACY hprov, // NULL
	LPTCSTR szSubsystemProtocol ); // имя системного хранилища → 
сертификатов:
                                       // TEXT("MY") – Личное
                                       // TEXT("ROOT") – Доверенные → 
                                       корневые центры сертификации
```

Данная функция возвращает дескриптор открытого хранилища, в случае ошибки – NULL.

Поиск сертификата в хранилище выполняет функция **`CertFindCertificateInStore`**:

```
PCCERT_CONTEXT WINAPI CertFindCertificateInStore(
	HCERTSTORE hCertStore, // дескриптор открытого хранилища → 
сертификатов
```

```
      DWORD dwCertEncodingType, // PKCS_7_ASN_ENCODING | →
X509_ASN_ENCODING
      DWORD dwFindFlags, // 0
      WORD dwFindType, // тип поиска (CERT_FIND_SUBJECT_STR_A – поиск →
по имени)
      const void *pvFindPara, // указатель на строку с именем искомого →
сертификата (если поиск по имени)
      PCCERT_CONTEXT pPrevCertContext ); // указатель на контекст →
предыдущего сертификата (NULL)
```

Пример вызова:

```
#define MY_ENCODING_TYPE (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)
TCHAR szCertNameA[] = TEXT("User_A"), // имя сертификата A
PCCERT_CONTEXT pCertA = NULL, // контекст сертификата A
      if (!(pCertA = CertFindCertificateInStore( // найти сертификат
            hStoreMy,  // hCertStore дескриптор открытого хранилища
сертификатов
            MY_ENCODING_TYPE, // dwCertEncodingType
            0, // dwFindFlags
            CERT_FIND_SUBJECT_STR_A, // dwFindType тип поиска (поиск по
имени)
            szCertNameA, // *pvFindPara указатель на строку с именем
искомого сертификата (если CERT_FIND_SUBJECT_STR)
            NULL))) // pPrevCertContext указатель на контекст предыдущего
сертификата
      {
            cout << "CertFindCertificateInStore User_A error" << endl;
      }
```

Функция для открытия системного хранилища сертификатов доверенных корневых центров сертификации была рассмотрена ранее.

Получить сертификат центра сертификации и проверить целостность сертификата пользователя можно с помощью функции **CertGetIssuerCertificateFromStore**:

```
PCCERT_CONTEXT WINAPI CertGetIssuerCertificateFromStore(
      HCERTSTORE hCertStore, // дескриптор открытого хранилища, в →
котором ищется сертификат ЦС
      PCCERT_CONTEXT pSubjectContext, // контекст пользовательского →
сертификата, для которого ищется сертификат ЦС
      PCCERT_CONTEXT pPrevIssuerContext, // контекст предыдущего →
найденного сертификата этого ЦС (NULL)
      DWORD *pdwFlags ); // характер проверки пользовательского →
сертификата
```

В качестве `*pdwFlags` можно использовать `CERT_STORE_SIGNATURE_FLAG | CERT_STORE_TIME_VALIDITY_FLAG`. Это позволит провести верификацию

электронной подписи пользовательского сертификата открытым ключом центра сертификации и проверить, не истёк ли срок действия сертификата.

Функция возвращает контекст найденного сертификата (или NULL, если он не найден). Если после выполнения pdwFlags равен нулю, значит, сертификат прошёл проверку.

Чтение файла в буфер было рассмотрено ранее.

Чтобы определить размер буфера для подписи, нужно использовать функцию **CryptSignMessage**, которая также используется для подписи:

```
BOOL CryptSignMessage(
  PCRYPT_SIGN_MESSAGE_PARA pSignPara, // параметры электронной подписи
  BOOL fDetachedSignature, // будет только подпись (true) или сообщение → 
тоже (false)
  DWORD cToBeSigned, // количество элементов в rgpbToBeSigned[] и → 
rgcbToBeSigned[]
  const BYTE * [] rgpbToBeSigned, // [массив] указателей на буферы, → 
содержащие то, что нужно подписать
  DWORD []  rgcbToBeSigned, // rgcbToBeSigned [массив] размеров → 
буферов, указанных в rgpbToBeSigned
  BYTE *pbSignedBlob, // буфер для результата
  DWORD *pcbSignedBlob); // размер *pbSignedBlob
```

Параметр pSignPara можно инициализировать следующим образом:

```
CRYPT_SIGN_MESSAGE_PARA SignPara = { sizeof(CRYPT_SIGN_MESSAGE_PARA) };
SignPara.dwMsgEncodingType = PKCS_7_ASN_ENCODING | X509_ASN_ENCODING;
      SignPara.pSigningCert = pCertA; // сертификат
      SignPara.HashAlgorithm.pszObjId = (LPSTR)CertAlgIdToOID(CALG_MD5); 
// алгоритм для хэширования
      SignPara.cMsgCert = 1; // количество элементов в rgpMsgCert
      SignPara.rgpMsgCert = &pCertA; // указатель на [массив] 
сертификатов, используемых для подписания
```

В данном случае параметр fDetachedSignature необходимо установить в TRUE.

При первом вызове функции (для определения размер буфера) параметр pbSignedBlob должен быть равен NULL. При втором вызове (для подписания) он должен содержать адрес буфера размером *pcbSignedBlob.

Освобождается контекст сертификата функцией **CertFreeCertificateContext**:

```
BOOL WINAPI CertFreeCertificateContext( PCCERT_CONTEXT pCertContext );
```

Функция **CertCloseStore** закрывает хранилище:

```
BOOL WINAPI CertCloseStore(
```

```
      HCERTSTORE hCertStore, // дескриптор закрываемого хранилища
      DWORD dwFlags ); // NULL
```

В случае успеха функция возвращает TRUE, в противном случае – FALSE.

Выделить память и освободить её можно так:

```
signBuffer = new BYTE[cbOutput];
delete[] signBuffer;
```

Для проверки электронной подписи, хранящейся отдельно от сообщения, используется функция **CryptVerifyDetachedMessageSignature**:

```
BOOL CryptVerifyDetachedMessageSignature(
      PCRYPT_VERIFY_MESSAGE_PARA pVerifyPara, // указатель на →
PCRYPT_VERIFY_MESSAGE_PARA (информация проверки ЭП)
      DWORD dwSignerIndex, // используется, если сообщение подписано →
несколькими пользователями
      const BYTE *pbDetachedSignBlob, // адрес буфера с подписью
      DWORD cbDetachedSignBlob, // размер *pbDetachedSignBlob
      DWORD cToBeSigned, // количество элементов в массивах →
rgpbToBeSigned и rgcbToBeSigned
      const BYTE * [] rgpbToBeSigned, // указатели на буферы, в которых →
находится то, что нужно хэшировать
      DWORD [] rgcbToBeSigned, // указатели на размеры буферов →
rgpbToBeSigned
      PCCERT_CONTEXT *ppSignerCert // контекст сертификата подписавшего
);
```

Пример вызова данной функции:

```
#define MY_ENCODING_TYPE (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)
DWORD tbsLength = 0, // длина файла
            sLength = 0; // длина подписи
//…
CRYPT_VERIFY_MESSAGE_PARA VerifyPara = {
sizeof(CRYPT_VERIFY_MESSAGE_PARA) }; // информация для проверки ЭП для →
CryptVerifyDetachedMessageSignature
VerifyPara.dwMsgAndCertEncodingType = MY_ENCODING_TYPE
BYTE *signBuffer = new BYTE[sLength]; // буфер с подписью
BYTE *buffer = new BYTE[tbsLength]; // данные, для которых нужно
проверить подпись
const BYTE *buffers = { buffer };
      if (!CryptVerifyDetachedMessageSignature( // Проверить ЭП
            &VerifyPara, // pVerifyPara указатель на →
PCRYPT_VERIFY_MESSAGE_PARA (информация проверки ЭП)
            0, // dwSignerIndex используется, если сообщение подписано →
несколькими пользователями
            signBuffer, // *pbDetachedSignBlob адрес буфера с подписью →
            sLength, // cbDetachedSignBlob размер *pbDetachedSignBlob →
            1, // cToBeSigned количество элементов в массивах →
rgpbToBeSigned и rgcbToBeSigned
            &buffers, // rgpbToBeSigned указатели на буферы, в которых →
находится то, что нужно хэшировать
```

```
            &tbsLength, // rgcbToBeSigned указатели на размеры буферов →
rgpbToBeSigned
            NULL)) // [opt] *ppSignerCert контекст сертификата →
подписавшего
      {
            cout << "invalid signature 0x" << hex << GetLastError() <<
endl;
      }
      else
      {
            cout << "signature verified" << endl;
      }
```

Для импорта сертификата используется функция **CertAddEncodedCertificateToStore**:

```
BOOL CertAddEncodedCertificateToStore(
  HCERTSTORE hCertStore, // дескриптор хранилища сертификатов
  DWORD dwCertEncodingType, // тип кодировки сертификата
  const BYTE *pbCertEncoded, // буфер, в котором находится сертификат
  DWORD cbCertEncoded, // размер *pbCertEncoded
  DWORD dwAddDisposition, // что делать, если сертификат существует
  PCCERT_CONTEXT *ppCertContext // указатель на контекст сертификата →
(для дальнейшего использования)
);
```

Функция устанавливает сертификат в формате DER.

Пример вызова:

```
      if (!(CertAddEncodedCertificateToStore( // добавить сертификат в
хранилище
            hStoreRoot, // hCertStore дескриптор хранилища сертификатов
            MY_ENCODING_TYPE, // dwCertEncodingType тип кодировки
сертификата
            buffer, // *pbCertEncoded буфер, в котором находится
сертификат
            length, // cbCertEncoded размер *pbCertEncoded
            CERT_STORE_ADD_REPLACE_EXISTING, // dwAddDisposition что
делать, если сертификат существует
            NULL // *ppCertContext указатель на контекст сертификата (для
дальнейшего использования)
      )))
      {
            cout << "CertAddEncodedCertificateToStore error" << hex <<
GetLastError() << endl;
      }
      else
      {
            cout << "the certificate is installed" << endl;
      }
```

Для выполнения задания по разработке приложения для формирования электронной подписи необходимо самостоятельно создать сертификаты. Для этого можно использовать криптографический пакет OpenSSL [11, с. 69].

Для создания сертификата центра сертификации можно использовать команду:

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

Нужно указать парольную фразу, страну, область, город, название организации и название сертификата:

```
Enter PEM pass phrase:
Verifying - Enter PEM pass phrase:
Country Name (2 letter code) [AU]:RU
State or Province Name (full name) [Some-State]:Penzenskaya obl.
Locality Name (eg, city) []:Penza
Organization Name (eg, company) [Internet Widgits Pty Ltd]:PSU
Organizational Unit Name (eg, section) []:
Common Name (e.g. server FQDN or YOUR name) []:CA_Test
Email Address []:
```

Создать запрос на сертификацию пользователя можно командой:

```
openssl.exe req -newkey rsa:2048 -keyout User_A_key.pem -out
User_A_req.pem
```

Затем запрос на сертификацию нужно подписать. Для этого сначала необходимо создать в директории `C:\Program Files (x86)\OpenSSL-Win32\bin` папку `demoCA`, а в ней – папку `newcerts`, пустой файл `index.txt` и файл `serial`, содержащий строку «01». Затем нужно ввести команду:

```
openssl.exe ca -md sha256 -keyfile ca_test_key.pem -cert ca_test_cert.pem
-in User_A_req.pem -out User_A_cert.pem
```

После этого нужно создать сертификат:

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

Созданные сертификаты будут в папке `C:\Program Files (x86)\OpenSSL-Win32\bin`. Для их импорта нужно:

а) сменить расширение файла `ca_test_cert.pem` на `crt`;

б) открыть файл и нажать «Установить сертификат…»;

в) установить сертификат для текущего пользователя в хранилище «Доверенные корневые центры сертификации»;

г) открыть файл `User_A_cert.p12`;

д) установить сертификат для текущего пользователя в хранилище «Личное».

Все пути приведены для 32-разрядной версии пакета OpenSSL, установленного с настройками по умолчанию в 64-разрядную систему. При выполнении команд рекомендуется запускать Командную строку от имени администратора.

Для проверки задания нужно сформировать сертификаты в формате DER. Для конвертирования сертификатов в этот формат можно использовать команды:

```
openssl.exe x509 -in ca_test_cert.crt -outform der -out ca_test_cert.der
openssl.exe x509 -in User_A_cert.p12 -outform der -out User_A_cert.der
```

7 Разработка приложения для проверки результатов выполнения практикума

7.1 Описание приложения для проверки результатов выполнения практикума

Для проверки результатов выполнения практикума было разработано приложение. Оно позволяет проверять правильность зашифрования, хэширования и формирования электронной подписи.

Для проверки правильности зашифрования требуется указать пути к файлам с открытым текстом, шифртекстом и ключом, а также пароль для расшифрования ключа и путь к файлу, в который будет записан расшифрованный текст. Сначала импортируется ключ, для чего выполняется генерация ещё одного ключа по паролю и расшифрование с его помощью ключа из файла, затем расшифровывается шифртекст и сравнивается полученный расшифрованный текст и открытый текст, содержащийся в файле, указанном в начале работы программы. При хэшировании пароля используется алгоритм SHA256, при расшифровании – AES.

Для проверки правильности хэширования приложение считывает путь к файлу, правильность хэширования которого требуется проверить, и хэш-код. Выполняется хэширование по алгоритму SHA256 и результат сравнивается с хэш-кодом, введённым в программу.

Для проверки правильности формирования электронной подписи необходимо указать пути к сертификатам центра сертификации и пользователя, а также пути к подписанному файлу и файлу, содержащему подпись. Для проверки сначала импортируется сертификат центра сертификации, затем сертификат пользователя (его подлинность проверяется с помощью сертификата центра сертификации). Затем с помощью указанного сертификата пользователя выполняется проверка подписи.

Диаграмма вариантов использования приложения для проверки представлена на рисунке 14.

Текст программы представлен в приложении Б.

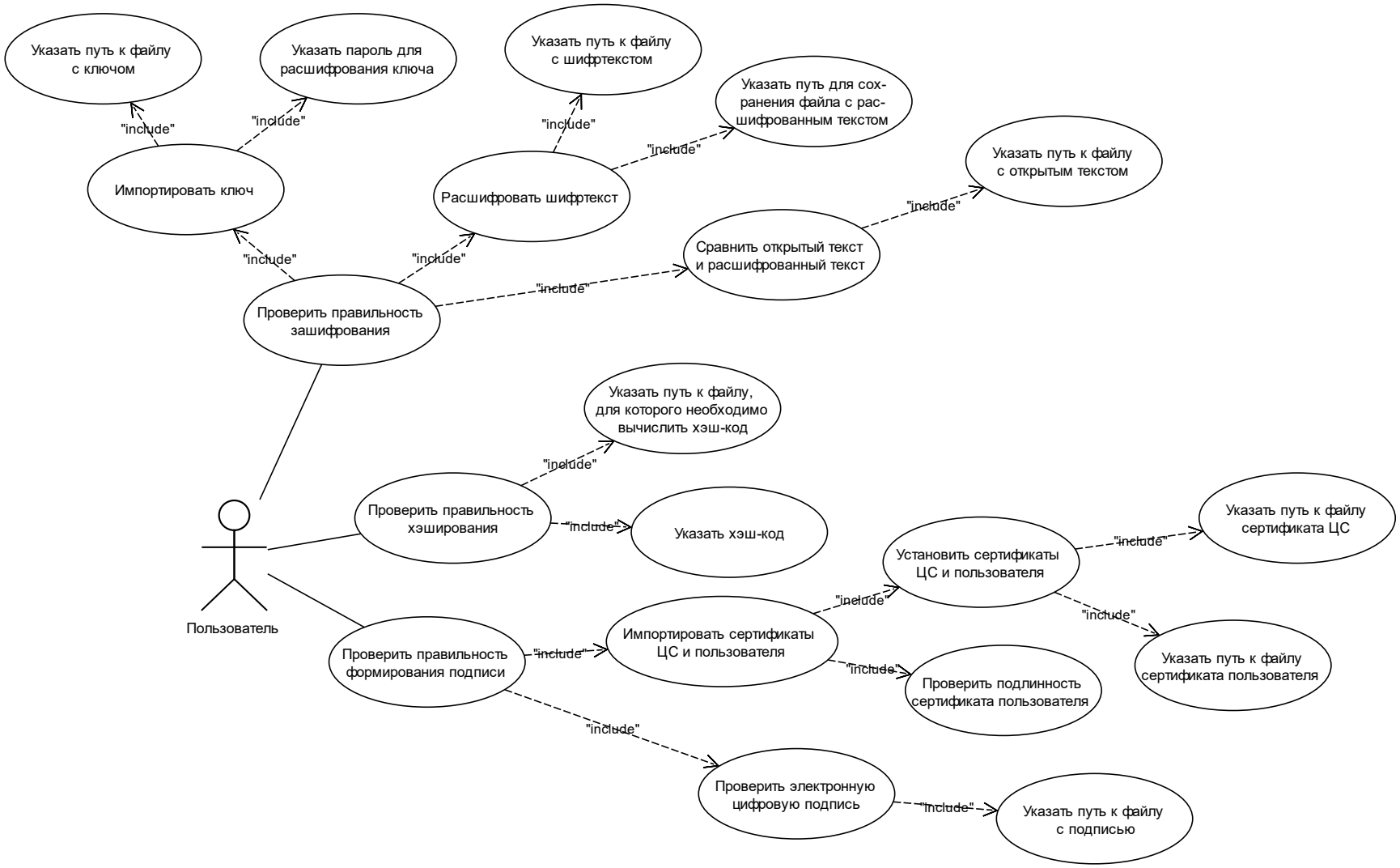

Рисунок 14 – Диаграмма вариантов использования приложения для проверки результатов

7.2 Проверка работоспособности приложения для проверки результатов выполнения практикума

7.2.1 Проверка правильности зашифрования

Для проверки работоспособности приложения для проверки результатов выполнения практикума сначала оно было запущено. Затем была выполнена проверка правильности зашифрования, осуществлённого с помощью реализованного ранее примера.

Для этого был введён путь к файлу с открытым текстом, содержащему строку:

```
один два три четыре пять шесть семь восемь девять десять
```

Была запрошена цифра, определяющая вариант использования приложения:

```
Путь к файлу с открытым текстом: 1.txt


Что необходимо проверить?
1 - зашифрование,
2 - хэширование,
3 - подпись,
0 - выйти:
```

Была выбрана проверка зашифрования путём ввода цифры 1, затем был введён путь к файлу с шифртекстом `2.txt` и путь к файлу с ключом `1.key`. Был введён пароль для расшифрования ключа и путь сохранения файла с расшифрованным текстом:

```
Путь к файлу с шифртекстом: 2.txt
Путь к файлу с ключом: 1.key
Пароль для рашифрования ключа: forExport
Путь сохранения файла с расшифрованным текстом: 4.txt
+ Зашифрование выполнено верно.
```

Было получено сообщение о том, что зашифрование выполнено верно.

Для проверки работоспособности приложения в случае неверного зашифрования в файле с шифртекстом `2.txt` последний байт `7A` был заменён на `BA`. Получившийся файл был назван `2_.txt`.

При следующем запуске приложения для проверки были введены следующие параметры:

```
Путь к файлу с открытым текстом: 1.txt


Что необходимо проверить?
1 - зашифрование,
2 - хэширование,
3 - подпись,
0 - выйти:
1
Путь к файлу с шифртекстом: 2_.txt
Путь к файлу с ключом: 1.key
Пароль для рашифрования ключа: forExport
Путь сохранения файла с расшифрованным текстом: 4.txt
decrypt error 0xc000003e- Зашифрование выполнено неверно: длины исходного открытого текста и получившегося расшифрованного текста не равны.
```

При проверке было обнаружено несоответствие длин исходного открытого текста и получившегося расшифрованного текста. Содержимое файла `4.txt`:

```
один два три четыре пять шесть семь восемь девять де
```

Для того, чтобы проверить работоспособность приложения в случае, если зашифрование было выполнено неверно, но длины исходного и расшифрованного текста совпадают, с помощью реализованного ранее примера было выполнено зашифрование файла `_1.txt`, содержащего строку:

```
десять девять восемь семь шесть пять четыре три два один
```

Получившийся шифртекст был записан в файл `_2.txt`.

Затем было запущено приложение для проверки правильности зашифрования:

```
Путь к файлу с шифртекстом: _2.txt
Путь к файлу с ключом: 1.key
Пароль для рашифрования ключа: forExport
Путь сохранения файла с расшифрованным текстом: 4.txt
- Зашифрование выполнено неверно.
```

Приложение работает правильно.

Также была проверена работоспособность приложения в случае ввода неправильного пароля для расшифрования ключа:

```
Путь к файлу с открытым текстом: 1.txt


Что необходимо проверить?
1 - зашифрование,
```

```
2 - хэширование,
3 - подпись,
0 - выйти:
1
Путь к файлу с шифртекстом: 2.txt
Путь к файлу с ключом: 1.key
Пароль для рашифрования ключа: wrongPassword
Путь сохранения файла с расшифрованным текстом: 4.txt
BCryptImportKey error 0xc000a002
Ошибка импорта ключа. Возможно, введён неверный пароль.
```

При вводе неправильного пароля на экран была выведена информация об ошибке при выполнении функции `BCryptImportKey` с кодом ошибки, а также информация о возможной причине её возникновения.

7.2.2 Проверка правильности вычисления хэш-кода

После проверки правильности зашифрования была выполнена проверка правильности хэширования. Для этого при запросе приложения пути к файлу с открытым текстом было введено имя файла `1.txt`. Затем была введена цифра 2, обозначающая, что требуется проверить хэширование. После этого был введён хэш-код, полученный ранее с помощью примера использования криптопровайдеров для вычисления хэш-кода:

```
Путь к файлу с открытым текстом: 1.txt


Что необходимо проверить?
1 - зашифрование,
2 - хэширование,
3 - подпись,
0 - выйти:
2
Хэш-код:
a9275e4a4f2ab88016ce686b9a5c4b996169adfeca4f475a6a2cdbe1d753305c
hash: a9275e4a4f2ab88016ce686b9a5c4b996169adfeca4f475a6a2cdbe1d753305c
+ Хэширование выполнено верно
```

Было выведено сообщение о том, что хэширование было выполнено верно. Затем для проверки случая, когда хэширование было бы выполнено неверно, последний символ хэш-кода был изменён:

```
Хэш-код:
a9275e4a4f2ab88016ce686b9a5c4b996169adfeca4f475a6a2cdbe1d753305e
hash: a9275e4a4f2ab88016ce686b9a5c4b996169adfeca4f475a6a2cdbe1d753305c
- Хэширование выполнено неверно
```

С помощью приложения были правильно определены случаи правильного и неправильного хэширования.

7.2.3 Проверка правильности формирования электронной подписи

Для проверки работоспособности приложения при проверке правильности формирования электронной подписи использовались файлы, полученные с помощью реализованного ранее примера использования криптопровайдеров для вычисления электронной подписи.

При соответствующих запросах приложения были введены:

– путь к файлу с открытым текстом (файл, для которого была сформирована подпись) – `1.txt`;

– путь к файлу с ЭЦП – `2.txt`;

– путь к сертификату центра сертификации – `ca_test_cert.der`;

– путь к сертификату пользователя – `User_A_cert.der`:

После этого появился запрос подтверждения удаления сертификата, показанный на рисунке 15.

Данный запрос появился из-за того, что такой сертификат уже был установлен ранее. После подтверждения удаления появился запрос подтверждения установки сертификата, показанный на рисунке 16.

После подтверждения установки появились сообщения об установке сертификатов и о том, что электронная цифровая подпись сформирована верно:

```
Путь к файлу с открытым текстом: 1.txt


Что необходимо проверить?
1 - зашифрование,
2 - хэширование,
```

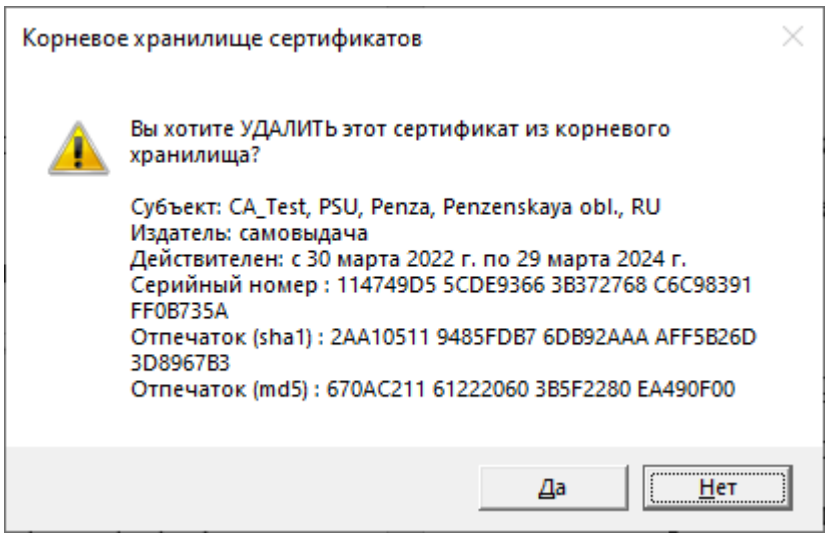

Рисунок 15 – Запрос подтверждения для удаления сертификата

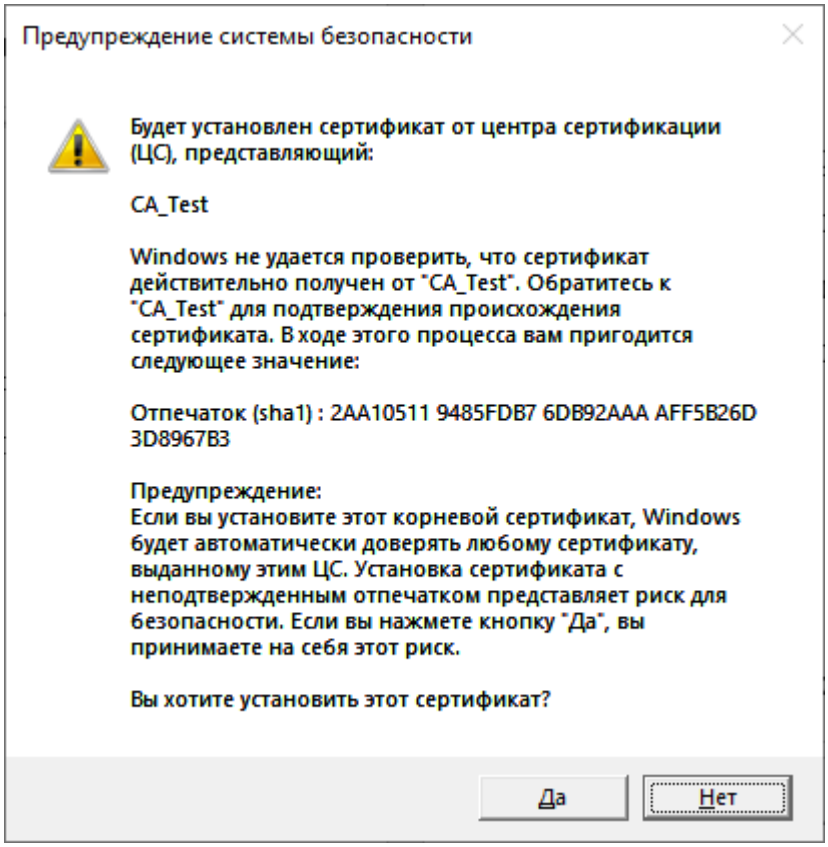

Рисунок 16 – Запрос подтверждения для установки сертификата

```
3 - подпись,
0 - выйти:
3
Путь к файлу с ЭЦП: 2.txt
Путь к сертификату центра сертификации: ca_test_cert.der
Путь к сертификату пользователя: User_A_cert.der
Сертификат центра сертификации установлен
Сертификат пользователя установлен
+ Электронная цифровая подпись сформирована верно
```

Затем был проверен случай, когда подпись была сформирована другим пользователем – подпись формировалась с помощью сертификата User_A, но при её проверке указывался путь к сертификату User_B:

```
Путь к файлу с ЭЦП: 2.txt
Путь к сертификату центра сертификации: ca_test_cert.der
Путь к сертификату пользователя: User_B_cert.der
Сертификат центра сертификации установлен
Сертификат пользователя установлен
invalid signature 80090006
- Электронная цифровая подпись сформирована неверно
```

На экран было выведено сообщение с кодом ошибки, затем результат проверки подписи. Результат соответствовал ожидаемому.

Также был проверен случай, когда подпись была сформирована для одного файла, а проверяется для другого. Подпись была сформирована для файла 1.txt с содержимым:

```
один два три четыре пять шесть семь восемь девять десять
```

А проверялась подпись для файла 1_.txt, содержащего строку:

```
десять девять восемь семь шесть пять четыре три два один
```

При проверке на экран было выведено следующее:

```
Путь к файлу с открытым текстом: 1_.txt


Что необходимо проверить?
1 - зашифрование,
2 - хэширование,
3 - подпись,
0 - выйти:
3
Путь к файлу с ЭЦП: 2.txt
Путь к сертификату центра сертификации: ca_test_cert.der
Путь к сертификату пользователя: User_A_cert.der
Сертификат центра сертификации установлен
Сертификат пользователя установлен
invalid signature 80090006
- Электронная цифровая подпись сформирована неверно
```

Таким образом, приложение для проверки результатов выполнения практикума работает верно.

8 Вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации

Профессиональная деятельность специалиста по защите информации зачастую связана с необходимостью длительного нахождения в одной позе (как правило, сидячей). Это может привести к следующим повреждениям опорно-двигательного аппарата [12]:

– уменьшению физической силы;
– потере выносливости;
– снижению плотности костей;
– искривлению позвоночника.

Также при этом могут появляться боли в позвоночнике [13].

Влияние сидячего положения на опорно-двигательный аппарат становится более выраженным при занятии неправильного положения.

Кроме того, при работе за компьютером из-за одновременного просмотра информации на мониторе и управления с помощью клавиатуры и мыши напрягаются мышцы шеи, головы, рук и плеч.

Вследствие монотонных сгибательно-разгибательных движений кисти, выполняемых в процессе набора текста на клавиатуре, возможно возникновение синдрома запястного канала – заболевания суставов кисти, приводящего к болям в пальцах и их онемению [14].

Для снижения воздействия рассмотренных факторов можно:

– сидеть правильно – например так, как показано на рисунке 17 [15];
– каждые 30 минут сидения делать перерывы – ходить, двигать плечами, приседать;
– гулять;
– делать зарядку;
– бегать;
– кататься на велосипеде.

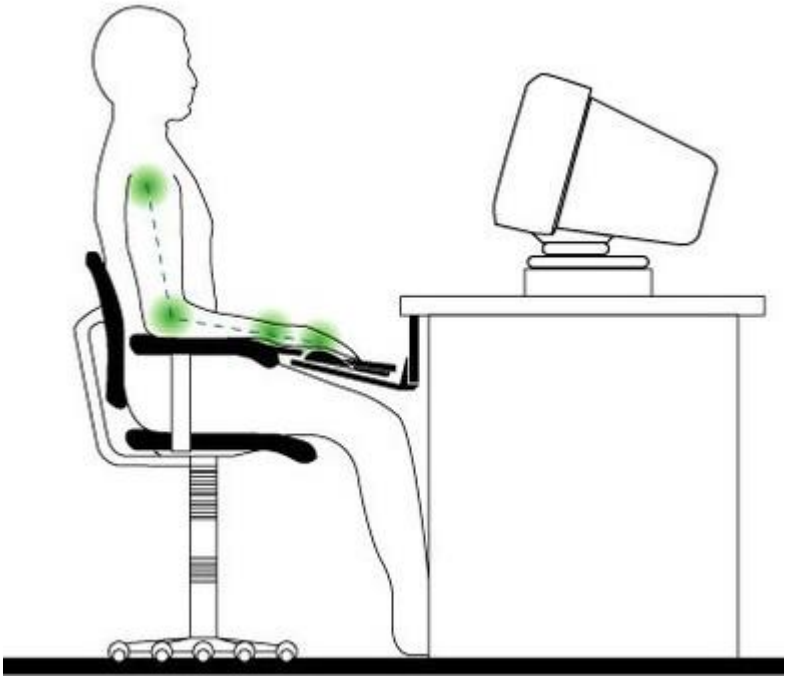

Рисунок 17 – Рекомендуемая поза для работы за компьютером

Заключение

В ходе выполнения выпускной квалификационной работы были решены следующие задачи:

- разработана структура практикума;
- разработан порядок проведения практикума;
- разработаны примеры реализации практикума;
- разработано приложение для проверки результатов выполнения практикума;
- рассмотрены вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации, а также методы и средства физической культуры, снижающие их воздействие.

Таким образом, задание на выпускную квалификационную работу выполнено в полном объеме и в отведённые для этого сроки.

Список использованных источников

1 Об утверждении Положения о лицензировании деятельности по разработке, производству, распространению шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств, выполнению работ, оказанию услуг в области шифрования информации, техническому обслуживанию шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств (за исключением случая, если техническое обслуживание шифровальных (криптографических) средств, информационных систем и телекоммуникационных систем, защищенных с использованием шифровальных (криптографических) средств, осуществляется для обеспечения собственных нужд юридического лица или индивидуального предпринимателя) [Электронный ресурс]: Постановление Правительства РФ от 16.04.2012 N 313. Доступ из справ.-правовой системы «КонсультантПлюс».

2 Уривский А. В., Чефранова А. О. К вопросу использования криптографических средств при предоставлении государственных электронных услуг // Интеллект. Инновации. Инвестиции. 2011. Спецвыпуск. URL: https://cyberleninka.ru/article/n/k-voprosu-ispolzovaniya-kriptograficheskih-sredstv-pri-predostavlenii-gosudarstvennyh-elektronnyh-uslug (дата обращения: 13.02.2022).

3 Нопин С. В., Шахов В. Г. Использование криптографических интерфейсов операционной системы Windows при разработке систем IP-телефонии, защищенных от несанкционированного доступа // ОНВ. 2006. №7 (43). URL: https://cyberleninka.ru/article/n/ispolzovanie-kriptograficheskih-interfeysov-operatsionnoy-sistemy-windows-pri-razrabotke-sistem-ip-telefonii-zaschischennyh-ot-1 (дата обращения: 13.02.2022).

4 Редикульцев Р.Н. О проблеме надежной идентификации пользователей в сетях общего пользования // Решетневские чтения. 2017. №21-2. URL: https://cyberleninka.ru/article/n/o-probleme-nadezhnoy-identifikatsii-polzovateley-v-setyah-obschego-polzovaniya (дата обращения: 13.02.2022).

5 А. П. Матковский, О. А. Немкова Защита данных в офисном пакете Microsoft Office 2007 операционной системы Windows XP // Науковий вісник НЛТУ України . 2010. №15. URL: https://cyberleninka.ru/article/n/zaschita-dannyh-v-ofisnom-pakete-microsoft-office-2007-operatsionnoy-sistemy-windows-xp (дата обращения: 13.02.2022).

6 API // Википедия. Свободная энциклопедия. Дата обновления: 18.02.2022. URL: https://ru.wikipedia.org/wiki/API (дата обращения: 20.02.2022).

7 Криптопровайдер // Википедия. Свободная энциклопедия. Дата обновления: 07.05.2020. URL: https://ru.wikipedia.org/wiki/Криптопровайдер (дата обращения: 20.02.2022).

8 Типы поставщиков служб шифрования // Microsoft Docs. Дата обновления: 25.08.2021. URL: https://docs.microsoft.com/ru-ru/windows/win32/seccrypto/cryptographic-provider-types (дата обращения: 20.02.2022).

9 Юрий Строжевский Использование Crypto API // RSDN Magazine. 2004. №5. Дата обновления: 10.12.2016. URL: http://rsdn.org/article/crypto/usingcryptoapi.xml (дата обращения: 20.02.2022).

10 CSP и процесс шифрования // Microsoft Docs. Дата обновления: 29.08.2021. URL: https://docs.microsoft.com/ru-ru/windows/win32/seccrypto/csps-and-the-cryptography-process (дата обращения: 20.02.2022);

11 Смышляев, А. Г. Методы и средства защиты информации: лабораторный практикум : учебное пособие / А. Г. Смышляев, А. В. Жуков, И. В. Жуков. – Белгород, 2016. – 102 с. URL: http://dspace.bsu.edu.ru/bitstream/123456789/16522/1/Smishliaev_Metod_Sredstva_16.pdf (дата обращения: 13.05.2022).

12 Парецкая А. «Насидеть» себе болезнь: что такое сидячий образ жизни и как он отражается на здоровье? // Санкт-Петербургское Государственное бюджетное учреждение здравоохранения «Городская поликлиника № 39». URL: http://gorpol39.spb.ru/здоровый-образ-жизни/насидеть-себе-болезнь-что-такое-сид/ (дата обращения: 02.06.2022).

13 Кисляков И. А., Беловешкин А. Г. Что происходит с организмом, если постоянно много сидеть? // Polismed.com : [сайт]. Дата обновления: 03.06.2021. URL: https://www.polismed.com/articles-chto-budet-esli-sidet.html (дата обращения: 02.06.2022).

14 Синдром запястного канала // Википедия. Свободная энциклопедия. Дата обновления: 22.04.2022. URL: https://ru.wikipedia.org/wiki/Синдром_запястного_канала (дата обращения: 02.06.2022).

15 Организация рабочего места с позиции здоровья спины (и не только) @Ariez // Хабр : [сайт]. Дата обновления: 08.12.2015. URL: https://habr.com/ru/post/369163/ (дата обращения: 02.06.2022).

16 Хорев, П. Б. Программно-аппаратная защита информации : учебное пособие / П.Б. Хорев. — 3-е изд., испр. и доп. — Москва : ИНФРА-М, 2022. — 327 с. — (Высшее образование: Бакалавриат). — DOI 10.12737/1035570. - ISBN 978-5-16-015471-8. - Текст : электронный. - URL: https://znanium.com/catalog/product/1865598 (дата обращения: 13.05.2022). – Режим доступа: по подписке.

Приложение А

(обязательное)

Примеры реализации практикума

Пример реализации приложения для зашифрования:

```
#pragma comment (lib, "bcrypt.lib") // маршрутизатор примитивов для CNG

#include <windows.h>
#include <bcrypt.h>
#include <tchar.h> // для строк
// перед константами ставить префикс “L”(или использовать макрос TEXT).
#include <iostream>
#include <fstream>
#include <string>

using namespace std;

int main()
{
	NTSTATUS status = 0; // инициализация кода ошибки, возвращаемого функциями CNG
	BCRYPT_ALG_HANDLE hAlg; // дескриптор провайдера алгоритма
	DWORD cbHashObject = 0, cbData = 0; //
	PBYTE pbHashObject,
		pbHashObjectExp = NULL;                    // буфер для хэш-объекта
	BCRYPT_HASH_HANDLE hHash,
		hHashExp; // дескриптор хэш-объекта
	TCHAR szPass[] = TEXT("password"); // хэшируемый пароль
	TCHAR szPassExp[] = TEXT("forExport"); // хэшируемый пароль для ключа зашифрования
ключа
	PBYTE pbHash,
		pbHashExp = NULL; // буфер для принятия хэш-кода
	DWORD cbHash = 0; // длина буфера для хэша
	PBYTE pbKeyObject = NULL,
		pbKeyObjectExp = NULL; // буфер для ключевого объекта
	DWORD cbKeyObject = 0; // длина буфера для ключевого объекта
	BCRYPT_KEY_HANDLE hKey,
		hKeyExp; // дескриптор ключевого объекта
	BYTE Plaintext[] = { 1, 2, 3, 4, 5, 6, 7, 8, 9, 10 }; // открытый текст
	DWORD cbOutput = 0, // размер буфера для шифртекста
		cbCiphertext = 0; // размер получившегося шифртекста
	PBYTE pbOutput = NULL; // буфер с шифртекстом
	DWORD cbDecryptedText, // размер буфера для расшированного текста
		cDecryptedText; // размер получившегося расшифрованного текста
	ULONG cbOutputExp = 0; // экспортировано байт
	PUCHAR expKeyBuf = NULL; // буфер для экспорта ключа


	DWORD  // длина буфера
		tail = 16; //
	BYTE *buffer;
	string plainTextFilePath; // путь к файлу с открытым текстом
	string cipherTextFilePath; // путь к файлу для записи шифртекста
	string decryptedTextFilePath, // путь к файлу для записи расшифрованного текста
		keyExpPath; // путь для экспорта ключа
	int length = 0; // длина файла

	cout << "plaintext: ";
```

```
	cin >> plainTextFilePath;
	cout << "ciphertext: ";
	cin >> cipherTextFilePath;
	cout << "decrypted: ";
	cin >> decryptedTextFilePath;
	cout << "export key to: ";
	cin >> keyExpPath;
	ifstream iF(plainTextFilePath, ios_base::in | ios_base::binary);
	ofstream oF(cipherTextFilePath, ios_base::out | ios_base::binary);
	ofstream oFkey(keyExpPath, ios_base::out | ios_base::binary);



	// загрузка и инициализация провайдера конкретного алгоритма
	if (!BCRYPT_SUCCESS(status = BCryptOpenAlgorithmProvider(
		&hAlg, // *phAlgorithm сюда запишется дескриптор провайдера алгоритма
		BCRYPT_SHA256_ALGORITHM, // pszAlgId идентификатор (имя) алгоритма (указатель)
		NULL, // pszImplementation имя провайдера криптографических примитивов (для
BCRYPT_AES_ALGORITHM // по умолчанию будет Microsoft Primitive Provider
		0))) // dwFlags 0 для обычного шифрования/хэширования
	{
		cout << "BCryptOpenAlgorithmProvider BCRYPT_SHA256_ALGORITHM error 0x" << hex
<< status;
	}


	// получение буфера для хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта,
значение свойства которого необходимо получить
		BCRYPT_OBJECT_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbHashObject, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных,
реально скопированных в pbOutput
		0))) // dwFlags 0
	{
		cout << "hash object buffer creation error 0x" << hex << status;
	}
	// выделение памяти для объекта
	pbHashObject = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHashObject);
	pbHashObjectExp = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHashObject); // объект для
создания ключа для зашифрования экспортируемого ключа

	// создание хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptCreateHash(hAlg, // hAlgorithm дескриптор
алгоритма хэширования,

	// полученный с помощью BCryptOpenAlgorithmProvider
		&hHash, // *phHash указатель на дескриптор создаваемого хэш-объекта
		pbHashObject, // pbHashObject указатель на буфер, в котором размещается
создаваемый хэш - объект
		cbHashObject, // cbHashObject размер буфера pbHashObject
		NULL, // pbSecret адрес буфера, содержащего ключ, используемый для создания
кода аутентичности сообщения
		0, // cbSecret размер pbSecret
		0))) // dwFlags 0
	{
		cout << "BCryptCreateHash error 0x" << hex << status;
	}

	// создание хэш-объекта для второго ключа
	if (!BCRYPT_SUCCESS(status = BCryptCreateHash(hAlg, // hAlgorithm дескриптор
алгоритма хэширования,
```

```
	// полученный с помощью BCryptOpenAlgorithmProvider
		&hHashExp, // *phHash указатель на дескриптор создаваемого хэш-объекта
		pbHashObjectExp, // pbHashObject указатель на буфер, в котором размещается создаваемый хэш - объект
		cbHashObject, // cbHashObject размер буфера pbHashObject
		NULL, // pbSecret адрес буфера, содержащего ключ, используемый для создания кода аутентичности сообщения
		0, // cbSecret размер pbSecret
		0))) // dwFlags 0
	{
		cout << "BCryptCreateHash2 error 0x" << hex << status;
	}


	// хэширование данных
	if (!BCRYPT_SUCCESS(status = BCryptHashData(hHash, // hHash дескриптор хэш-объекта
		(PBYTE)szPass, // pbInput адрес буфера с входными данными
		_tcslen(szPass) * sizeof(TCHAR), // cbInput размер pbInput
		0))) // dwFlags
	{
		cout << "BCryptHashData error 0x" << hex << status;
	}


	// хэширование данных для второго ключа
	if (!BCRYPT_SUCCESS(status = BCryptHashData(hHashExp, // hHash дескриптор хэш-объекта
		(PBYTE)szPassExp, // pbInput адрес буфера с входными данными
		_tcslen(szPassExp) * sizeof(TCHAR), // cbInput размер pbInput
		0))) // dwFlags
	{
		cout << "BCryptHashData2 error 0x" << hex << status;
	}

	// извлечение хэш-кода из хэш-объекта
	// создание буфера для хэш-кода
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта, значение свойства которого необходимо получить
		BCRYPT_HASH_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbHash, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных, реально скопированных в pbOutput
		0))) // dwFlags 0
	{
		cout << "hash code buffer creation error 0x" << hex << status;
	}


	// выделение памяти для объекта (хэша)
	pbHash = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHash);
	pbHashExp = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHash); // для второго ключа


	// извлчение хэш-кода из хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptFinishHash(hHash, // hHash дескриптор хэш-объекта
		pbHash, // pbOutput буфер для принятия хэш-кода
		cbHash, // cbOutput размер буфера pbOutput
		0))) // dwFlags
	{
		cout << "BCryptFinishHash error 0x" << hex << status;
	}
```

```
	// извлчение хэш-кода из хэш-объекта для второго ключа
	if (!BCRYPT_SUCCESS(status = BCryptFinishHash(hHashExp, // hHash дескриптор хэш-
объекта
		pbHashExp, // pbOutput буфер для принятия хэш-кода
		cbHash, // cbOutput размер буфера pbOutput
		0))) // dwFlags
	{
		cout << "BCryptFinishHash error 0x" << hex << status;
	}

	// загрузка и инициализация провайдера конкретного алгоритма (шифрования)
	if (!BCRYPT_SUCCESS(status = BCryptOpenAlgorithmProvider(
		&hAlg, // *phAlgorithm сюда запишется дескриптор провайдера алгоритма
		BCRYPT_AES_ALGORITHM, // pszAlgId идентификатор (имя) алгоритма (указатель)
		NULL, // pszImplementation имя провайдера криптографических примитивов (для
BCRYPT_AES_ALGORITHM // по умолчанию будет Microsoft Primitive Provider
		0))) // dwFlags 0 для обычного шифрования/хэширования
	{
		cout << "BCryptOpenAlgorithmProvider BCRYPT_AES_ALGORITHM error 0x" << hex <<
status;
	}

	// получение буфера
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта,
значение свойства которого необходимо получить
		BCRYPT_OBJECT_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbKeyObject, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных,
реально скопированных в pbOutput
		0))) // dwFlags 0
	{
		cout << "key object buffer creation error 0x" << hex << status;
	}
	// выделение памяти для объекта
	pbKeyObject = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbKeyObject);
	pbKeyObjectExp = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbKeyObject); // для второго
ключа



	// Создание ключевого объекта
	// и помещение в него ключа симметричного шифрования загруженного ранее алгоритма
	if (!BCRYPT_SUCCESS(status = BCryptGenerateSymmetricKey(hAlg, // hAlgorithm
полученный с помощью BCryptOpenAlgorithmProvider дескриптор алгоритма шифрования
		&hKey, // *phKey дескриптор создаваемого ключевого объекта
		pbKeyObject, // pbKeyObject указатель на буфер, в котором размещается
созданный ключевой объект
		cbKeyObject, // cbKeyObject размер буфера pbKeyObject
		(PBYTE)pbHash, // pbSecret указатель на буфер, который содержит ключевой
материал
		16, // cbSecret размер буфера bpSecret
		0))) // dwFlags
	{
		cout << "BCryptGenerateSymmetricKey error 0x" << hex << status;
	}

	// Создание ключевого объекта
// и помещение в него ключа симметричного шифрования загруженного ранее алгоритма
	if (!BCRYPT_SUCCESS(status = BCryptGenerateSymmetricKey(hAlg, // hAlgorithm
полученный с помощью BCryptOpenAlgorithmProvider дескриптор алгоритма шифрования
		&hKeyExp, // *phKey дескриптор создаваемого ключевого объекта
		pbKeyObjectExp, // pbKeyObject указатель на буфер, в котором размещается
созданный ключевой объект
```

```
		cbKeyObject, // cbKeyObject размер буфера pbKeyObject
		(PBYTE)pbHashExp, // pbSecret указатель на буфер, который содержит ключевой
материал
		16, // cbSecret размер буфера bpSecret
		0))) // dwFlags
	{
		cout << "BCryptGenerateSymmetricKey error 0x" << hex << status;
	}

	iF.seekg(0, iF.end); //
	length = iF.tellg(); // определение длины файла
	iF.seekg(0, iF.beg); //

	buffer = new BYTE[length + tail];

	iF.read((char*)buffer, length); // прочитать файл в буфер

		// зашифрование (определение размера буфера)
	if (!BCRYPT_SUCCESS(status = BCryptEncrypt(hKey, // hKey дескриптор созданного ранее
	ключа
		buffer, // pbInput адрес буфера с открытым текстом
		length, // cbInput размер буфера с открытым текстом
		NULL, // *pPaddingInfo указатель на структуру, определяющую параметры
дополнения блоков
		NULL, // pbIV адрес буфера, содержащего вектор инициализации
		0, // cbIV размер вектора инициализации
		NULL, // pbOutput адрес буфера, в который будет помещаться шифртекст
		0, // cbOutput размер буфера для шифртекста
		&cbOutput, // *pcbResult размер получившегося шифртекста
		BCRYPT_BLOCK_PADDING))) // dwFlags
	{
		cout << "BCryptEncrypt buffer size define error 0x" << hex << status;
	}


		// зашифрование
	if (!BCRYPT_SUCCESS(status = BCryptEncrypt(hKey, // hKey дескриптор созданного ранее
	ключа
		buffer, // pbInput адрес буфера с открытым текстом
		length, // cbInput размер буфера с открытым текстом
		NULL, // *pPaddingInfo указатель на структуру, определяющую параметры
дополнения блоков
		NULL, // pbIV адрес буфера, содержащего вектор инициализации
		0, // cbIV размер вектора инициализации
		buffer, // pbOutput адрес буфера, в который будет помещаться шифртекст
		cbOutput, // cbOutput размер буфера для шифртекста
		&cbCiphertext, // *pcbResult размер получившегося шифртекста
		BCRYPT_BLOCK_PADDING))) // dwFlags
	{
		cout << "encrypt error 0x" << hex << status;
	}


	oF.write((char*)buffer, cbCiphertext); // записать зашифрованные данные в файл
	oF.close();
	iF.close();


	ifstream iF2(cipherTextFilePath, ios_base::in | ios_base::binary);
	ofstream oF2(decryptedTextFilePath, ios_base::out | ios_base::binary);

	iF2.seekg(0, iF2.end); //
	length = iF2.tellg(); // определение длины файла
	iF2.seekg(0, iF2.beg); //
```

```
	buffer = new BYTE[length];

	iF2.read((char*)buffer, length); // прочитать файл в буфер


		// расшифрование (определение размера буфера)
	if (!BCRYPT_SUCCESS(status = BCryptDecrypt(hKey, // hKey дескриптор созданного ранее
	ключа
		buffer, // pbInput адрес буфера с шифртекстом
		length, // cbInput размер буфера с шифртекстом
		NULL, // *pPaddingInfo указатель на структуру, определяющую параметры
дополнения блоков
		NULL, // pbIV адрес буфера, содержащего вектор инициализации
		0, // cbIV размер вектора инициализации
		NULL, // pbOutput адрес буфера, в который будет помещаться расшифрованный
текст
		0, // cbOutput размер буфера для расшифрованного текста
		&cbDecryptedText, // *pcbResult размер получившегося расшифрованного текста
		BCRYPT_BLOCK_PADDING))) // dwFlags
	{
		cout << "BCryptDecrypt buffer size define error 0x" << hex << status;
	}


		// расшифрование
	if (!BCRYPT_SUCCESS(status = BCryptDecrypt(hKey, // hKey дескриптор созданного ранее
	ключа
		buffer, // pbInput адрес буфера с шифртекстом
		length, // cbInput размер буфера с шифртекстом
		NULL, // *pPaddingInfo указатель на структуру, определяющую параметры
дополнения блоков
		NULL, // pbIV адрес буфера, содержащего вектор инициализации
		0, // cbIV размер вектора инициализации
		buffer, // pbOutput адрес буфера, в который будет помещаться расшифрованный
текст
		cbDecryptedText, // cbOutput размер буфера для расшифрованного текста
		&cDecryptedText, // *pcbResult размер получившегося расшифрованного текста
		BCRYPT_BLOCK_PADDING))) // dwFlags
	{
		cout << "decrypt error 0x" << hex << status;
	}



	oF2.write((char*)buffer, cDecryptedText); // записать зашифрованные данные в файл
	oF2.close();
	iF2.close();

	// экспорт ключа 1 (зашифрование ключом 2)
	if (!BCRYPT_SUCCESS(status = BCryptExportKey( // экспорт ключа (определение размера)
		hKey, // hKey дескриптор экспортируемого ключа
		hKeyExp, // hExportKey дескриптор ключа, используемого для зашифрования hKey
		BCRYPT_AES_WRAP_KEY_BLOB, // pszBlobType тип BLOB-объекта,
		NULL, // pbOutput буфер для результата
		NULL, // cbOutput размер pbOutput
		&cbOutputExp, // *pcbResult количество скопированных байт
		NULL))) // dwFlags
	{
		cout << "BCryptExportKey define buffer size error 0x" << hex << status <<
endl;
	}

	expKeyBuf = new UCHAR[cbOutputExp];
```

```
	if (!BCRYPT_SUCCESS(status = BCryptExportKey( // экспорт ключа
		hKey, // hKey дескриптор экспортируемого ключа
		hKeyExp, // hExportKey дескриптор ключа, используемого для зашифрования hKey
		BCRYPT_AES_WRAP_KEY_BLOB, // pszBlobType тип BLOB-объекта,
		expKeyBuf, // pbOutput буфер для результата
		cbOutputExp, // cbOutput размер pbOutput
		&cbOutputExp, // *pcbResult количество скопированных байт
		NULL))) // dwFlags
	{
		cout << "BCryptExportKey error 0x" << hex << status << endl;
	}

	oFkey.write((char*)expKeyBuf, cbOutputExp); // записать зашифрованные данные в файл
	oFkey.close();

	delete[] buffer;
	delete[] expKeyBuf;


	if (!BCRYPT_SUCCESS(status = BCryptDestroyHash(hHash))) // уничтожение хэш-объекта
	{
		cout << "BCryptDestroyHash error 0x" << hex << status;
	}
	if (!BCRYPT_SUCCESS(status = BCryptDestroyHash(hHashExp))) // уничтожение хэш-объекта
	{
		cout << "BCryptDestroyHash2 error 0x" << hex << status;
	}

	if (!BCRYPT_SUCCESS(status = BCryptDestroyKey(hKey))) // уничтожение ключевого
объекта
	{
		cout << "BCryptDestroyKey error 0x" << hex << status;
	}
	if (!BCRYPT_SUCCESS(status = BCryptDestroyKey(hKeyExp))) // уничтожение ключевого
объекта
	{
		cout << "BCryptDestroyKey2 error 0x" << hex << status;
	}

	if (!BCRYPT_SUCCESS(status = BCryptCloseAlgorithmProvider( // выгрузка провайдера
алгоритма
		hAlg, // hAlgorithm дескриптор провайдера
		0))) // dwFlags
	{
		cout << "BCryptCloseAlgorithmProvider error 0x" << hex << status;
	}




	//освобождение объекта
	HeapFree(GetProcessHeap(), 0, pbHashObject); // освобождение буфера хэш-объекта
	HeapFree(GetProcessHeap(), 0, pbHashObjectExp); // освобождение буфера хэш-объекта

		//освобождение объекта
	HeapFree(GetProcessHeap(), 0, pbHash); // освобождение буфера хэша
	HeapFree(GetProcessHeap(), 0, pbHashExp); // освобождение буфера хэша

	HeapFree(GetProcessHeap(), 0, pbKeyObject); // освобождение буфера ключевого объекта
	HeapFree(GetProcessHeap(), 0, pbKeyObjectExp); // освобождение буфера ключевого
объекта
```

```
	return 0;
	}
```

Пример реализации приложения для хэширования:

```
#pragma comment (lib, "bcrypt.lib") // маршрутизатор примитивов для CNG

#include <windows.h>
#include <bcrypt.h>
#include <tchar.h> // для строк
// перед константами ставить префикс “L”(или использовать макрос TEXT).
#include <iostream>
#include <fstream>
#include <string>
#include <sstream>

using namespace std;

int main()
{
	NTSTATUS status = 0; // инициализация кода ошибки, возвращаемого функциями CNG
	BCRYPT_ALG_HANDLE hAlg; // дескриптор провайдера алгоритма
	DWORD cbHashObject = 0, cbData = 0; //
	PBYTE pbHashObject = NULL;		// буфер для хэш-объекта
	BCRYPT_HASH_HANDLE hHash; // дескриптор хэш-объекта
	PBYTE pbHash = NULL; // буфер для принятия хэш-кода
	DWORD cbHash = 0; // длина буфера для хэша


	BYTE *buffer;
	string filePath; // путь к файлу, для которого будет вычисляться хэш
	int length = 0; // длина файла
	string hash; // хэш-код, вводимый пользователем
	ostringstream oStrHash; // вычисленный хэш

	cout << "file: ";
	cin >> filePath;

	ifstream iF(filePath, ios_base::in | ios_base::binary);



	// загрузка и инициализация провайдера конкретного алгоритма
	if (!BCRYPT_SUCCESS(status = BCryptOpenAlgorithmProvider(
		&hAlg, // *phAlgorithm сюда запишется дескриптор провайдера алгоритма
		BCRYPT_SHA256_ALGORITHM, // pszAlgId идентификатор (имя) алгоритма (указатель)
		NULL, // pszImplementation имя провайдера криптографических примитивов (для
BCRYPT_AES_ALGORITHM // по умолчанию будет Microsoft Primitive Provider
		0))) // dwFlags 0 для обычного шифрования/хэширования
	{
		cout << "BCryptOpenAlgorithmProvider BCRYPT_SHA256_ALGORITHM error 0x" << hex
<< status;
	}


	// получение буфера для хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта,
значение свойства которого необходимо получить
		BCRYPT_OBJECT_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbHashObject, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных,
реально скопированных в pbOutput
```

```
		0))) // dwFlags 0
	{
		cout << "hash object buffer creation error 0x" << hex << status;
	}
	// выделение памяти для объекта
	pbHashObject = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHashObject);

	// создание хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptCreateHash(hAlg, // hAlgorithm дескриптор
алгоритма хэширования,

	// полученный с помощью BCryptOpenAlgorithmProvider
		&hHash, // *phHash указатель на дескриптор создаваемого хэш-объекта
		pbHashObject, // pbHashObject указатель на буфер, в котором размещается
создаваемый хэш - объект
		cbHashObject, // cbHashObject размер буфера pbHashObject
		NULL, // pbSecret адрес буфера, содержащего ключ, используемый для создания
кода аутентичности сообщения
		0, // cbSecret размер pbSecret
		0))) // dwFlags 0
	{
		cout << "BCryptCreateHash error 0x" << hex << status;
	}

	iF.seekg(0, iF.end); //
	length = iF.tellg(); // определение длины файла
	iF.seekg(0, iF.beg); //

	buffer = new BYTE[length];

	iF.read((char*)buffer, length); // прочитать файл в буфер

	// хэширование данных
	if (!BCRYPT_SUCCESS(status = BCryptHashData(hHash, // hHash дескриптор хэш-объекта
		buffer, // pbInput адрес буфера с входными данными
		length, // cbInput размер pbInput
		0))) // dwFlags
	{
		cout << "BCryptHashData error 0x" << hex << status;
	}

	// извлечение хэш-кода из хэш-объекта
	// создание буфера для хэш-кода
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта,
значение свойства которого необходимо получить
		BCRYPT_HASH_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbHash, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных,
реально скопированных в pbOutput
		0))) // dwFlags 0
	{
		cout << "hash code buffer creation error 0x" << hex << status;
	}
	// выделение памяти для объекта (хэша)
	pbHash = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHash);
	// извлечение хэш-кода из хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptFinishHash(hHash, // hHash дескриптор хэш-объекта
		pbHash, // pbOutput буфер для принятия хэш-кода
		cbHash, // cbOutput размер буфера pbOutput
		0))) // dwFlags
	{
		cout << "BCryptFinishHash error 0x" << hex << status;
	}
```

```
	//вывод хэш-кода на экран
	cout << "hash: ";
	for (int i = 0; i < cbHash; i++)
	{
		oStrHash << hex << (int)pbHash[i];
	}
	cout << oStrHash.str() << endl;

	cout << "verification hash: ";
	cin >> hash;
	if (oStrHash.str() == hash)
	{
		cout << "==" << endl;
	}
	else
	{
		cout << "!=" << endl;
	}


	iF.close();

	delete[] buffer;


	if (!BCRYPT_SUCCESS(status = BCryptDestroyHash(hHash))) // уничтожение хэш-объекта
	{
		cout << "BCryptDestroyHash error 0x" << hex << status;
	}


	if (!BCRYPT_SUCCESS(status = BCryptCloseAlgorithmProvider( // выгрузка провайдера
алгоритма
		hAlg, // hAlgorithm дескриптор провайдера
		0))) // dwFlags
	{
		cout << "BCryptCloseAlgorithmProvider error 0x" << hex << status;
	}

	//освобождение объекта
	HeapFree(GetProcessHeap(), 0, pbHashObject); // освобождение буфера хэш-объекта

		//освобождение объекта
	HeapFree(GetProcessHeap(), 0, pbHash); // освобождение буфера хэша




	return 0;
	}
```

Пример реализации приложения для формирования электронной подписи:

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

#include <windows.h>
#include <wincrypt.h>
#include <iostream>
#include <fstream>
#include <string>

#define MY_ENCODING_TYPE (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)
```

```
using namespace std;

int main()
{


	HCERTSTORE hStoreMy = NULL,
		hStoreRoot = NULL; // хранилище
	PCCERT_CONTEXT pCert = NULL; // указатель на предыдущий сертификат для
CertEnumCertificatesInStore
	TCHAR szNameString[128]; // имя для CertGetNameString
	TCHAR szCertNameA[] = TEXT("User_A"), // имя сертификата A
		szCertNameB[] = TEXT("User_B"); // имя сертификата B
	PCCERT_CONTEXT pCertA = NULL, // контекст сертификата A
		pCertB = NULL;  // контекст сертификата B
	DWORD signatureAndTimeCheck = CERT_STORE_SIGNATURE_FLAG |
CERT_STORE_TIME_VALIDITY_FLAG;
	CRYPT_SIGN_MESSAGE_PARA SignPara = { sizeof(CRYPT_SIGN_MESSAGE_PARA) }; // параметры
электронной подписи для CryptSignAndEncryptMessage
	CRYPT_VERIFY_MESSAGE_PARA VerifyPara = { sizeof(CRYPT_VERIFY_MESSAGE_PARA) }; //
информация для проверки ЭП для CryptDecryptAndVerifyMessageSignature


	BYTE *buffer; // данные

	BYTE *signBuffer = NULL; // подпись
	string file; // путь к файлу с открытым текстом
	string sign; // путь к файлу для записи шифртекста
	string signIn; // путь к файлу с подписью для проверки
	//string decryptedTextFilePath; // путь к файлу для записи расшифрованного текста
	DWORD tbsLength = 0, // длина файла
		sLength = 0; // длина подписи
	DWORD cbOutput = 0, // размер буфера для результата sign
		tail = 2000;

	cout << "file: ";
	cin >> file;
	cout << "sign out: ";
	cin >> sign;
	cout << "sign  in: ";
	cin >> signIn;
	ifstream iFtoBeSigned(file, ios_base::in | ios_base::binary);
	ofstream oFsign(sign, ios_base::out | ios_base::binary);

	if (!(hStoreMy = CertOpenSystemStore( // открыть системное хранилище
		NULL, // hprov (не используется)
		TEXT("MY")))) // szSubsystemProtocol имя системного хранилища сертификатов
					// MY - Личное
					// ROOT - Доверенные корневые центры сертификации
	{
		cout << "CertOpenSystemStore MY Error" << endl;
	}

	while (pCert = CertEnumCertificatesInStore( // перечисление сертификатов
		hStoreMy, // hCertStore дескриптор хранилища
		pCert)) // pPrevCertContext указатель на предыдущий сертификат
	{
		//Обработка извлеченного контекста сертификата
		if (CertGetNameString( // получить имя
			pCert, // pCertContext контекст сертификата
			CERT_NAME_SIMPLE_DISPLAY_TYPE, // dwType какую часть имени возвращать
(CN - общепринятое имя)
			0, // dwFlags (0 для имени субъекта)
			NULL, // *pvTypePara
```

```
                    szNameString, // pszNameString куда записать полученную строку
                    128)  // cchNameString количество символов, которое может вместить
pszNameString
                    > 1) // функция возвращает количество полученных символов или 1, если
требуемого имени нет
             {
                    //Обработка строки szNameString
                    cout << szNameString << endl;
             }
      }



      if (!(pCertA = CertFindCertificateInStore( // найти сертификат
             hStoreMy,  // hCertStore дескриптор открытого хранилища сертификатов
             MY_ENCODING_TYPE, // dwCertEncodingType
             0, // dwFindFlags
             CERT_FIND_SUBJECT_STR_A, // dwFindType тип поиска (поиск по имени)
             szCertNameA, // *pvFindPara указатель на строку с именем искомого сертификата
(если CERT_FIND_SUBJECT_STR)
             NULL))) // pPrevCertContext указатель на контекст предыдущего сертификата
      {
             cout << "CertFindCertificateInStore User_A error" << endl;
      }


      if (!(pCertB = CertFindCertificateInStore( // найти сертификат
             hStoreMy,  // hCertStore дескриптор открытого хранилища сертификатов
             MY_ENCODING_TYPE, // dwCertEncodingType
             0, // dwFindFlags
             CERT_FIND_SUBJECT_STR_A, // dwFindType тип поиска (поиск по имени)
             szCertNameB, // *pvFindPara указатель на строку с именем искомого сертификата
(если CERT_FIND_SUBJECT_STR)
             NULL))) // pPrevCertContext указатель на контекст предыдущего сертификата
      {
             cout << "CertFindCertificateInStore User_B error" << endl;
      }

      if (!(hStoreRoot = CertOpenSystemStore( // открыть системное хранилище
             NULL, // hprov (не используется)
             TEXT("ROOT")))) // szSubsystemProtocol имя системного хранилища сертификатов
                                          // MY - Личное
                                          // ROOT - Доверенные корневые центры сертификации
      {
             cout << "CertOpenSystemStore ROOT Error" << endl;
      }


      if (!(CertGetIssuerCertificateFromStore( // получить сертфификат ЦС и проверить
целостность сертификата пользователя A
             hStoreRoot, // hCertStore дескриптор открытого хранилища, в котором ищется
сертификат ЦС
             pCertA, // pSubjectContext контекст пользовательского сертификата, для
которого ищется сертификат ЦС
             NULL, // pPrevIssuerContext контекст предыдущего найденного сертификата этого
ЦС
             &signatureAndTimeCheck))) // *pdwFlags характер проверки пользовательского
сертификата
      {
             cout << "CertGetIssuerCertificateFromStore error";
      }

      if (signatureAndTimeCheck)
      {
```

```
		cout << "cert check error: wrong signature or time" << endl;
	}

	if (!(CertGetIssuerCertificateFromStore( // получить сертфификат ЦС и проверить целостность сертификата пользователя B
		hStoreRoot, // hCertStore дескриптор открытого хранилища, в котором ищется сертификат ЦС
		pCertB, // pSubjectContext контекст пользовательского сертификата, для которого ищется сертификат ЦС
		NULL, // pPrevIssuerContext контекст предыдущего найденного сертификата этого ЦС
		&signatureAndTimeCheck))) // *pdwFlags характер проверки пользовательского сертификата
	{
		cout << "CertGetIssuerCertificateFromStore error";
	}

	if (signatureAndTimeCheck)
	{
		cout << "cert check error: wrong signature or time" << endl;
	}


	// параметры электронной подписи (структура)
	SignPara.dwMsgEncodingType = MY_ENCODING_TYPE;
	SignPara.pSigningCert = pCertA; // сертификат
	SignPara.HashAlgorithm.pszObjId = (LPSTR)CertAlgIdToOID(CALG_MD5)/*const_cast<char*>(szOID_RSA_SHA256RSA)*/; // алгоритм для хэширования
	SignPara.cMsgCert = 1; // количество элементов в rgpMsgCert
	SignPara.rgpMsgCert = &pCertA; // указатель на [массив] сертификатов, используемых для подписания

	iFtoBeSigned.seekg(0, iFtoBeSigned.end); //
	tbsLength = iFtoBeSigned.tellg();		// определение длины файла
	iFtoBeSigned.seekg(0, iFtoBeSigned.beg); //

	buffer = new BYTE[tbsLength + tail];
	const BYTE *buffers = { buffer };

	iFtoBeSigned.read((char*)buffer, tbsLength); // прочитать файл в буфер


	if (!CryptSignMessage( // подписать сообщение (определение размера буфера)
		&SignPara, // [in] pSignPara параметры электронной подписи
		TRUE, // [in] fDetachedSignature в результате будет подпись [только (true)] [и сообщение (false)]
		1, //cToBeSigned количество элементов в rgpbToBeSigned[] и rgcbToBeSigned[]
		&buffers, // rgpbToBeSigned [массив] указателей на буферы, содержащие то, что нужно подписать
		&tbsLength, // rgcbToBeSigned [массив] размеров буферов, указанных в rgpbToBeSigned
		NULL, // *pbSignedBlob буфер для результата
		&cbOutput)) // *pcbSignedBlob размер *pbSignedBlob
	{
		cout << "CryptSignMessage buffer size define error 0x" << hex << GetLastError();
	}


	signBuffer = new BYTE[cbOutput];

	if (!CryptSignMessage( // подписать сообщение
		&SignPara, // [in] pSignPara параметры электронной подписи
```

```
		TRUE, // [in] fDetachedSignature в результате будет подпись [только (true)] [и
сообщение (false)]
		1, //cToBeSigned количество элементов в rgpbToBeSigned[] и rgcbToBeSigned[]
		&buffers, // rgpbToBeSigned [массив] указателей на буферы, содержащие то, что
нужно подписать
		&tbsLength, // rgcbToBeSigned [массив] размеров буферов, указанных в
rgpbToBeSigned
		signBuffer, // *pbSignedBlob буфер для результата
		&cbOutput)) // *pcbSignedBlob размер *pbSignedBlob
	{
		cout << "CryptSignMessage error 0x" << hex << GetLastError();
	}


	oFsign.write((char*)signBuffer, cbOutput); // записать подпись в файл
	oFsign.close();
	iFtoBeSigned.close();


	// информация для проверки ЭП (структура)
	VerifyPara.dwMsgAndCertEncodingType = MY_ENCODING_TYPE;

	ifstream iFsign(signIn, ios_base::in | ios_base::binary);

	iFsign.seekg(0, iFsign.end); //
	sLength = iFsign.tellg();  // определение длины файла
	iFsign.seekg(0, iFsign.beg); //

	delete[] signBuffer;
	signBuffer = new BYTE[sLength];

	iFsign.read((char*)signBuffer, sLength); // прочитать файл в буфер

	// проверка подписи
	if (!CryptVerifyDetachedMessageSignature( // Проверить ЭП
		&VerifyPara, // pVerifyPara указатель на PCRYPT_VERIFY_MESSAGE_PARA
(информация проверки ЭП)
		0, // dwSignerIndex используется, если сообщение подписано несколькими
пользователями
		signBuffer, // *pbDetachedSignBlob адрес буфера с подписью
		sLength, // cbDetachedSignBlob размер *pbDetachedSignBlob
		1, // cToBeSigned количество элементов в массивах rgpbToBeSigned и
rgcbToBeSigned
		&buffers, // rgpbToBeSigned указатели на буферы, в которых находится то, что
нужно хэшировать
		&tbsLength, // rgcbToBeSigned указатели на размеры буферов rgpbToBeSigned
		NULL/*&pCertA*/)) // [opt] *ppSignerCert контекст сертификата подписавшего
	{
		cout << "invalid signature " << hex << GetLastError() << endl;
	}
	else
	{
		cout << "signature verified" << endl;
	}

	delete[] buffer;
	delete[] signBuffer;


	CertFreeCertificateContext( // освободить контекст сертификата
		pCertA); // pCertContext контекст сертификата

	CertFreeCertificateContext( // освободить контекст сертификата
		pCertB); // pCertContext контекст сертификата
```

```
if (!(CertCloseStore( // Закрыть хранилище
	hStoreMy, // hCertStore дескриптор закрываемого хранилища
	NULL))) // dwFlags
{
	cout << "CertCloseStore MY error" << endl;
}


if (!(CertCloseStore( // Закрыть хранилище
	hStoreRoot, // hCertStore дескриптор закрываемого хранилища
	NULL))) // dwFlags
{
	cout << "CertCloseStore ROOT error" << endl;
}

return 0;
}
```

Приложение Б

(обязательное)

Текст программы для проверки результатов выполнения практикума

main.cpp:

```
#include <iostream>
#include <string>
#include <locale.h>
#include <tchar.h>
#include "decrypt.h"
#include "hash.h"
#include "sign.h"
using namespace std;

int main()
{
	string plaintext, // путь к файлу с открытым текстом
		ciphertext, // путь к файлу с шифртекстом
		key, // путь к файлу с ключом
		passphrase, // парольная фраза
		decrypted, // путь к файлу для записи расшифрованного текста
		hash, // хэш-код
		signature, // путь к файлу с подписью
		rootCert, // путь к сертификату ЦС
		myCert; // путь к пользовательскому сертификату
	int operation = 4; // выбор действия
	setlocale(LC_ALL, ".1251");
	try
	{
		cout << "Путь к файлу с открытым текстом: ";
		cin >> plaintext;
		while (operation != 0)
		{
			cout << endl << endl << "Что необходимо проверить?\n1 -
зашифрование,\n2 - хэширование,\n3 - подпись,\n0 - выйти:\n";
			cin >> operation;
			if (1 == operation)
			{
				cout << "Путь к файлу с шифртекстом: ";
				cin >> ciphertext;
				cout << "Путь к файлу с ключом: ";
				cin >> key;
				cout << "Пароль для рашифрования ключа: ";
				cin >> passphrase;
				cout << "Путь сохранения файла с расшифрованным текстом: ";
				cin >> decrypted;
				Decrypt d(plaintext, ciphertext, key, passphrase, decrypted);
				d.Decr();
			}
			else if (2 == operation)
			{
				cout << "Хэш-код: " << endl;
				cin >> hash;
				Hash h(plaintext, hash);
				h.HashCode();
			}
			else if (3 == operation)
```

```
                {
                        cout << "Путь к файлу с ЭЦП: ";
                        cin >> signature;
                        cout << "Путь к сертификату центра сертификации: ";
                        cin >> rootCert;
                        cout << "Путь к сертификату пользователя: ";
                        cin >> myCert;
                        Sign s(plaintext, signature, rootCert, myCert);
                        s.Signature();
                }

        }
    }
    catch (invalid_argument e)
    {
        cerr << e.what() << endl;
        exit(1);
    }
    return 0;
}
```

decrypt.h:

```
#pragma once
#pragma comment (lib, "bcrypt.lib") // маршрутизатор примитивов для CNG

#include <windows.h>
#include <bcrypt.h>
#include <tchar.h>
#include <iostream>
#include <fstream>
#include <string>
using namespace std;

class Decrypt // Класс для проверки правильности зашифрования
{
private:
    NTSTATUS status = 0; // инициализация кода ошибки, возвращаемого функциями CNG
    BCRYPT_ALG_HANDLE hAlg; // дескриптор провайдера алгоритма
    DWORD cbHashObject = 0, cbData = 0; //
    PBYTE pbHashObject = NULL;              // буфер для хэш-объекта
    BCRYPT_HASH_HANDLE hHash; // дескриптор хэш-объекта
    string passphrase; // пароль
    TCHAR* szPass; // хэшируемый пароль
    PBYTE pbHash = NULL; // буфер для принятия хэш-кода
    DWORD cbHash = 0; // длина буфера для хэша
    PBYTE pbKeyObject = NULL; // буфер для ключевого объекта
    DWORD cbKeyObject = 0; // длина буфера для ключевого объекта
    BCRYPT_KEY_HANDLE hKey, // дескриптор ключевого объекта
        hKeyImp; // дескриптор импортируемого ключевого объекта
    DWORD cbOutput = 0, // размер буфера для шифртекста
        cbCiphertext = 0; // размер получившегося шифртекста
    PBYTE pbOutput = NULL; // буфер с шифртекстом
    PBYTE pbDecryptedText; // буфер для расшифрованного текста
    DWORD cbDecryptedText, // размер буфера для расшированного текста
        cDecryptedText; // размер получившегося расшифрованного текста


    BYTE *buffer;
    BYTE *bufPlain; // буфер для открытого текста
    string plainTextFilePath; // путь к файлу с открытым текстом
```

```
	string cipherTextFilePath; // путь к файлу для записи шифртекста
	string decryptedTextFilePath, // путь к файлу для записи расшифрованного текста
		keyFilePath;
	int length = 0, // длина файла
		lengthPlain = 0, // длина открытого текста (для сравнения с расшифрованным)
		lengthDecrypted = 0; // длина расшифрованного текста (для сравнения с
открытым)
	PUCHAR impKeyBuf; // буфер для импорта ключа

public:
	Decrypt(string in_plaintext, string in_ciphertext, string in_key, string
in_passphrase, string in_decrypted)
		:plainTextFilePath(in_plaintext),
		cipherTextFilePath(in_ciphertext),
		keyFilePath(in_key),
		passphrase(in_passphrase),
		decryptedTextFilePath(in_decrypted) {}; // инициализация

	void Decr(); // метод для проверки правильности зашифрования
	};
```

decrypt.cpp:

```
#include "decrypt.h"
using namespace std;

void Decrypt::Decr()
{
	// string to TCHAR for BCryptHashData
	szPass = new TCHAR[passphrase.size() + 1];
	copy(passphrase.begin(), passphrase.end(), szPass);
	szPass[passphrase.size()] = 0;

	ifstream iFkey(keyFilePath, ios_base::in | ios_base::binary);
	if (!iFkey.is_open())
	{
		cout << "Файл " << keyFilePath << " не может быть открыт" << endl;
		throw invalid_argument("Ошибка открытия файла");
	}



	// загрузка и инициализация провайдера конкретного алгоритма
	if (!BCRYPT_SUCCESS(status = BCryptOpenAlgorithmProvider(
		&hAlg, // *phAlgorithm сюда запишется дескриптор провайдера алгоритма
		BCRYPT_SHA256_ALGORITHM, // pszAlgId идентификатор (имя) алгоритма (указатель)
		NULL, // pszImplementation имя провайдера криптографических примитивов (для
BCRYPT_AES_ALGORITHM // по умолчанию будет Microsoft Primitive Provider
		0))) // dwFlags 0 для обычного шифрования/хэширования
	{
		cout << "BCryptOpenAlgorithmProvider BCRYPT_SHA256_ALGORITHM error 0x" << hex
<< status << endl;
		throw invalid_argument("Ошибка инициализации провайдера алгоритма
хэширования");
	}


	// получение буфера для хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта,
значение свойства которого необходимо получить
		BCRYPT_OBJECT_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbHashObject, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных,
реально скопированных в pbOutput
```

```
		0))) // dwFlags 0
	{
		cout << "hash object buffer creation error 0x" << hex << status << endl;
		throw invalid_argument("Ошибка получения размера хэш-объекта");
	}
	// выделение памяти для объекта
	pbHashObject = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHashObject);

	// создание хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptCreateHash(hAlg, // hAlgorithm дескриптор
алгоритма хэширования,

	// полученный с помощью BCryptOpenAlgorithmProvider
		&hHash, // *phHash указатель на дескриптор создаваемого хэш-объекта
		pbHashObject, // pbHashObject указатель на буфер, в котором размещается
создаваемый хэш - объект
		cbHashObject, // cbHashObject размер буфера pbHashObject
		NULL, // pbSecret адрес буфера, содержащего ключ, используемый для создания
кода аутентичности сообщения
		0, // cbSecret размер pbSecret
		0))) // dwFlags 0
	{
		cout << "BCryptCreateHash error 0x" << hex << status << endl;
		throw invalid_argument("Ошибка создания хэш-объекта");
	}

	// хэширование данных
	if (!BCRYPT_SUCCESS(status = BCryptHashData(hHash, // hHash дескриптор хэш-объекта
		(PBYTE)szPass, // pbInput адрес буфера с входными данными
		_tcslen(szPass) * sizeof(TCHAR), // cbInput размер pbInput
		0))) // dwFlags
	{
		cout << "BCryptHashData error 0x" << hex << status << endl;
		throw invalid_argument("Ошибка хэширования парольной фразы");
	}

	// извлечение хэш-кода из хэш-объекта
	// создание буфера для хэш-кода
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта,
значение свойства которого необходимо получить
		BCRYPT_HASH_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbHash, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных,
реально скопированных в pbOutput
		0))) // dwFlags 0
	{
		cout << "hash code buffer creation error 0x" << hex << status;
	}
	// выделение памяти для объекта (хэша)
	pbHash = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHash);
	// извлчение хэш-кода из хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptFinishHash(hHash, // hHash дескриптор хэш-объекта
		pbHash, // pbOutput буфер для принятия хэш-кода
		cbHash, // cbOutput размер буфера pbOutput
		0))) // dwFlags
	{
		cout << "BCryptFinishHash error 0x" << hex << status;
	}

	// загрузка и инициализация провайдера конкретного алгоритма (шифрования)
	if (!BCRYPT_SUCCESS(status = BCryptOpenAlgorithmProvider(
		&hAlg, // *phAlgorithm сюда запишется дескриптор провайдера алгоритма
		BCRYPT_AES_ALGORITHM, // pszAlgId идентификатор (имя) алгоритма (указатель)
```

```
		NULL, // pszImplementation имя провайдера криптографических примитивов (для BCRYPT_AES_ALGORITHM // по умолчанию будет Microsoft Primitive Provider
		0))) // dwFlags 0 для обычного шифрования/хэширования
	{
		cout << "BCryptOpenAlgorithmProvider BCRYPT_AES_ALGORITHM error 0x" << hex << status;
	}

	// получение буфера
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта, значение свойства которого необходимо получить
		BCRYPT_OBJECT_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbKeyObject, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных, реально скопированных в pbOutput
		0))) // dwFlags 0
	{
		cout << "key object buffer creation error 0x" << hex << status;
	}
	// выделение памяти для объекта
	pbKeyObject = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbKeyObject);



	// Создание ключевого объекта
	// и помещение в него ключа симметричного шифрования загруженного ранее алгоритма
	if (!BCRYPT_SUCCESS(status = BCryptGenerateSymmetricKey(hAlg, // hAlgorithm полученный с помощью BCryptOpenAlgorithmProvider дескриптор алгоритма шифрования
		&hKey, // *phKey дескриптор создаваемого ключевого объекта
		pbKeyObject, // pbKeyObject указатель на буфер, в котором размещается созданный ключевой объект
		cbKeyObject, // cbKeyObject размер буфера pbKeyObject
		(PBYTE)pbHash, // pbSecret указатель на буфер, который содержит ключевой материал
		16, // cbSecret размер буфера bpSecret
		0))) // dwFlags
	{
		cout << "BCryptGenerateSymmetricKey error 0x" << hex << status;
	}

	// импорт ключа

	iFkey.seekg(0, iFkey.end); //
	length = iFkey.tellg(); // определение длины файла
	iFkey.seekg(0, iFkey.beg); //

	impKeyBuf = new UCHAR[length];
	iFkey.read((char*)impKeyBuf, length); // прочитать файл в буфер

	if (!BCRYPT_SUCCESS(status = BCryptImportKey( // импорт ключа
		hAlg, // hAlgorithm дескриптор алгоритма
		hKey, // hImportKey дескриптор ключа, используемого для расшифрования hKey
		BCRYPT_AES_WRAP_KEY_BLOB, // pszBlobType тип BLOB-объекта,
		&hKeyImp, // *phKey сюда запишется импортированный ключ
		NULL, // pbKeyObject
		0/*cbKeyObject*/, // cbKeyObject размер pbKeyObject
		impKeyBuf, // pbInput буфер с импортируемым ключом
		length, // cbInput размер pbInput
		NULL))) // dwFlags
	{
		cout << "BCryptImportKey error 0x" << hex << status << endl;
		throw invalid_argument("Ошибка импорта ключа. Возможно, введён неверный пароль.");
```

```
	}


	ifstream iF2(cipherTextFilePath, ios_base::in | ios_base::binary);
	ofstream oF2(decryptedTextFilePath, ios_base::out | ios_base::binary);

	if (!iF2.is_open())
	{
		cout << "Файл " << cipherTextFilePath << " не может быть открыт" << endl;
		throw invalid_argument("Ошибка открытия файла");
	}

	iF2.seekg(0, iF2.end); //
	length = iF2.tellg(); // определение длины файла
	iF2.seekg(0, iF2.beg); //

	buffer = new BYTE[length];

	iF2.read((char*)buffer, length); // прочитать файл в буфер


		// расшифрование (определение размера буфера)
	if (!BCRYPT_SUCCESS(status = BCryptDecrypt(hKeyImp, // hKey дескриптор созданного
ранее  ключа
		buffer, // pbInput адрес буфера с шифртекстом
		length, // cbInput размер буфера с шифртекстом
		NULL, // *pPaddingInfo указатель на структуру, определяющую параметры
дополнения блоков
		NULL, // pbIV адрес буфера, содержащего вектор инициализации
		0, // cbIV размер вектора инициализации
		NULL, // pbOutput адрес буфера, в который будет помещаться расшифрованный
текст
		0, // cbOutput размер буфера для расшифрованного текста
		&cbDecryptedText, // *pcbResult размер получившегося расшифрованного текста
		BCRYPT_BLOCK_PADDING))) // dwFlags
	{
		cout << "BCryptDecrypt buffer size define error 0x" << hex << status;
	}



		// расшифрование
	if (!BCRYPT_SUCCESS(status = BCryptDecrypt(hKeyImp, // hKey дескриптор созданного
ранее  ключа
		buffer, // pbInput адрес буфера с шифртекстом
		length, // cbInput размер буфера с шифртекстом
		NULL, // *pPaddingInfo указатель на структуру, определяющую параметры
дополнения блоков
		NULL, // pbIV адрес буфера, содержащего вектор инициализации
		0, // cbIV размер вектора инициализации
		buffer, // pbOutput адрес буфера, в который будет помещаться расшифрованный
текст
		cbDecryptedText, // cbOutput размер буфера для расшифрованного текста
		&cDecryptedText, // *pcbResult размер получившегося расшифрованного текста
		BCRYPT_BLOCK_PADDING))) // dwFlags
	{
		cout << "decrypt error 0x" << hex << status;
	}



	oF2.write((char*)buffer, cDecryptedText); // записать расшифрованные данные в файл
	oF2.close();
	iF2.close();
```

```
	// сравнить расшифрованные данные и открытый текст
	ifstream iFplain(plainTextFilePath, ios_base::in | ios_base::binary);

	if (!iFplain.is_open())
	{
		cout << "Файл " << plainTextFilePath << " не может быть открыт" << endl;
		throw invalid_argument("Ошибка открытия файла");
	}

	iFplain.seekg(0, iFplain.end); //
	length = iFplain.tellg(); // определение длины файла
	iFplain.seekg(0, iFplain.beg); //

	bufPlain = new BYTE[length];

	iFplain.read((char*)bufPlain, length); // прочитать файл в буфер

	if (length != cDecryptedText)
	{
		cout << "- Зашифрование выполнено неверно: длины исходного открытого текста и
получившегося расшифрованного текста не равны." << endl;
	}
	else
	{
		int i = 0;
		while ((i < length) && (buffer[i] == bufPlain[i]))
		{
			i++;
		}
		if (i == length)
		{
			cout << "+ Зашифрование выполнено верно." << endl;
		}
		else
		{
			cout << "- Зашифрование выполнено неверно." << endl;
		}
	}

	delete[] buffer;


	if (!BCRYPT_SUCCESS(status = BCryptDestroyHash(hHash))) // уничтожение хэш-объекта
	{
		cout << "BCryptDestroyHash error 0x" << hex << status;
	}

	if (!BCRYPT_SUCCESS(status = BCryptDestroyKey(hKey))) // уничтожение ключевого
объекта
	{
		cout << "BCryptDestroyKey error 0x" << hex << status;
	}

	if (!BCRYPT_SUCCESS(status = BCryptCloseAlgorithmProvider( // выгрузка провайдера
алгоритма
		hAlg, // hAlgorithm дескриптор провайдера
		0))) // dwFlags
	{
		cout << "BCryptCloseAlgorithmProvider error 0x" << hex << status;
	}
```

```
	//освобождение объекта
	HeapFree(GetProcessHeap(), 0, pbHashObject); // освобождение буфера хэш-объекта

		//освобождение объекта
	HeapFree(GetProcessHeap(), 0, pbHash); // освобождение буфера хэша

	HeapFree(GetProcessHeap(), 0, pbKeyObject); // освобождение буфера ключевого объекта


	}
```

hash.h:

```
#pragma once
#pragma comment (lib, "bcrypt.lib") // маршрутизатор примитивов для CNG

#include <windows.h>
#include <bcrypt.h>
#include <sstream>
#include <fstream>
#include <iostream>
using namespace std;

class Hash // Класс для проверки правильности вычисления хэш-кода
{
private:
	NTSTATUS status = 0; // инициализация кода ошибки, возвращаемого функциями CNG
	BCRYPT_ALG_HANDLE hAlg; // дескриптор провайдера алгоритма
	DWORD cbHashObject = 0, cbData = 0; //
	PBYTE pbHashObject = NULL;		// буфер для хэш-объекта
	BCRYPT_HASH_HANDLE hHash; // дескриптор хэш-объекта
	PBYTE pbHash = NULL; // буфер для принятия хэш-кода
	DWORD cbHash = 0; // длина буфера для хэша


	BYTE *buffer;
	string filePath; // путь к файлу, для которого будет вычисляться хэш
	int length = 0; // длина файла
	string hash; // хэш-код, вводимый пользователем
	ostringstream oStrHash; // вычисленный хэш

public:
	Hash(string in_plaintext, string in_hash)
		:filePath(in_plaintext),
		hash(in_hash) {}; // инициализация

	void HashCode(); // метод для проверки правильности зашифрования


	};
```

hash.cpp:

```
#include "hash.h"
using namespace std;


void Hash::HashCode()
{
	ifstream iF(filePath, ios_base::in | ios_base::binary);

	if (!iF.is_open())
	{
```

```
            cout << "Файл " << filePath << " не может быть открыт" << endl;
            throw invalid_argument("Ошибка открытия файла");
        }


        // загрузка и инициализация провайдера конкретного алгоритма
        if (!BCRYPT_SUCCESS(status = BCryptOpenAlgorithmProvider(
                &hAlg, // *phAlgorithm сюда запишется дескриптор провайдера алгоритма
                BCRYPT_SHA256_ALGORITHM, // pszAlgId идентификатор (имя) алгоритма (указатель)
                NULL, // pszImplementation имя провайдера криптографических примитивов (для
BCRYPT_AES_ALGORITHM // по умолчанию будет Microsoft Primitive Provider
                0))) // dwFlags 0 для обычного шифрования/хэширования
        {
                cout << "BCryptOpenAlgorithmProvider BCRYPT_SHA256_ALGORITHM error 0x" << hex
<< status;
        }


        // получение буфера для хэш-объекта
        if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта,
значение свойства которого необходимо получить
                BCRYPT_OBJECT_LENGTH, // pszProperty имя свойства (указатель)
                (PBYTE)&cbHashObject, // pbOutput сюда передастся значение свойства
                sizeof(DWORD), // cbOutput размер буфера pbOutput
                &cbData, // *pcbResult адрес переменной, куда возвращается размер данных,
реально скопированных в pbOutput
                0))) // dwFlags 0
        {
                cout << "hash object buffer creation error 0x" << hex << status;
        }
        // выделение памяти для объекта
        pbHashObject = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHashObject);

        // создание хэш-объекта
        if (!BCRYPT_SUCCESS(status = BCryptCreateHash(hAlg, // hAlgorithm дескриптор
алгоритма хэширования,

        // полученный с помощью BCryptOpenAlgorithmProvider
                &hHash, // *phHash указатель на дескриптор создаваемого хэш-объекта
                pbHashObject, // pbHashObject указатель на буфер, в котором размещается
создаваемый хэш - объект
                cbHashObject, // cbHashObject размер буфера pbHashObject
                NULL, // pbSecret адрес буфера, содержащего ключ, используемый для создания
кода аутентичности сообщения
                0, // cbSecret размер pbSecret
                0))) // dwFlags 0
        {
                cout << "BCryptCreateHash error 0x" << hex << status;
        }

        iF.seekg(0, iF.end); //
        length = iF.tellg(); // определение длины файла
        iF.seekg(0, iF.beg); //

        buffer = new BYTE[length];

        iF.read((char*)buffer, length); // прочитать файл в буфер

        // хэширование данных
        if (!BCRYPT_SUCCESS(status = BCryptHashData(hHash, // hHash дескриптор хэш-объекта
                buffer, // pbInput адрес буфера с входными данными
                length, // cbInput размер pbInput
                0))) // dwFlags
        {
```

```
		cout << "BCryptHashData error 0x" << hex << status;
	}

	// извлечение хэш-кода из хэш-объекта
	// создание буфера для хэш-кода
	if (!BCRYPT_SUCCESS(status = BCryptGetProperty(hAlg, // hObject дескриптор объекта,
значение свойства которого необходимо получить
		BCRYPT_HASH_LENGTH, // pszProperty имя свойства (указатель)
		(PBYTE)&cbHash, // pbOutput сюда передастся значение свойства
		sizeof(DWORD), // cbOutput размер буфера pbOutput
		&cbData, // *pcbResult адрес переменной, куда возвращается размер данных,
реально скопированных в pbOutput
		0))) // dwFlags 0
	{
		cout << "hash code buffer creation error 0x" << hex << status;
	}
	// выделение памяти для объекта (хэша)
	pbHash = (PBYTE)HeapAlloc(GetProcessHeap(), 0, cbHash);
	// извлечение хэш-кода из хэш-объекта
	if (!BCRYPT_SUCCESS(status = BCryptFinishHash(hHash, // hHash дескриптор хэш-объекта
		pbHash, // pbOutput буфер для принятия хэш-кода
		cbHash, // cbOutput размер буфера pbOutput
		0))) // dwFlags
	{
		cout << "BCryptFinishHash error 0x" << hex << status;
	}

	//вывод хэш-кода на экран
	cout << "hash: ";
	for (int i = 0; i < cbHash; i++)
	{
		oStrHash << hex << (int)pbHash[i];
	}
	cout << oStrHash.str() << endl;

	if (oStrHash.str() == hash)
	{
		cout << "+ Хэширование выполнено верно" << endl;
	}
	else
	{
		cout << "- Хэширование выполнено неверно" << endl;
	}


	iF.close();

	delete[] buffer;


	if (!BCRYPT_SUCCESS(status = BCryptDestroyHash(hHash))) // уничтожение хэш-объекта
	{
		cout << "BCryptDestroyHash error 0x" << hex << status;
	}


	if (!BCRYPT_SUCCESS(status = BCryptCloseAlgorithmProvider( // выгрузка провайдера
алгоритма
		hAlg, // hAlgorithm дескриптор провайдера
		0))) // dwFlags
	{
		cout << "BCryptCloseAlgorithmProvider error 0x" << hex << status;
	}
```

```
	//освобождение объекта
	HeapFree(GetProcessHeap(), 0, pbHashObject); // освобождение буфера хэш-объекта

		//освобождение объекта
	HeapFree(GetProcessHeap(), 0, pbHash); // освобождение буфера хэша
	}
```

sign.h:

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

#include <windows.h>
#include <wincrypt.h>
#include <fstream>
#include <string>
#include <iostream>

#define MY_ENCODING_TYPE (PKCS_7_ASN_ENCODING | X509_ASN_ENCODING)

using namespace std;

class Sign // Класс для проверки правильности формирования ЭЦП
{
private:
	HCERTSTORE hStoreMy = NULL,
		hStoreRoot = NULL; // хранилище
	PCCERT_CONTEXT pCert = NULL; // указатель на предыдущий сертификат для
CertEnumCertificatesInStore
	TCHAR szNameString[128]; // имя для CertGetNameString
	string pathCertRoot, pathCertUser; // пути сертификатов
	PCCERT_CONTEXT pCertA = NULL, // контекст сертификата A
		pCertB = NULL;  // контекст сертификата B
	DWORD signatureAndTimeCheck = CERT_STORE_SIGNATURE_FLAG |
CERT_STORE_TIME_VALIDITY_FLAG;
	CRYPT_SIGN_MESSAGE_PARA SignPara = { sizeof(CRYPT_SIGN_MESSAGE_PARA) }; // параметры
электронной подписи для CryptSignAndEncryptMessage
	CRYPT_VERIFY_MESSAGE_PARA VerifyPara = { sizeof(CRYPT_VERIFY_MESSAGE_PARA) }; //
информация для проверки ЭП для CryptDecryptAndVerifyMessageSignature


	BYTE *buffer; // данные

	BYTE *signBuffer = NULL; // подпись
	string file; // путь к файлу с открытым текстом
	string signIn; // путь к файлу с подписью для проверки
	//string decryptedTextFilePath; // путь к файлу для записи расшифрованного текста
	DWORD tbsLength = 0, // длина файла
		sLength = 0; // длина подписи
	DWORD cbOutput = 0, // размер буфера для результата sign
		tail = 2000;
	int length; // длина файла

public:
	Sign(string in_plaintext, string in_signature, string in_rootCert, string in_myCert)
		:file(in_plaintext),
		signIn(in_signature),
		pathCertRoot(in_rootCert),
		pathCertUser(in_myCert)    {}; // инициализация

	void Signature(); // метод для проверки правильности формирования ЭЦП
	};
```

sign.cpp:

```
#include "sign.h"
using namespace std;

PCCERT_CONTEXT WINAPI PfnCryptGetSignerCertificate( // для использования сертификата из
файла der вместо сертификата, встроенного в подпись
	void *pvGetArg,
	DWORD dwCertEncodingType,
	PCERT_INFO pSignerId,
	HCERTSTORE hMsgCertStore)
{
	return (PCCERT_CONTEXT)pvGetArg;
}

void Sign::Signature()
{
	ifstream iFcertRoot(pathCertRoot, ios_base::in | ios_base::binary);

	if (!iFcertRoot.is_open())
	{
		cout << "Файл " << pathCertRoot << " не может быть открыт" << endl;
		throw invalid_argument("Ошибка открытия файла");
	}

	if (!(hStoreRoot = CertOpenSystemStore( // открыть системное хранилище
		NULL, // hprov (не используется)
		TEXT("ROOT")))) // szSubsystemProtocol имя системного хранилища сертификатов
					// MY - Личное
					// ROOT - Доверенные корневые центры сертификации
	{
		cout << "CertOpenSystemStore ROOT Error" << endl;
	}

	iFcertRoot.seekg(0, iFcertRoot.end);	//
	length = iFcertRoot.tellg();			// определение длины файла
	iFcertRoot.seekg(0, iFcertRoot.beg);	//

	buffer = new BYTE[length];

	iFcertRoot.read((char*)buffer, length); // прочитать файл в буфер

	if (!(CertAddEncodedCertificateToStore( // добавить сертификат в хранилище
		hStoreRoot, // hCertStore дескриптор хранилища сертификатов
		MY_ENCODING_TYPE, // dwCertEncodingType тип кодировки сертификата
		buffer, // *pbCertEncoded буфер, в котором находится сертификат
		length, // cbCertEncoded размер *pbCertEncoded
		CERT_STORE_ADD_REPLACE_EXISTING, // dwAddDisposition что делать, если
сертификат существует
		NULL))) // *ppCertContext указатель на контекст сертификата (для дальнейшего
использования)
	{
		cout << "CertAddEncodedCertificateToStore (CA's cert) error" << hex <<
GetLastError() << endl;
	}
	else
	{
		cout << "Сертификат центра сертификации установлен" << endl;
	}


	ifstream iFcertUser(pathCertUser, ios_base::in | ios_base::binary);

	if (!iFcertUser.is_open())
	{
		cout << "Файл " << pathCertUser << " не может быть открыт" << endl;
```

```
		throw invalid_argument("Ошибка открытия файла");
	}

	if (!(hStoreMy = CertOpenSystemStore( // открыть системное хранилище
		NULL, // hprov (не используется)
		TEXT("MY")))) // szSubsystemProtocol имя системного хранилища сертификатов
					// MY - Личное
					// ROOT - Доверенные корневые центры сертификации
	{
		cout << "CertOpenSystemStore MY Error" << endl;
	}

	iFcertUser.seekg(0, iFcertUser.end);	//
	length = iFcertUser.tellg();			// определение длины файла
	iFcertUser.seekg(0, iFcertUser.beg);	//

	delete[] buffer;
	buffer = new BYTE[length];

	iFcertUser.read((char*)buffer, length); // прочитать файл в буфер

	if (!(CertAddEncodedCertificateToStore( // добавить сертификат в хранилище
		hStoreMy, // hCertStore дескриптор хранилища сертификатов
		MY_ENCODING_TYPE, // dwCertEncodingType тип кодировки сертификата
		buffer, // *pbCertEncoded буфер, в котором находится сертификат
		length, // cbCertEncoded размер *pbCertEncoded
		CERT_STORE_ADD_REPLACE_EXISTING, // dwAddDisposition что делать, если
сертификат существует
		&pCertA))) // *ppCertContext указатель на контекст сертификата (для
дальнейшего использования)
	{
		cout << "CertAddEncodedCertificateToStore (user's cert) error" << hex <<
GetLastError() << endl;
	}
	else
	{
		cout << "Сертификат пользователя установлен" << endl;
	}

	ifstream iFtoBeSigned(file, ios_base::in | ios_base::binary);

	if (!iFtoBeSigned.is_open())
	{
		cout << "Файл " << file << " не может быть открыт" << endl;
		throw invalid_argument("Ошибка открытия файла");
	}


	if (!(CertGetIssuerCertificateFromStore( // получить сертфификат ЦС и проверить
целостность сертификата пользователя A
		hStoreRoot, // hCertStore дескриптор открытого хранилища, в котором ищется
сертификат ЦС
		pCertA, // pSubjectContext контекст пользовательского сертификата, для
которого ищется сертификат ЦС
		NULL, // pPrevIssuerContext контекст предыдущего найденного сертификата этого
ЦС
		&signatureAndTimeCheck))) // *pdwFlags характер проверки пользовательского
сертификата
	{
		cout << "CertGetIssuerCertificateFromStore error";
	}

	if (signatureAndTimeCheck)
	{
```

```
		cout << "cert check error: wrong signature or time" << endl;
	}

	iFtoBeSigned.seekg(0, iFtoBeSigned.end); //
	tbsLength = iFtoBeSigned.tellg();               // определение длины файла
	iFtoBeSigned.seekg(0, iFtoBeSigned.beg); //

	buffer = new BYTE[tbsLength];
	const BYTE *buffers = { buffer };

	iFtoBeSigned.read((char*)buffer, tbsLength); // прочитать файл в буфер

	// информация для проверки ЭП (структура)
	VerifyPara.dwMsgAndCertEncodingType = MY_ENCODING_TYPE;
	VerifyPara.pfnGetSignerCertificate = PfnCryptGetSignerCertificate;
	VerifyPara.pvGetArg = (void*)pCertA;

	ifstream iFsign(signIn, ios_base::in | ios_base::binary);

	if (!iFsign.is_open())
	{
		cout << "Файл " << signIn << " не может быть открыт" << endl;
		throw invalid_argument("Ошибка открытия файла");
	}

	iFsign.seekg(0, iFsign.end); //
	sLength = iFsign.tellg();  // определение длины файла
	iFsign.seekg(0, iFsign.beg); //

	signBuffer = new BYTE[sLength];

	iFsign.read((char*)signBuffer, sLength); // прочитать файл в буфер

	// проверка подписи
	if (!CryptVerifyDetachedMessageSignature( // Проверить ЭП
		&VerifyPara, // pVerifyPara указатель на PCRYPT_VERIFY_MESSAGE_PARA
(информация проверки ЭП)
		0, // dwSignerIndex используется, если сообщение подписано несколькими
пользователями
		signBuffer, // *pbDetachedSignBlob адрес буфера с подписью
		sLength, // cbDetachedSignBlob размер *pbDetachedSignBlob
		1, // cToBeSigned количество элементов в массивах rgpbToBeSigned и
rgcbToBeSigned
		&buffers, // rgpbToBeSigned указатели на буферы, в которых находится то, что
нужно хэшировать
		&tbsLength, // rgcbToBeSigned указатели на размеры буферов rgpbToBeSigned
		NULL/*&pCertA*/)) // [opt] *ppSignerCert контекст сертификата подписавшего
	{
		cout << "invalid signature " << hex << GetLastError() << endl;
		cout << "- Электронная цифровая подпись сформирована неверно" << endl;
	}
	else
	{
		cout << "+ Электронная цифровая подпись сформирована верно" << endl;
	}

	delete[] buffer;
	delete[] signBuffer;


	CertFreeCertificateContext( // освободить контекст сертификата
		pCertA); // pCertContext контекст сертификата
```

```
	if (!(CertCloseStore( // Закрыть хранилище
		hStoreMy, // hCertStore дескриптор закрываемого хранилища
		NULL))) // dwFlags
	{
		cout << "CertCloseStore MY error" << endl;
	}


	if (!(CertCloseStore( // Закрыть хранилище
		hStoreRoot, // hCertStore дескриптор закрываемого хранилища
		NULL))) // dwFlags
	{
		cout << "CertCloseStore ROOT error" << endl;
	}
}
```