

Пенза 2023

Оглавление

Введение.....	5
1 Обзор криптографических библиотек в операционных системах семейства Linux.....	7
2 Анализ возможностей криптографических библиотек.....	9
3 Разработка структуры практикума.....	12
4 Разработка порядка проведения практикума.....	16
5 Разработка рекомендаций по проведению практикума.....	19
5.1 Установка криптографической библиотеки и настройка проекта.....	19
5.2 Рекомендации для разработки программы шифрования AES-256 GCM.....	21
5.3 Рекомендации для разработки программы вычисления хэш-функции SHA-256.....	23
5.4 Рекомендации для разработки программы шифрования и генерации ключей RSA.....	24
5.5 Рекомендации для разработки программы вычисления и проверки электронной подписи.....	29
5.6 Рекомендации для разработки программы создания сертификата и проверки электронной подписи.....	32
6 Разработка контрольных примеров и вариантов для проведения практикума.....	37
6.1 Разработка контрольных примеров для проведения практикума.....	37
6.2 Разработка вариантов для проведения практикума.....	41
7 Разработка примеров реализации практикума.....	49
7.1 Алгоритмы реализации функций по применению криптографической библиотеки.....	49
7.2 Проверка работоспособности примеров реализации функций по применению криптографической библиотеки.....	56

7.2.1 Проверка работоспособности программы шифрования AES-256 GCM.....	56
7.2.2 Проверка работоспособности программы вычисления хэш-функции SHA-256.....	58
7.2.3 Проверка работоспособности программы шифрования и генерации ключей RSA.....	59
7.2.4 Проверка работоспособности программы для вычисления и проверки электронной подписи.....	62
7.2.5 Проверка работоспособности программы создания сертификата и проверки электронной подписи.....	63
8 Разработка приложения для проверки результатов выполнения практикума.....	65
8.1 Словесное описание разрабатываемого приложения.....	65
8.2 Проверка работоспособности приложения для проверки результатов выполнения практикума.....	68
8.2.1 Проверка работоспособности приложения в режиме проверки шифрования AES-256 GCM.....	68
8.2.2 Проверка работоспособности приложения в режиме проверки хэш-функции SHA-256.....	69
8.2.3 Проверка работоспособности приложения в режиме проверки шифрования RSA.....	70
8.2.4 Проверка работоспособности приложения в режиме проверки электронной подписи с помощью открытого ключа.....	72
8.2.5 Проверка работоспособности приложения в режиме проверки электронной подписи с помощью сертификата.....	74
9 Вредные психофизиологические факторы и методы их снижения в работе специалиста по защите информации.....	77
9.1 Вредные психофизиологические факторы, влияющие на опорно-двигательный аппарат специалиста по защите информации.....	77

9.2 Методы и средства физической культуры, снижающие воздействие психофизиологических факторов.....	78
Заключение.....	80
Список использованных источников.....	81
Приложение А. Обязательное. Коды примеров реализации практикума.....	83
Приложение Б. Обязательное. Код приложения для проверки результатов выполнения практикума.....	105

Введение

В современном мире безопасность является одним из наиболее важных факторов, требующих особого внимания. Криптографические технологии являются одним из наиболее эффективных средств, с помощью которых можно повысить уровень безопасности.

При разработке приложений для защиты информации используются криптографические преобразования. Для этого используются компоненты операционных систем (криптобиблиотеки, криптопровайдеры), предоставляющие разработчикам приложений необходимые криптографические преобразования.

Для выпускной квалификационной работы была выбрана тема "Практикум по программной реализации криптографических преобразований с использованием библиотек в операционных системах семейства Linux".

Криптография является обязательным компонентом любой системы защиты информации. В связи с этим изучение криптографии является важным направлением обучения студентов в области информационной безопасности. Однако только теоретических знаний недостаточно для успешного применения криптографических методов на практике. Для того, чтобы студенты научились реализовывать криптографические преобразования на практике, им необходимо знать, как использовать криптографические библиотеки в операционных системах.

Интерес к использованию криптографических преобразований в современных операционных системах семейства Linux растет с каждым днем. Это связано с тем, что Linux является открытой операционной системой, что предоставляет возможности для создания и настройки различных систем безопасности. Кроме того, Linux – бесплатная система, что делает ее особенно привлекательной для студентов и преподавателей.

Таким образом, тема "Практикум по программной реализации

криптографических преобразований с использованием библиотек в операционных системах семейства Linux" является актуальной, так как поможет студентам приобрести практические навыки по реализации криптографических преобразований в Linux, что является важной составляющей при изучении информационной безопасности.

Цель работы: разработка практикума по программной реализации криптографических преобразований с использованием библиотек в операционных системах семейства Linux.

Для достижения поставленной цели необходимо решить следующие задачи:

- провести обзор криптографических библиотек в операционных системах семейства Linux;
- провести анализ возможностей криптографических библиотек;
- разработать структуру практикума;
- разработать порядок проведения практикума;
- разработать примеры реализации практикума;
- разработать приложение для проверки результатов выполнения практикума.

Дипломный проект состоит из 9 глав. В первой главе проведён обзор криптографических библиотек в операционных системах семейства Linux. Во второй главе – анализ возможностей криптографических библиотек. В третьей главе разработана структура практикума. В четвёртой главе выполнена разработка порядка проведения практикума. В пятой главе разработаны рекомендации по проведению практикума. В шестой главе проведена разработка контрольных примеров и вариантов для проведения практикума. В седьмой главе разработаны примеры реализации практикума. В восьмой главе выполнена разработка приложения для проверки результатов выполнения практикума. В девятой главе определены вредные психофизиологические факторы и методы их снижения в работе специалиста по защите информации.

1 Обзор криптографических библиотек в операционных системах семейства Linux

Криптографические библиотеки для семейства операционных систем Linux – это наборы программных инструментов, которые позволяют создавать, управлять и усиливать шифрование.

Были рассмотрены только криптографические библиотеки с открытым исходным кодом для семейства операционных систем linux:

- Botan;
- Cryptlib;
- Crypto++;
- Libgcrypt;
- Nettle;
- OpenSSL;
- WolfCrypt.

Криптографическая библиотека Botan [1] поддерживает алгоритмы с открытым ключом (RSA, DSA, ECDSA, EdDSA, DH, ECDH, ElGamal, DSS), режимы шифрования (CTR, CBC, XTS, CFB, OFB, EAX, OCB, GCM, SIV, CCM, ChaCha20-Poly1305), криптографические хэш-функции (MD5, SHA-1, SHA-2, SHA-3, RIPEMD-160, Whirlpool, BLAKE2b, SM3), эллиптические кривые (NIST, SECG, ECC Brainpool, Curve25519, SM2).

Криптографическая библиотека Cryptlib [2] поддерживает алгоритмы с открытым ключом (RSA, DSA, ECDSA, DH, ECDH, ElGamal, DSS), режимы шифрования (ECB, CBC, CFB, GCM), криптографические хэш-функции (MD5, SHA-1, SHA-2), эллиптические кривые (NIST, SECG, ECC Brainpool).

Криптографическая библиотека Crypto++ [3] поддерживает алгоритмы с открытым ключом (RSA, DSA, ECDSA, DH, ECDH, ElGamal, DSS), режимы шифрования (ECB, CTR, CBC, XTS, CFB, OFB, EAX, GCM, CCM, ChaCha20-Poly1305), криптографические хэш-функций (MD2, MD4, MD5, SHA-1, SHA-2,

SHA-3, RIPEMD-160, TIGER, Whirlpool, BLAKE2b, BLAKE2s, SM3), эллиптические кривые (NIST, SECG, ECC Brainpool).

Криптографическая библиотека Libgcrypt [4] поддерживает алгоритмы с открытым ключом (RSA, DSA, ECDSA, EdDSA, DH, ECDH, ElGamal, DSS), режимы шифрования (ECB, CTR, CBC, XTS, CFB, OFB, EAX, OCB, GCM, SIV, CCM), криптографические хэш-функций (MD2, MD4, MD5, SHA-1, SHA-2, SHA-3, TIGER, Whirlpool, BLAKE2b, BLAKE2s, SM3), эллиптические кривые (NIST, SECG, ECC Brainpool, Curve25519, Curve448, SM2).

Криптографическая библиотека Nettle [5] поддерживает алгоритмы с открытым ключом (RSA, DSA, EdDSA), режимы шифрования (ECB, CTR, CBC, XTS, CFB, EAX, GCM, SIV, CCM, ChaCha20-Poly1305), криптографические хэш-функций (MD2, MD4, MD5, SHA-1, SHA-2, SHA-3, RIPEMD-160, Poly1305, SM3), эллиптические кривые (NIST, SECG, Curve25519, Curve448).

Криптографическая библиотека OpenSSL [6] поддерживает алгоритмы с открытым ключом (RSA, DSA, ECDSA, EdDSA, DH, ECDH), режимы шифрования (ECB, CTR, CBC, XTS, CFB, OFB, OCB, GCM, CCM, ChaCha20-Poly1305), криптографические хэш-функций (MD2, MD4, MD5, SHA-1, SHA-2, SHA-3, BLAKE2b, BLAKE2s, SM3), эллиптические кривые (NIST, SECG, ECC Brainpool, Curve25519, Curve448, SM2).

Криптографическая библиотека WolfCrypt [7] поддерживает алгоритмы с открытым ключом (RSA, DSA, ECDSA, EdDSA, DH, ECDH, DSS), режимы шифрования (ECB, CTR, CBC, XTS, CFB, OFB, GCM, CCM, ChaCha20-Poly1305), криптографические хэш-функций (MD2, MD4, MD5, SHA-1, SHA-2, SHA-3, RIPEMD-160, Poly1305), эллиптические кривые (NIST, ECC Brainpool, Curve25519, Curve448).

2 Анализ возможностей криптографических библиотек

В таблице 1 представлена поддержка криптографическими библиотеками алгоритмов с открытым ключом.

Таблица 1 – Поддержка криптографическими библиотеками алгоритмов с открытым ключом

	Алгоритмы							
Библиотеки	RSA	DSA	ECDSA	EdDSA	DH	ECDH	ElGamal	DSS
Botan[1]	+	+	+	+	+	+	+	+
Cryptlib[2]	+	+	+	-	+	+	+	+
Crypto++[3]	+	+	+	-	+	+	+	+
Libgcrypt[4]	+	+	+	+	+	+	+	+
Nettle[5]	+	+	-	+	-	-	-	-
OpenSSL[6]	+	+	+	+	+	+	-	-
WolfCrypt[7]	+	+	+	+	+	+	-	+

В таблице 2 представлена поддержка криптографическими библиотеками режимов шифрования.

Таблица 2 – Поддержка криптографическими библиотеками режимов шифрования

	Режимы шифрования					
Библиотеки	ECB	CTR	CBC	XTS	CFB	OFB
Botan	-	+	+	+	+	+
Cryptlib	+	-	+	-	+	-
Crypto++	+	+	+	+	+	+
Libgcrypt	+	+	+	+	+	+
Nettle	+	+	+	+	+	-
OpenSSL	+	+	+	+	+	+
WolfCrypt	+	+	+	+	+	+

В таблице 3 представлена поддержка криптографическими библиотеками аутентифицированных режимов шифрования.

Таблица 3 – Поддержка криптографическими библиотеками аутентифицированных режимов шифрования

	Аутентифицированные режимы шифрования					
Библиотеки	EAX	OCB	GCM	SIV	CCM	ChaCha20-Poly1305
Botan	+	+	+	+	+	+
Cryptlib	-	-	+	-	-	-
Crypto++	+	-	+	-	+	+
Libgcrypt	+	+	+	+	+	-
Nettle	+	-	+	+	+	+
OpenSSL	-	+	+	-	+	+
WolfCrypt	-	-	+	-	+	+

В таблице 4 представлена поддержка библиотеками криптографических хэш-функций.

Таблица 4 – Поддержка библиотеками криптографических хэш-функций

	Хэш-функции							
Библиотеки	MD2/ MD4/ MD5	SHA-1/ SHA-2/ SHA-3	RIPEMD- 160	Poly1305	TIGER	Whirlpool	BLAKE2 (b/s)	SM3
Botan	-/-/+	+/+/+	+	-	-	+	+/-	+
Cryptlib	-/-/+	+/+/-	-	-	-	-	-/-	-
Crypto++	+/+/+	+/+/+	+	-	+	+	+/+	+
Libgcrypt	+/+/+	+/+/+	-	-	+	+	+/+	+
Nettle	+/+/+	+/+/+	+	+	-	-	-/-	+
OpenSSL	-+/+	+/+/+	+	-	-	+	+/+	+
WolfCrypt	+/+/+	+/+/+	+	+	-	-	-/-	-

В таблице 5 представлена поддержка криптографическими библиотеками эллиптических кривых.

Таблица 5 – Поддержка криптографическими библиотеками эллиптических кривых

	Эллиптические кривые					
Библиотеки	NIST	SECG	ECC Brainpool	Curve25519	Curve448	SM2
Botan	+	+	+	+	-	+
Cryptlib	+	+	+	-	-	-

Продолжение таблицы 5

	Эллиптические кривые					
Библиотеки	NIST	SECG	ECC Brainpool	Curve25519	Curve448	SM2
Crypto++	+	+	+	+	-	-
Libgcrypt	+	+	+	+	+	+
Nettle	+	+	-	+	+	-
OpenSSL	+	+	+	+	+	+
WolfCrypt	+	-	+	+	+	-

3 Разработка структуры практикума

Практикум предназначен для студентов, изучающих основы криптографии и программирования на C++. В практикуме студенты изучат основные алгоритмы шифрования, подписи, вычисления хэш-функций, работы с сертификатами, а также научатся работать с криптографическими библиотеками. Практикум содержит пять лабораторных работ, каждая из которых состоит из нескольких заданий.

Задания на лабораторную работу 1:

- установить криптографическую библиотеку OpenSSL на Linux;
- настроить проект в CodeLite для работы с OpenSSL;
- на языке программирования C++ написать программный код, который реализует симметричный алгоритм шифрования AES-256 GCM. Открытый текст должен считываться из файла. Код программы представить в отчёте о выполненной работе;
- выполнить тестирование разработанной программы с помощью контрольного примера и отразить результаты в отчёте о выполненной работе;
- согласно своего варианта зашифровать открытый текст, а затем расшифровать полученный шифротекст. Отрастить результаты в отчёте о выполненной работе.

Задания на лабораторную работу 2:

- на языке программирования C++ написать программный код, который реализует хэш-функцию. Текст должен считываться из файла. Хэш должен записываться в файл. Код программы представить в отчёте о выполненной работе;
- выполнить тестирование разработанной программы с помощью контрольных примеров и отразить результаты в отчёте о выполненной работе;
- модернизировать программу, чтобы выполнялась проверка целостности файла. Код программы представить в отчёте о выполненной работе;

- согласно своего варианта задания проверить целостность файла.

Результаты проверки отразить в отчёте о выполненной работе.

Задания к лабораторной работе 3:

- на языке программирования C++ написать программный код, который реализует создание ключевой пары для асимметричного алгоритма шифрования RSA. Ключи должны быть записаны в файл;

- на языке программирования C++ написать программный код, который реализует асимметричный алгоритм шифрования RSA. Открытый текст должен считываться из файла. Код программы представить в отчёте о выполненной работе;

- согласно своего варианта зашифровать исходный текст, а затем расшифровать полученный шифротекст. Отобразить ключи шифрования и результаты шифрования в отчёте о выполненной работе.

Задания на лабораторную работу 4:

- на языке программирования C++ написать программный код, который реализует создание электронной подписи. Текст должен считываться из файла. Подпись должна записываться в файл. Код программы представить в отчёте о выполненной работе;

- выполнить тестирование разработанной программы с помощью контрольного примера и отразить результаты в отчёте о выполненной работе;

- модернизировать программу, чтобы выполнялась проверка электронной подписи. Код программы представить в отчёте о выполненной работе;

- согласно своего варианта задания создать и проверить электронную подпись файла. Результаты проверки отразить в отчёте о выполненной работе.

Задания на лабораторную работу 5:

- создать сертификат, используя ключи из контрольного примера;

- на языке программирования C++ написать программный код, который реализует проверку электронной подписи. Текст должен считываться из файла. Открытый ключ нужно получать из сертификата. Код программы представить в

отчёте о выполненной работе;

- выполнить тестирование разработанной программы с помощью контрольного примера, и отразить результаты в отчёте о выполненной работе;
- согласно своего варианта задания создать и проверить электронную подпись файла с помощью сертификата. Результаты проверки отразить в отчёте о выполненной работе.

Диаграмма вариантов использования практикума представлена на рисунке 1.

Практикум позволяет студентам познакомиться с основными алгоритмами шифрования, подписи, вычисления хэш-функций, работы с сертификатами, а также научиться работать с криптографическими библиотеками. В ходе практикума студенты получают практические навыки работы с языком программирования C++ и различными инструментами криптографии.

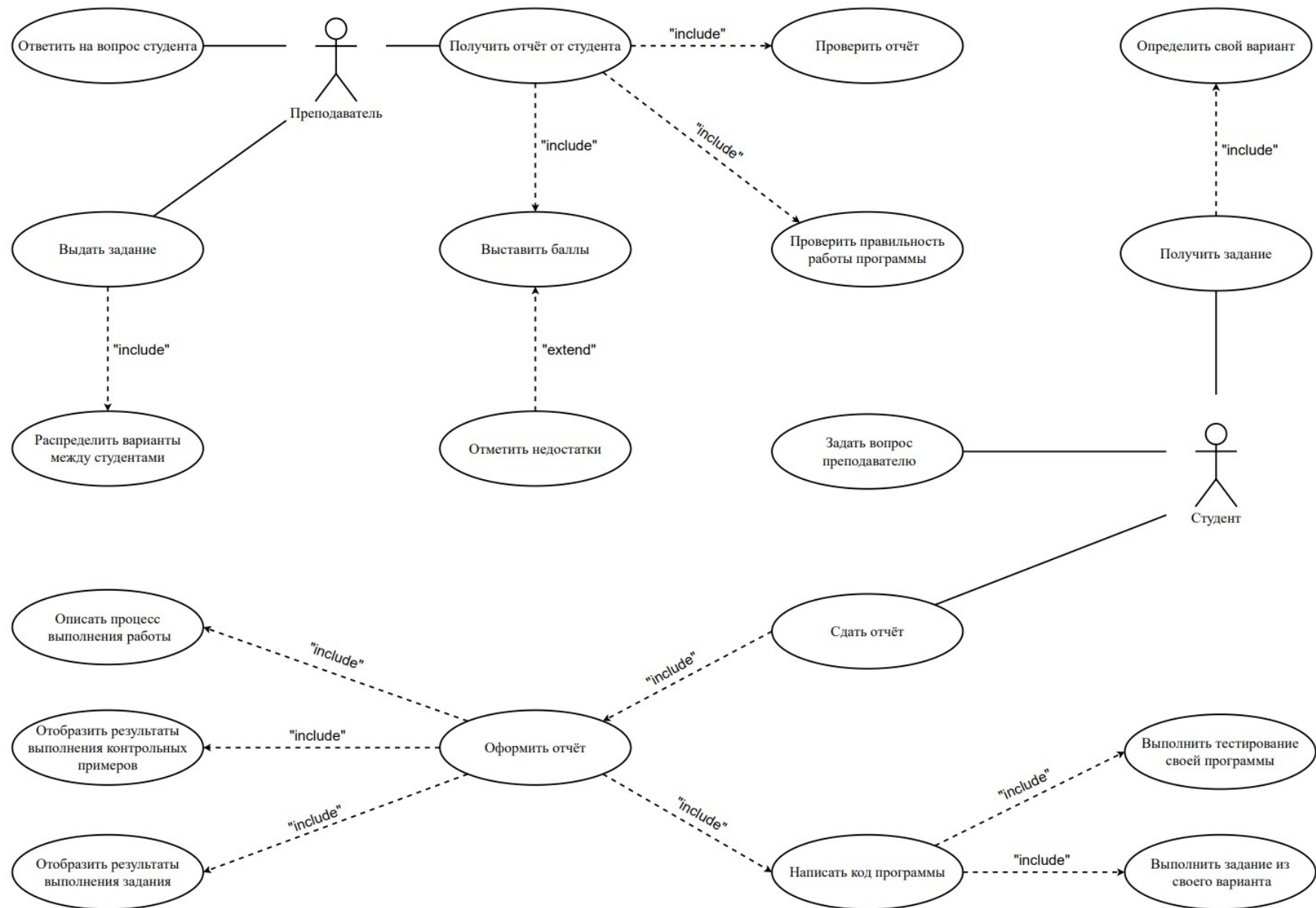


Рисунок 1 – Диаграмма вариантов использования практикума

4 Разработка порядка проведения практикума

Был разработан порядок проведения практикума.

Для каждой лабораторной работы были разработаны варианты заданий, в которых содержится необходимая информация для выполнения практикума.

Порядок проведения практикума:

- получение варианта задания и рекомендаций к выполнению практикума;
- выполнение заданий практикума;
- написание кода программы;
- написании отчёта в соответствии с требованиями;
- сдача отчёта преподавателю.

Диаграмма последовательности действий по выполнению практикума представлена на рисунках 2 и 3.

В ходе выполнения практикума необходимо следовать рекомендациям по проведению практикума, использовать материалы лекций, дополнительные пособия, методические указания, источники из интернета, задавать вопросы преподавателю на занятии.

При написании отчёта изложение материала должно идти в логическом порядке. Должны отсутствовать грамматические и синтаксические ошибки. Шрифт Times New Roman, кегль – 14, междустрочный интервал – 1,5, выравнивание по ширине, отступ первой строки – 1,25.

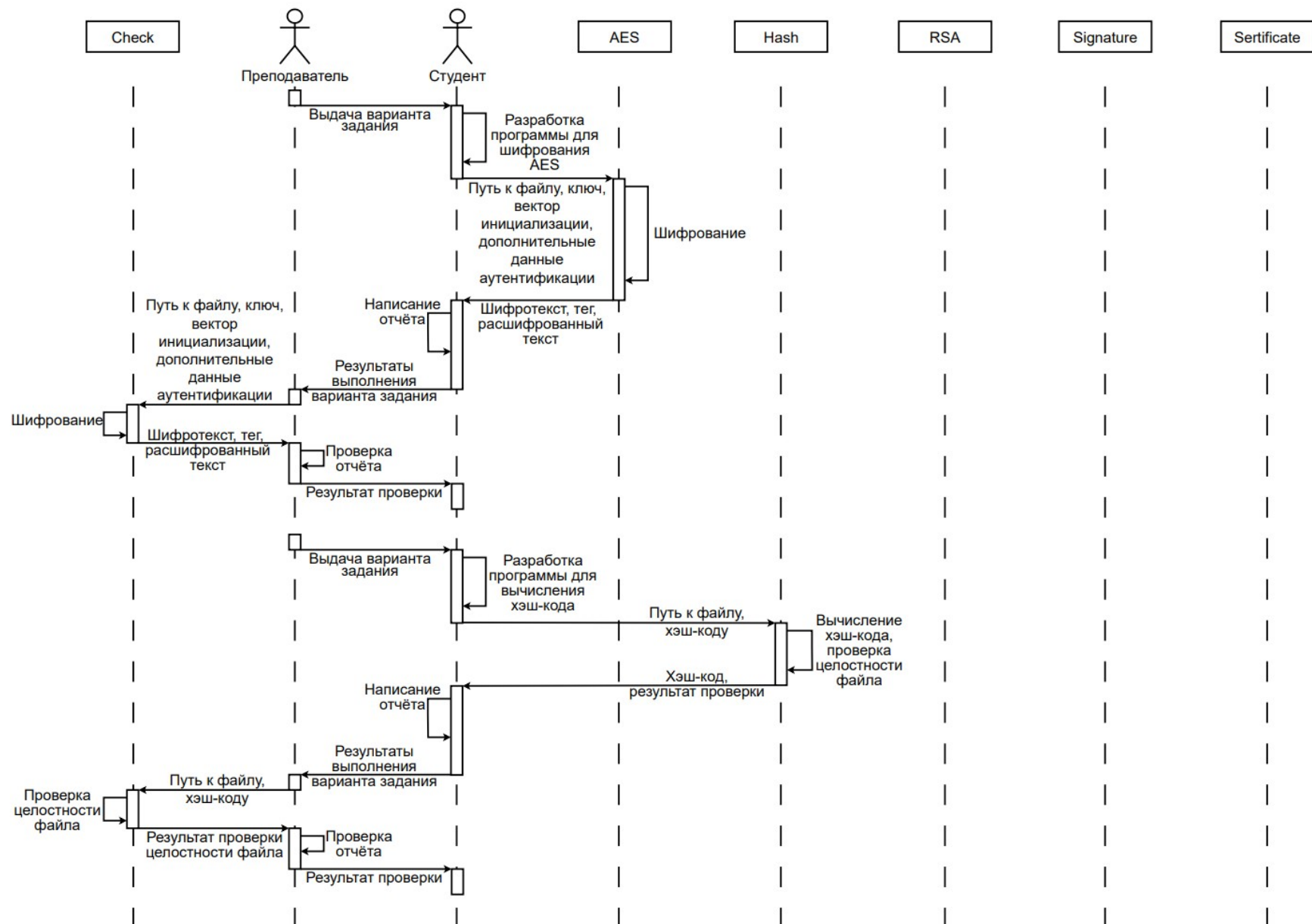


Рисунок 2 – Диаграмма последовательности действий по выполнению практикума

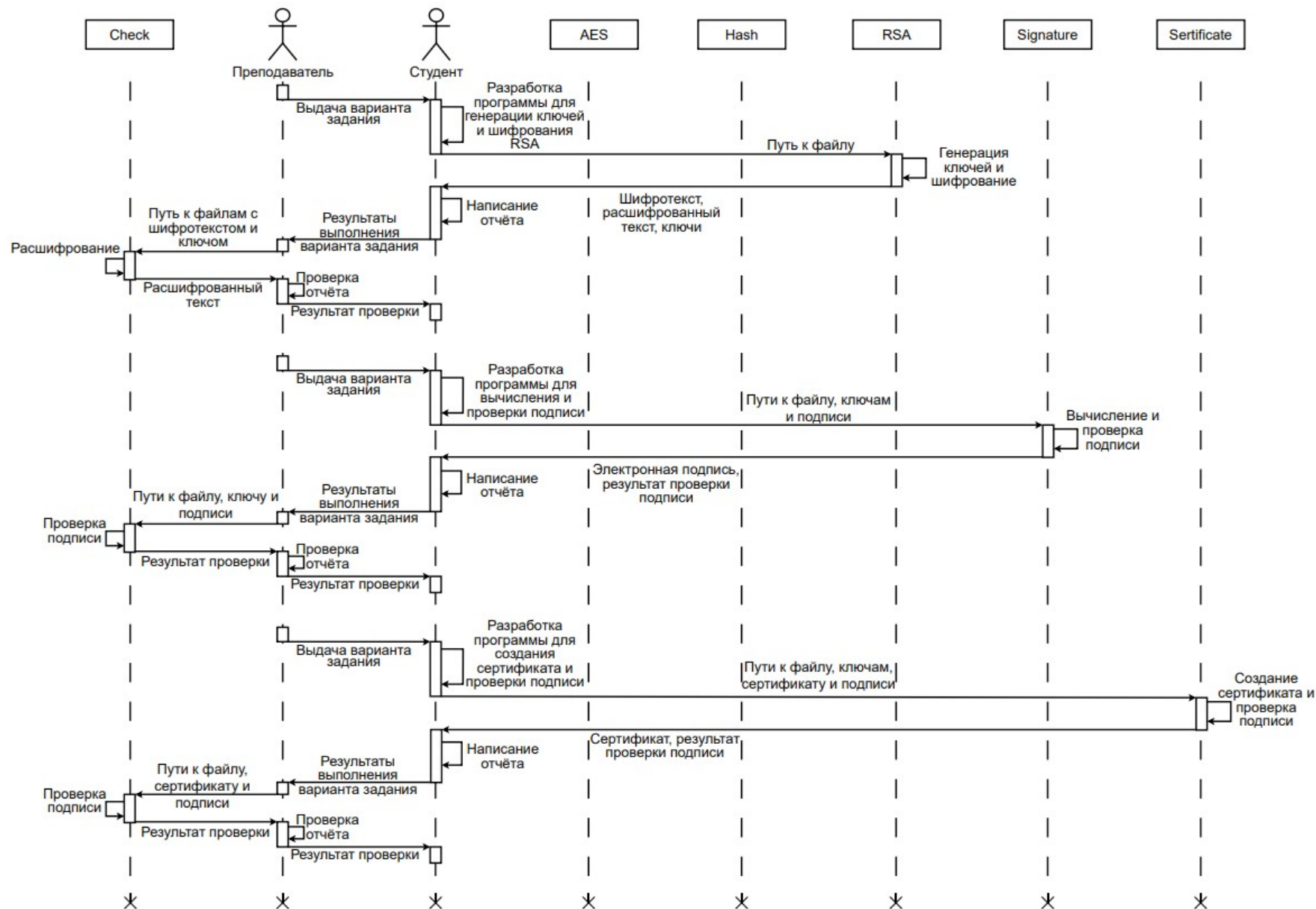


Рисунок 3 – Диаграмма последовательности действий по выполнению практикума

5 Разработка рекомендаций по проведению практикума

5.1 Установка криптографической библиотеки и настройка проекта

Чтобы работать с библиотекой OpenSSL, необходимо установить её на используемое устройство.

Для этого необходимо открыть терминал и выполнить следующую команду, чтобы обновить репозиторий пакетов:

```
sudo apt-get update
```

Затем установить необходимые пакеты, необходимые для сборки OpenSSL и для использования его в Codelite:

```
sudo apt-get install libssl-dev libcrypto++dev libcurl4-openssl-dev
```

Команда «`sudo apt-get install libssl-dev libcrypto++dev libcurl4-openssl-dev`» устанавливает необходимые библиотеки для работы с SSL и HTTPS в Linux:

- `libssl-dev` - библиотека для работы с SSL/TLS протоколами;
- `libcrypto++dev` - библиотека для работы с криптографическими алгоритмами;
- `libcurl4-openssl-dev` - библиотека для работы с протоколами HTTP и HTTPS.

Для установки OpenSSL на Linux необходимо использовать менеджер пакетов установленного дистрибутива. Необходимо выполнить команду:

```
sudo apt-get install openssl
```

Чтобы выполнить настройку проекта необходимо:

- открыть Codelite и создать новый проект;
- настроить компоновщик C++ для проекта, открыв настройки проекта, как показано на рисунке 4;
- в появившемся окне открыть настройки компоновщика "Linker" и настроить опции компоновщика, добавив следующую строку в поле "Linker Options": `-lssl -lcrypto`, как показано на рисунке 5.

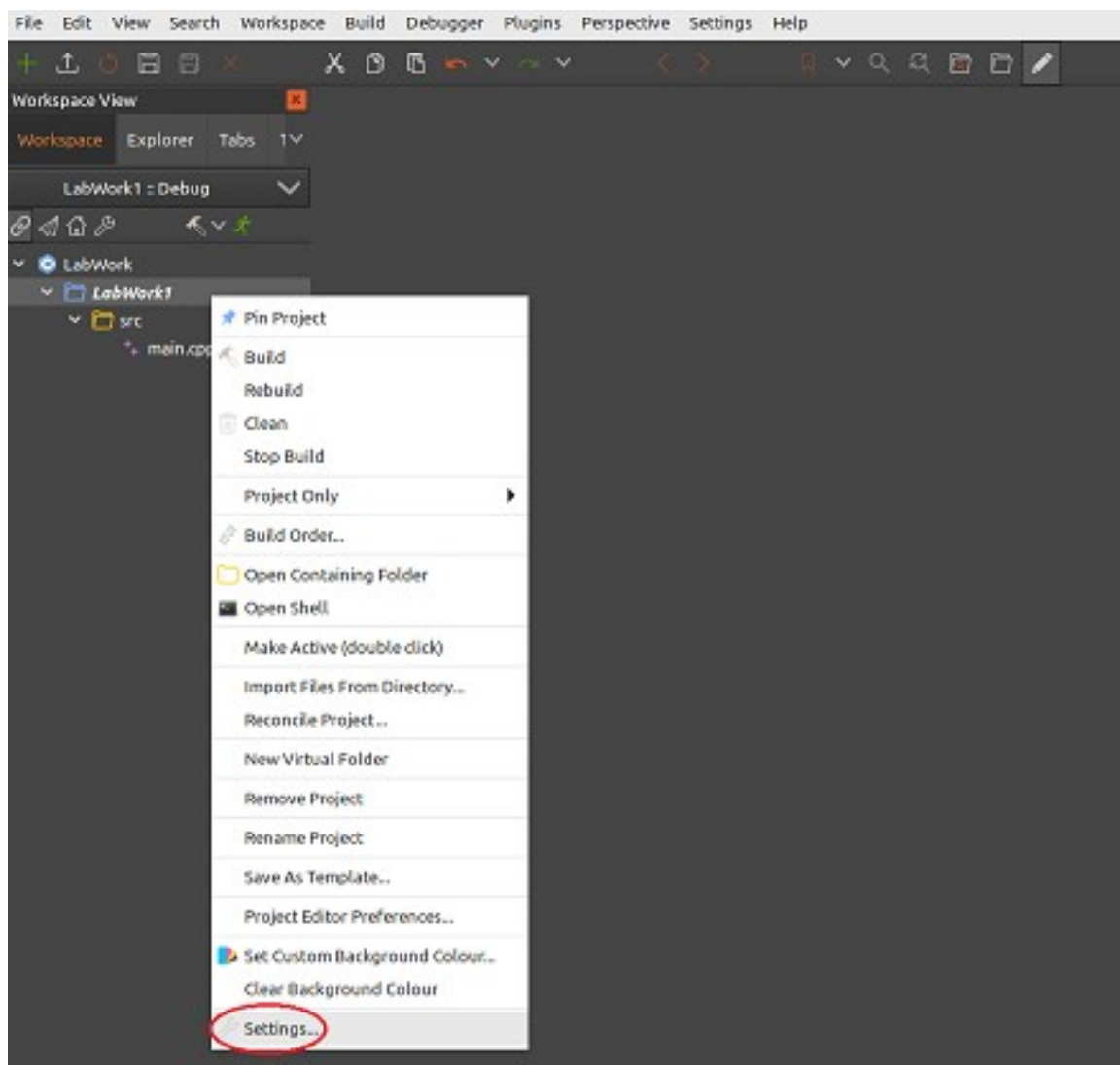


Рисунок 4 – Открытие настройки проекта

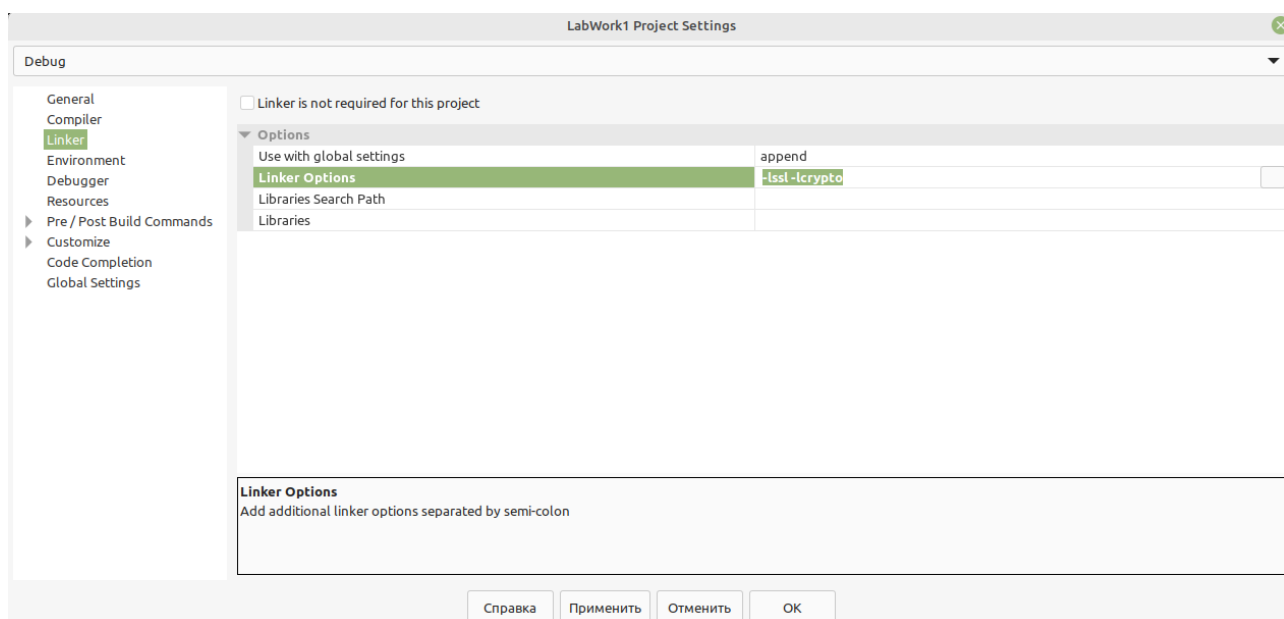


Рисунок 5 – Настройка компоновщика

5.2 Рекомендации для разработки программы шифрования AES-256 GCM

Симметричные алгоритмы используют один ключ как для зашифрования, так и для расшифрования данных. AES-256 GCM – это блочный криптографический алгоритм, используемый для шифрования данных. Он использует блочный шифр AES с длиной ключа 256 бит и вектор инициализации (IV) длиной 16 байт, а также дополнительные данные аутентификации (AAD), которые не шифруются, но используются для аутентификации. В GCM используется функция Галуа, которая комбинирует блоки шифротекста и код аутентификации, чтобы получить тег аутентификации. Для создания зашифрованной информации можно использовать функции OpenSSL.

`EVP_CIPHER_CTX_new()` – это функция, которая создает и инициализирует структуру `EVP_CIPHER_CTX` для зашифрования. Данная функция не требует аргументов и возвращает указатель на структуру `EVP_CIPHER_CTX`. Эта функция должна быть вызвана перед использованием других функций из библиотеки OpenSSL для зашифрования данных.

Пример использования функции `EVP_CIPHER_CTX_new()`:

```
EVP_CIPHER_CTX *ctx = EVP_CIPHER_CTX_new();
```

`EVP_EncryptInit_ex()` – функция, которая инициализирует структуру `EVP_CIPHER_CTX` для зашифрования. Эта функция принимает 5 аргумента: указатель на структуру `EVP_CIPHER_CTX`, указатель на объект типа `EVP_CIPHER` (алгоритм шифрования), указатель на ключ и указатель на инициализирующий вектор. Функция возвращает 1 в случае успеха и 0 в случае ошибки.

Пример использования функции `EVP_EncryptInit_ex()`:

```
EVP_EncryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL, NULL);  
EVP_EncryptInit_ex(ctx, NULL, NULL, key, iv);
```

`EVP_EncryptUpdate()` – функция, которая используется для зашифрования данных. Для установки AAD функция принимает 4 параметра: указатель на

структуру `EVP_CIPHER_CTX`, размер текста, указатель на AAD и её размер. Для зашифрования эта функция принимает 5 параметров: указатель на структуру `EVP_CIPHER_CTX`, указатель на буфер с зашифрованным текстом, указатель на буфер с открытым текстом и размеры обоих буферов. Функция возвращает 1 в случае успеха и 0 в случае ошибки.

Примеры использования функции `EVP_EncryptUpdate()`:

```
EVP_EncryptUpdate(ctx, NULL, &len, aad, aad_len)
EVP_EncryptUpdate(ctx, ciphertext, &outlen, plaintext,
strlen((char *)plaintext))
```

`EVP_EncryptFinal_ex()` – функция, которая завершает зашифрование данных. Эта функция принимает 3 параметра: указатель на структуру `EVP_CIPHER_CTX`, указатель на буфер с зашифрованным текстом и указатель на переменную, содержащую размер зашифрованных данных. Функция возвращает 1 в случае успеха и 0 в случае ошибки.

Пример использования функции `EVP_EncryptFinal_ex()`:

```
EVP_EncryptFinal_ex(ctx, ciphertext + outlen, &final_len);
```

Функция `EVP_CIPHER_CTX_ctrl()` позволяет устанавливать различные параметры, специфичные для шифра. В данном случае функцию можно использовать для получения тега GCM длиной 16 байтов с помощью команды управления `EVP_CTRL_GCM_GET_TAG`. Параметры функции: `ctx` – контекст шифра, команды управления, размера тега, `tag` – указатель на буфер для сохранения тега. Функция возвращает 1 в случае успеха и 0 в случае ошибки.

Пример использования функции `EVP_CIPHER_CTX_ctrl()`:

```
EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16, tag);
```

При завершении работы с зашифрованием необходимо освободить занятые ресурсы и удалить структуру `EVP_CIPHER_CTX` с помощью функции `EVP_CIPHER_CTX_free()`:

```
EVP_CIPHER_CTX_free(ctx);
```

Для расшифрования зашифрованной информации с помощью симметричного алгоритма можно использовать функции OpenSSL. Функции, используемые для расшифрования, работают по такому же принципу, что и

функции для зашифрования.

`EVP_DecryptInit_ex()` – инициализация структуры `EVP_CIPHER_CTX` для расшифрования.

`EVP_DecryptUpdate()` – расшифрование данных.

`EVP_CIPHER_CTX_ctrl()` – функцию можно использовать для установки тега с помощью команды управления `EVP_CTRL_GCM_SET_TAG`.

`EVP_DecryptFinal_ex()` – завершение расшифрования.

При расшифровании необходимо использовать тот же ключ, что и при зашифровании.

5.3 Рекомендации для разработки программы вычисления хэш-функции SHA-256

Для реализации хэш-функции необходимо подключить библиотеку `<openssl/sha.h>`.

```
#include <openssl/sha.h>
```

`SHA256_CTX` – это структура данных, которая содержит состояние алгоритма хеширования SHA-256.

Пример использования объявления структуры `SHA256_CTX`:

```
SHA256_CTX sha256;
```

Функция `SHA256_Init()` используется для инициализации состояния алгоритма хеширования SHA-256. Она принимает указатель на структуру `SHA256_CTX`, которая содержит текущее состояние алгоритма. Функция устанавливает начальное значение для этой структуры.

Параметр функции: указатель на структуру `SHA256_CTX (&sha256)`.

Пример использования функции `SHA256_Init()`:

```
SHA256_Init(&sha256);
```

Функция `SHA256_Update()` используется для обновления состояния алгоритма хеширования SHA-256 и добавления данных к текущему блоку данных. Она принимает указатель на структуру `SHA256_CTX (&sha256)`,

указатель на содержимое файла для хэширования (`kont.c_str()`) и размер содержимого файла (`kont.size()`).

Пример использования функции `SHA256_Update()`:

```
SHA256_Update(&sha256, kont.c_str(), kont.size());
```

Функция `SHA256_Final()` используется для завершения хеширования и вычисления финального хэш-кода для данных, обработанных алгоритмом SHA-256. Она принимает указатель на буфер, в который будет записан финальный хэш-код (`hash`), и указатель на структуру `SHA256_CTX` (`&sha256`).

Пример использования функции `SHA256_Final()`:

```
SHA256_Final(hash, &sha256);
```

Для записи в файл и чтения хэша из файла можно воспользоваться методами `write` и `read`.

Примеры записи в файл и чтения из файла хэша:

```
hashFile.write((char*)hash, SHA256_DIGEST_LENGTH);  
hashFile2.read((char*)hash2, SHA256_DIGEST_LENGTH);
```

5.4 Рекомендации для разработки программы шифрования и генерации ключей RSA

Для создания ключевой пары для асимметричного алгоритма шифрования необходимо подключить библиотеки:

```
#include <openssl/rsa.h>  
#include <openssl/pem.h>
```

Для создания ключевой пары создают объекты `RSA` и `BIGNUM`. `RSA` – это структура, которая содержит открытый и закрытый ключи `RSA`. `BIGNUM` – это структура, которая содержит большие целые числа. Необходимо очистить память после использования.

```
RSA *keypair = RSA_new();  
BIGNUM *e = BN_new();  
BN_free(e);  
RSA_free(keypair);
```

Функция `BN_set_word()` устанавливает значение числа типа `BIGNUM` равным переданному значению типа `unsigned long`. В данном примере функция

устанавливает значение переменной `e` равным константе `RSA_F4`.

Константа `RSA_F4` – это один из возможных публичных экспонентов для генерации ключей RSA. Публичный экспонент – это число, которое используется вместе с модулем для генерации открытого ключа. В данном случае константа `RSA_F4` равна 65537.

Пример использования функции `BN_set_word()`:

```
BN_set_word(e, RSA_F4);
```

Функция `RSA_generate_key_ex()` генерирует пару ключей RSA и сохраняет ее в структуре RSA. Первый параметр функции – указатель на структуру RSA, в которую будет сохранена пара ключей. Второй параметр – размер модуля в битах. Третий параметр – публичный экспонент. Четвертый параметр – колбэк-функция для отображения прогресса генерации ключей (в данном примере `NULL`).

Пример использования функции `RSA_generate_key_ex()`:

```
RSA_generate_key_ex(keypair, 2048, e, NULL);
```

`BIO_new` – создает и возвращает указатель на новую структуру BIO, которая представляет собой буфер в памяти. Необходимо создать два объекта типа BIO для открытого и закрытого ключей. Необходимо очистить память после использования.

Пример использования функции `BIO_new`:

```
BIO *priv = BIO_new(BIO_s_mem());  
BIO *publ = BIO_new(BIO_s_mem());  
BIO_free_all(priv);  
BIO_free_all(publ);
```

`PEM_write_bio_RSAPrivateKey` – записывает закрытый ключ RSA в формате PEM в указанный в качестве параметра буфер в памяти.

PEM (Privacy-Enhanced Mail) формат – это текстовый формат, который используется для хранения данных с конфиденциальной информацией. Он широко используется для хранения и передачи сертификатов, ключей и другой конфиденциальной информации в безопасном формате.

В качестве параметров функция также использует указатель на структуру

RSA, которая представляет собой пару ключей RSA, а также дополнительные параметры, такие как пароль для кодирования ключа.

`PEM_write_bio_RSAPublicKey` – записывает открытый ключ RSA в формате PEM в указанный в качестве параметра буфер в памяти. В качестве параметров функция также использует указатель на структуру RSA, которая представляет собой пару ключей RSA.

Пример использования функций `PEM_write_bio_RSAPrivateKey` и `PEM_write_bio_RSAPublicKey`:

```
PEM_write_bio_RSAPrivateKey(priv, keypair, NULL, NULL, 0, NULL,
NULL);
PEM_write_bio_RSAPublicKey(publ, keypair);
```

Функция `BIO_pending()` проверяет, есть ли доступные, но не считанные данные в буфере, на который указывает указатель на структуру BIO.

Когда данные записываются в BIO в памяти, они сохраняются в буфере, который является внутренним хранилищем BIO. Функция `BIO_pending()` проверяет, есть ли в этом буфере еще непрочитанные данные, которые можно получить с помощью функций чтения данных из BIO.

Функция возвращает количество доступных, но не считанных байтов в буфере данных, связанных с указанным BIO. Если в буфере нет доступных данных, функция возвращает ноль.

Пример использования функции `BIO_pending()`:

```
int priv_len = BIO_pending(priv);
```

`BIO_read()` читает данные из буферов, связанных с BIO для закрытого и открытого ключей. Параметры функции: указатель на структуру BIO, которая содержат буферы в памяти, указатель на буфер, в котором будут записаны прочитанные данные, размеры буферов.

Пример использования функции `BIO_read()`:

```
BIO_read(priv, priv_key, priv_len);
```

Требуется добавить нулевой символ в конце массивов, чтобы преобразовать `char` буферы в строки.

Функция `PEM_write_RSAPrivateKey()` используется для записи закрытого

ключа RSA в файл в формате PEM. В качестве параметров, эта функция принимает: указатель на структуру типа FILE, представляющую файл, в который будет записан закрытый ключ, указатель на структуру типа RSA, которая содержит информацию о ключе RSA, который нужно записать в файл.

Дополнительные параметры:

- enc - указатель на структуру EVP_CIPHER, содержащую информацию о схеме шифрования. Если параметр enc равен NULL, данные в файле будут записаны в открытом виде;

- kstr - парольная фраза, используемая для шифрования ключа. Если параметр равен NULL, ключ будет сохранен в незашифрованном виде;

- klen - длина парольной фразы в байтах;

- cb - указатель на функцию обратного вызова для обработки ошибок. Если параметр cb равен NULL, ошибки будут выводиться на стандартный вывод ошибок stderr;

- u - указатель на пользовательские данные для функции обратного вызова.

Функция PEM_write_RSAPublicKey() используется для записи открытого ключа RSA в файл в формате PEM. В качестве параметров эта функция принимает:

- указатель на структуру типа FILE, представляющую файл, в который будет записан открытый ключ;

- указатель на структуру типа RSA, которая содержит информацию о ключе RSA, который нужно записать в файл.

Пример использования функций PEM_write_RSAPrivateKey() и PEM_write_RSAPublicKey():

```
PEM_write_RSAPrivateKey(private_key_file, keypair, NULL, NULL, 0,
NULL, NULL);
PEM_write_RSAPublicKey(public_key_file, keypair);
```

Прочитать закрытый и открытый ключи можно с помощью функций PEM_read_RSAPrivateKey() и PEM_read_RSAPublicKey(). В качестве параметра

функции принимают указатель на файл, где хранится ключ.

Пример использования функций `PEM_read_RSAPrivateKey()` и `PEM_read_RSAPublicKey()`:

```
RSA *rsapriv = PEM_read_RSAPrivateKey(fp, NULL, NULL, NULL);  
RSA *rsapubl = PEM_read_RSAPublicKey(fp, NULL, NULL, NULL);
```

Функция `RSA_public_encrypt()` используется для шифрования данных с помощью открытого ключа RSA. В качестве параметров эта функция принимает:

- `len` - длину открытого текста в байтах;
- `from` - указатель на массив данных, представляющих открытый текст, который должен быть зашифрован;
- `to` - указатель на выходной буфер, в который будет записан зашифрованный текст;
- `key` - указатель на структуру RSA, содержащую открытый ключ RSA, используемый для шифрования;
- `padding` - параметр, определяющий схему дополнения для RSA шифрования.

Пример использования функции `RSA_public_encrypt()`:

```
RSA_public_encrypt(plaintext.length() + 1, (unsigned char *)  
plaintext.c_str(), encrypted, keypair, RSA_PKCS1_OAEP_PADDING);
```

Функция `RSA_private_decrypt()` используется для расшифрования данных с помощью приватного ключа RSA. В качестве параметров, эта функция принимает:

- `len` - длину зашифрованного текста в байтах;
- `from` - указатель на массив данных, представляющих зашифрованный текст, который должен быть расшифрован;
- `to` - указатель на выходной буфер, в который будет записан расшифрованный текст;
- `key` - указатель на структуру RSA, содержащую приватный ключ RSA, используемый для расшифрования;
- `padding` - параметр, определяющий схему дополнения для RSA

расшифрования.

Пример использования функции `RSA_private_decrypt()`:

```
RSA_private_decrypt(encrypted_length, encrypted, decrypted,  
keypair, RSA_PKCS1_OAEP_PADDING);
```

Для создания случайных ключей можно использовать функцию `RAND_bytes()` из библиотеки OpenSSL. Нужно подключить библиотеку:

```
#include <openssl/rand.h>
```

Пример использования функции `RAND_bytes()`:

```
RAND_bytes(key, sizeof(key));
```

5.5 Рекомендации для разработки программы вычисления и проверки электронной подписи

Функция `EVP_MD_CTX_new()` создает новый контекст хэширования, используя библиотеку OpenSSL. Этот контекст используется для хранения параметров хэширования, таких как тип хэша и промежуточное состояние хэша. После использования функции необходимо освободить память.

Пример использования функции `EVP_MD_CTX_new()`:

```
EVP_MD_CTX *mdctx = EVP_MD_CTX_new();  
EVP_MD_CTX_free(mdctx);
```

Функция `EVP_PKEY_new()` создает новый контекст ключа, который используется для хранения параметров и самих ключей в библиотеке OpenSSL. Возвращаемое значение – указатель на созданный контекст ключа. После использования функции необходимо освободить память.

Пример использования функции `EVP_PKEY_new()`:

```
EVP_PKEY *pkey = EVP_PKEY_new();  
EVP_PKEY_free(pkey);
```

Функция `EVP_PKEY_assign_RSA()` присваивает ключ RSA, хранящийся в структуре, контексту ключа.

Пример использования функции `EVP_PKEY_assign_RSA()`:

```
EVP_PKEY_assign_RSA(pkey, keypair);
```

Функция `EVP_PKEY_CTX_new()` создает новый контекст операции для использования с заданным контекстом ключа. Второй параметр установлен в

NULL, поскольку в этом примере не используется никакой дополнительный контекст операции.

Пример использования функции `EVP_PKEY_new()`:

```
EVP_PKEY_CTX *pctx;  
pctx = EVP_PKEY_CTX_new(pkey, NULL);
```

Функция `EVP_PKEY_sign_init()` инициализирует контекст подписи.

Пример использования функции `EVP_PKEY_sign_init()`:

```
EVP_PKEY_sign_init(pctx);
```

Функция `EVP_PKEY_CTX_set_rsa_padding()` устанавливает тип заполнения RSA для контекста подписи. Первый атрибут – это контекст подписи, а второй – тип заполнения. `RSA_PKCS1_PADDING` – тип заполнения, который обычно используется для подписи данных RSA.

Пример использования функции `EVP_PKEY_CTX_set_rsa_padding()`:

```
EVP_PKEY_CTX_set_rsa_padding(pctx, RSA_PKCS1_PADDING);
```

Функция `EVP_DigestSignInit()` инициализирует контекст хеширования. Она принимает пять атрибутов: контекст хеширования, контекст подписи, который был инициализирован ранее, используемый алгоритм хеширования, дополнительный контекст хеширования и ключ, используемый для подписи.

Пример использования функции `EVP_DigestSignInit()`:

```
EVP_DigestSignInit(mdctx, &pctx, EVP_sha256(), NULL, pkey);
```

Функция `EVP_DigestSignUpdate()` добавляет данные для подписи. Она принимает три атрибута: контекст хеширования, данные, которые следует подписать, и длина данных.

Пример использования функции `EVP_DigestSignUpdate()`:

```
EVP_DigestSignUpdate(mdctx, message.c_str(), message.length());
```

Функция `EVP_DigestSignFinal()` получает финальную подпись. Она принимает три атрибута: контекст хеширования, буфер, в который помещается подпись, и длина полученной подписи.

Пример использования функции `EVP_DigestSignFinal()`:

```
EVP_DigestSignFinal(mdctx, sig, &siglen);
```

Функция `EVP_DigestVerifyInit()` инициализирует контекст проверки подписи. Она принимает пять атрибутов: контекст хеширования, контекст

подписи, который был инициализирован ранее, используемый алгоритм хеширования, дополнительный контекст хеширования и ключ, который используется для проверки подписи.

Пример использования функции `EVP_DigestVerifyInit()`:

```
EVP_DigestVerifyInit(mdctx, &pctx, EVP_sha256(), NULL, pkey);
```

Функция `EVP_DigestVerifyUpdate()` добавляет данные для проверки подписи. Она принимает три атрибута: контекст хеширования, данные, которые должны содержаться в подписи, длина добавляемых данных.

Пример использования функции `EVP_DigestVerifyUpdate()`:

```
EVP_DigestVerifyUpdate(mdctx, message.c_str(), message.length());
```

Функция `EVP_DigestVerifyFinal()` производит финальную проверку подписи. Она принимает три атрибута: контекст хеширования, подпись, которую необходимо проверить, и длина подписи.

Пример использования функции `EVP_DigestVerifyFinal()`:

```
EVP_DigestVerifyFinal(mdctx, sig1, siglen);
```

Записать подпись в файл можно с помощью `*.write()`. В качестве параметров необходимо указать переменную, в которую будет записываться подпись, и длину подписи. Для работы с файлами необходимо подключить библиотеку `<fstream>`.

Пример записи подписи в файл:

```
#include <fstream>
ofstream out("signature.sig");
out.write((char*)sig, siglen);
out.close();
```

Чтобы прочитать подпись из файла, можно использовать библиотеку `<sstream>`. `Stringstream` – класс, который представляет поток данных и позволяет управлять ими как с помощью операторов ввода-вывода (`<<` и `>>`), так и с помощью обычных строк в стиле C.

Экземпляр класса `stringstream` создается с помощью конструктора без параметров. Он решает задачу преобразования потока данных в строку.

`rdbuf()` – метод класса `fstream`, возвращает указатель на объект `basic_streambuf`, который является буфером данного потока.

Метод `str()` класса `stringstream` преобразует поток в строку, которая будет храниться в переменной.

Пример чтение подписи из файла:

```
#include <sstream>
string siglstring;
ifstream Signature("signature.sig", ios::binary);
stringstream buffer;
buffer << Signature.rdbuf();
siglstring = buffer.str();
Signature.close();
```

Чтобы преобразовать строку `string` в массив `unsigned char`, можно использовать библиотеку `<string.h>`. Функция `memcpy()` копирует содержимое строки массив. Первый аргумент – адрес первого элемента массива, второй аргумент – адрес первого элемента строки, третий аргумент – длина строки + 1, таким образом, данные из строки, включая завершающий символ `'\0'`, будут скопированы в массив.

Функция `strcpy()` копирует строку из одной переменной в другую. Аргументы функции – адрес переменной, в которую будет скопирована строка, и адрес переменной, содержащей копируемую строку. Функция будет копировать символы строки в массив, пока не встретит завершающий символ `'\0'`.

Пример преобразования строки в массив:

```
#include <string.h>
memcpy(sig1, siglstring.c_str(), siglstring.size() + 1);
strcpy((char*)sig1, siglstring.c_str());
```

5.6 Рекомендации для разработки программы создания сертификата и проверки электронной подписи

Х.509 является стандартом для формата, который определяет содержимое и формат сертификатов цифровых подписей. Структура сертификата Х.509 определяет формат информации об удостоверяющем центре, владельце сертификата, информации о действительности сертификата и подписи.

Для создания сертификата можно использовать функции `X509_new()`, `ASN1_INTEGER_set()`, `X509_gmtime_adj()` и `X509_set_pubkey()`.

`X509_new()` – это функция для создания нового структурированного объекта `X.509`, который представляет собой сертификат цифровой подписи.

Пример использования функции `X509_new()`:

```
X509 *x509 = X509_new();
```

`ASN1_INTEGER_set()` – это функция для установки значения серийного номера сертификата `X.509`.

`X509_get_serialNumber()` это функция, которая принимает в качестве параметра сертификат `X509` и возвращает объект типа `ASN1_INTEGER`. `ASN1_INTEGER` содержит серийный номер сертификата, который идентифицирует его в контексте конкретного удостоверяющего центра или системы Public Key Infrastructure (PKI).

Пример использования функций `ASN1_INTEGER_set()` и `X509_get_serialNumber()`:

```
ASN1_INTEGER_set(X509_get_serialNumber(x509), 0);
```

`X509_gmtime_adj()` – это функция для установки времени начала и окончания действия сертификата.

`X509_get_notBefore()` – это функция, которая принимает в качестве параметра сертификат `X509` и возвращает объект типа `ASN1_TIME`. `ASN1_TIME` содержит информацию о времени начала действия сертификата.

`X509_get_notAfter()` – это функция, которая принимает в качестве параметра сертификат `X509` и возвращает объект типа `ASN1_TIME`. `ASN1_TIME` содержит информацию о времени окончания действия сертификата.

Пример использования функций `X509_gmtime_adj()`, `X509_get_notBefore()` и `X509_get_notAfter()`:

```
X509_gmtime_adj(X509_get_notBefore(x509), 0);  
X509_gmtime_adj(X509_get_notAfter(x509), (long)60*60*24*365);
```

`X509_set_pubkey()` – это функция библиотеки OpenSSL, которая принимает в качестве параметров указатель на сертификат `X509` и открытый

ключ, который будет установлен внутри сертификата.

Пример использования функции `X509_set_pubkey()`:

```
X509_set_pubkey(x509, pkey);
```

Параметр `pkey` в функции `X509_set_pubkey()` является указателем на структуру ключа `EVP_PKEY`.

Для установки имени владельца сертификата можно использовать функции `X509_NAME_new()`, `X509_NAME_add_entry_by_txt()` и `X509_set_subject_name()`.

`X509_NAME_new()` – функция, создающая новый объект типа `X509_NAME`, который представляет имя владельца X509-сертификата. После окончания работы необходимо очистить память.

Пример использования функции `X509_NAME_new()`:

```
X509_NAME *name = X509_NAME_new();
```

```
X509_NAME_free(name);
```

`X509_NAME_add_entry_by_txt()` – функция, которая добавляет поле в имя владельца X509-сертификата. Она принимает следующие параметры:

- `name` - указатель на объект `X509_NAME`, в который будет добавляться поле владельца;

- `field` - название поля владельца X509-сертификата;

- `type` - тип кодировки поля;

- `value` - значение поля владельца;

- `len` - длина значения поля;

- `loc` - индекс поля, если имеется, или -1.

Пример использования функции `X509_NAME_add_entry_by_txt()`:

```
X509_NAME_add_entry_by_txt(name, "CN", MBSTRING_ASC, (const unsigned char *) "example.com", -1, -1, 0);
```

`X509_set_subject_name()` – функция библиотеки `OpenSSL`, которая устанавливает имя владельца X509-сертификата. Она принимает следующие параметры:

- `x509` - указатель на структуру сертификата `X509`, в которую будет добавлено имя владельца;

– name - указатель на имя владельца X509-сертификата, который будет установлен в соответствующее поле владельца сертификата.

Пример использования функции X509_set_subject_name():

```
X509_set_subject_name(x509, name);
```

Чтобы подписать файл, можно использовать функцию X509_sign().

X509_sign() – функция библиотеки OpenSSL, которая подписывает сертификат X509 цифровой подписью. Она принимает следующие параметры:

– x509 - указатель на структуру сертификата X509, который будет подписан;

– pkey - указатель на объект EVP_PKEY, который содержит закрытый ключ электронной подписи;

– digest - указатель на алгоритм хеширования, который будет использоваться для подписи сертификата.

Пример использования функции X509_sign():

```
X509_sign(x509, pkey, EVP_sha256());
```

Записать сертификат в файл можно с использованием функции PEM_write_X509(). В качестве параметров она принимает указатели на файл и структуру сертификата.

Пример использования функции PEM_write_X509():

```
PEM_write_X509(fp, x509);
```

Прочитать сертификат из файла можно с помощью функции PEM_read_X509(). В качестве параметра она принимает указатель на файл.

Пример использования функции PEM_read_X509():

```
X509 *x509 = PEM_read_X509(fp, NULL, NULL, NULL);
```

Проверить соответствие закрытого ключа сертификату можно с помощью функции X509_check_private_key(). В качестве параметров она принимает указатели на структуру сертификата и структуру ключа EVP_PKEY.

Пример использования функции X509_check_private_key():

```
if (X509_check_private_key(x509, privkey)) {  
    std::cout << "Закрытый ключ соответствует сертификату\n";  
} else {  
    std::cout << "Закрытый ключ не соответствует сертификату\n";  
}
```

}

Чтобы считать открытый ключ из сертификата, можно использовать функцию `X509_get_pubkey()`. Она принимает указатель на структуру сертификата `X509`.

Пример использования функции `X509_get_pubkey()`:

```
EVP_PKEY* pubkey = X509_get_pubkey(x509);.
```

6 Разработка контрольных примеров и вариантов для проведения практикума

6.1 Разработка контрольных примеров для проведения практикума

Для проведения практикума было разработано 3 контрольных примера.

Контрольный пример для проверки работы программы шифрования AES-256 GCM представлен в таблице 6.

Таблица 6 – Контрольный пример для AES-256 GCM

Открытый текст																	
GY3OQeG4ouXmj2m0nEeu3YzkC6RVDKIEi6QIyN7uQ9xBpZ5SgXqDAWJarMAvxFo3OTSHr dULuxYVuZe05BBw4r4Tnhw4ki68guKVOjnnrTcB3XxelAktrbSusGoXbUsPfsYVrphUj89GJOfk zLah7aP2fKr0rVGHBiSZreeTQOFFYmtYYq71Auc18xBfWHuNS9AYH6X3DdvlLxs4VLivIuKp ZWpheIuR9R8GvaKGq96ga1f84VWyXrONhUufClk1q198LJP5m1HLV5fQW6htEFiAtHE2Kn2 TOw6UEebtZe0v5kXffskjFjDTsCbVHLZmqmjfYJNtIXA17Clj8y4Q3x6dUS8AxGw1wUpTcCef qhOlZOZirQFz2AjVUrNYvcsTljy0vCiyyqqgGVbOHVvvUgLpFIS6qtXBnuXivhkJLbnfNtsjniR UXAfOZfPn0ziTk3tYSFDPvuc0hkKam0XOUnU198Tx7T6JFbiW9vLXBUuUs7uv8I42sw0bpU7 on4Nm6BxBDPiWgAxGkGztqxmWhCCsZZ7wONIXWFWQWNY0UrQvu03HmXOyGGlz305L K1g3JBhXA9dVBD1EBwQ6zaEkP053bLTjMZjQ8jlxmKZd19tbisTmew5tl4zpRYxiYNRrX7S CViaZLjhb9H1X94H4uZJ1aGfiQ1GI1lXm478UrMZt2oENSf6JRVwR4S90i6eRDpPaxcfa2chdL HQN9bceh1IypJ6sAYcliddhuKCZ0X0yYjSnBVpLPWFCaQIM3Dk9cmBDMw3aju7epvepoVGaj DJzm2Al6ZmSt3ONWinmKtWUIB1Q567K3sekuBk50f6D11lo99gcy8EEce7ewNaDTwIETYY5 utCwRHGkE7ocluzOB8dHvYrGv5TWiyulc1LVHTO3N0OyfXzQQZGgyo67qAHOishlhLvrBBb G6m99OMDOXFncvl2osQsmMraK8Pdl8wPjhrGEreZv7q4mKXefAg7DnyI1J8GUdoiRFxPijicd CuQyfZMoKB0AmcBaMQ																	
Ключ									Вектор инициализации						AAD		
HDtTfHff4Z3VFPTYpgHcYmaBblQSwtaj									k95U7ctB1pRonE62						Reference_example_ variant1		
Зашифрованный текст																	
0000 - 27 ad 86 97 f6 8d e5 8e 47 07 6a 8a 60 88 a8 73																	
0010 - cd 17 08 b0 fe fc 22 d1 7d 8c ff 06 13 2c 69 6b																	
0020 - 55 1c e5 5a 44 fc 74 ce a3 af 7e 50 7e af 10 9e																	
0030 - 3b a6 2f d2 16 d6 c4 b2 b2 98 c7 d5 24 4b f2 02																	
0040 - 52 d4 6b 9b 9e 4d 12 bb 8d dc 5b 94 af a4 e8 43																	
0050 - fe af 15 c7 be cf 65 bf e0 29 27 eb 8a 87 73 0e																	
0060 - 8f 1a d9 a1 9e 92 3c 29 68 e1 e6 fb a0 e6 58 bc																	
0070 - 0f 2b 39 07 a0 d9 d2 4e 8c 28 55 77 8a 01 2b 01																	
0080 - b7 00 2a 8d 9c 28 22 c1 4b 56 cd 9e fe bf d8 64																	
0090 - f7 1a c1 ba bf 48 6e 11 34 e3 c5 ee 63 7b 61 12																	
00a0 - b0 4b 45 98 ed 35 45 97 d2 57 fc 0d a9 e6 bd 0a																	
00b0 - 18 b7 16 08 46 3b 6c e9 38 0a 85 2d da 45 a6 75																	
00c0 - 3e ac 7b a7 73 fd bc 8c e7 ae 01 4e 04 af e8 44																	
00d0 - a0 1d ce 04 87 c8 84 eb 3b 16 d9 74 62 a6 84 ac																	
00e0 - 21 61 99 d3 76 29 fa 39 59 8b 63 6b 0e 77 3e aa																	
00f0 - 85 79 48 ad e1 ec 5c f9 38 60 36 08 e5 35 5d 75																	

Продолжение таблицы 6

Зашифрованный текст																										
0100	-	96	9f	f2	be	2d	bd	44	13	4e	5a	4d	0d	80	2f	12	a9									
0110	-	b9	51	88	21	1f	77	75	75	88	ba	4f	05	e8	d7	c0	50									
0120	-	dd	44	8d	ca	d0	1f	5b	ad	16	bf	88	59	55	01	fe	6a									
0130	-	85	b7	78	e4	d2	da	50	55	9a	f8	8b	c8	7d	23	b8	d0									
0140	-	dc	94	1f	aa	7d	9a	33	2f	06	93	5a	14	e4	af	62	35									
0150	-	5b	f3	74	c4	6c	4e	b0	b9	16	a5	08	ca	2d	3b	8b	3a									
0160	-	af	4c	5d	a5	c8	09	e6	a3	87	de	f6	81	1b	f3	81	c1									
0170	-	35	bc	54	a2	0d	9f	2e	04	cf	2d	dc	b0	ba	54	0e	3c									
0180	-	e3	1e	b3	18	b2	de	50	f8	69	28	d3	a4	b7	99	78	70									
0190	-	21	2f	0b	bf	36	13	3c	23	ac	fa	14	39	18	2f	34	32									
01a0	-	97	39	c5	95	c1	15	86	bf	8a	53	75	fb	85	4e	fb	5e									
01b0	-	e8	5f	0f	34	be	aa	f6	b2	a3	3a	82	69	64	f5	ef	7c									
01c0	-	9b	af	89	d1	56	7a	8f	1a	c1	3a	c7	a1	eb	d6	7b	ad									
01d0	-	ef	cf	c6	e8	7a	6f	35	ab	dd	a8	64	a3	3b	76	62	2e									
01e0	-	47	7c	68	22	9a	de	b0	10	e7	61	dd	d4	24	ab	56	33									
01f0	-	50	f5	8c	48	3d	6b	46	64	19	18	31	61	5c	80	e4	62									
0200	-	08	f0	e3	a4	5b	5c	de	29	e9	e1	96	ca	49	7e	d5	4c									
0210	-	d1	fa	ce	8c	c9	b0	cb	4e	37	2e	75	4b	5b	71	29	ea									
0220	-	ca	dc	91	05	71	d4	8a	f1	8b	8e	5c	06	eb	c0	70	06									
0230	-	d8	43	38	d1	4e	87	ac	94	d2	c2	4c	8a	53	0b	9f	6c									
0240	-	cb	fc	f9	36	73	aa	ba	5b	1d	a6	e7	62	15	8a	2e	8d									
0250	-	3c	93	67	33	1b	fc	9e	55	4a	3b	56	fb	a1	0e	66	88									
0260	-	e2	68	71	0b	78	e0	88	de	47	18	99	fa	e7	42	28	80									
0270	-	cb	ad	5f	ed	a1	59	c5	fd	7c	cf	69	5c	37	d6	c2	b9									
0280	-	f9	ce	28	b7	23	92	c2	3a	b1	1c	3c	a2	09	86	c5	a6									
0290	-	8a	b8	eb	2a	92	d0	50	44	46	4e	a1	d3	64	38	4d	1a									
02a0	-	62	22	be	56	76	fe	8b	cc	24	2a	c1	21	b6	86	20	25									
02b0	-	cb	1a	97	51	38	3c	10	d1	b6	a1	6a	e6	eb	44	6d	64									
02c0	-	57	e2	cb	57	71	fe	df	db	67	b1	11	13	bf	19	12	cd									
02d0	-	49	15	b0	c1	a7	a8	12	6e	4b	45	49	0c	e4	29	de	7d									
02e0	-	21	cd	48	fd	de	62	1d	e1	76	01	2f	34	d4	e8	60	9d									
02f0	-	22	a8	0f	8e	f8	84	f3	23	08	1d	cd	91	c9	07	02	04									
0300	-	5a	8f	ba	70	a8	a4	ca	65	78	13	39	22	fa	3d	76	bd									
0310	-	9a	79	09	8c	05	c0	8f	5f	a6	4d	33	79	d9	89	10	6b									
0320	-	80	67	c6	56	35	02	cd	d2	dc	e0	f5	1f	b8	cd	c2	4b									
0330	-	96	b6	ae	9f	18	3b	54	37	39	8a	a5	bc	85	0d	df	b0									
0340	-	53	c9	22	0b	f7	4a	bb	c2	18	30	6c	98	49	f7	24	f1									
0350	-	25	34	df	d4	ec	35	4a	fc	23	a3	fb	0e	a4	70	ea	4e									
0360	-	5e	20	be	8c	6c	36	93	20	cd	a6	0d	4e	ad	ae	fc	a1									
0370	-	9b	9d	f2	8c	91	d1	5d	8a	b2	d0	fa	a3	35	81	dc	93									
0380	-	b7	e4	da	4e	c6	71	7c	bf	a7	f6	41	f5	e3	fc	28	5f									
0390	-	9e	c9	cd	77	cc	ff	f8	09	35	48	c6	a1	2d	53	1c	2d									
03a0	-	e1	37	26	34	e8	20	9b	b1	67	08	94	d0	e6	c0	87	ad									
03b0	-	e6	31	e8	da	1c	c2	bb	9b	b8	cb	d7	84	6a	1b	41	f1									
03c0	-	3a	76	e9	8c	b0	a5	28	7e	50	49	21	dc	a4	38	a8	c5									
03d0	-	31	59	83	c0	b5	7e	10	44	67	1d	85	08	dd	80	b1	2d									
03e0	-	bc	3a	e4	c0	18	31	70	75	ab	2b	e0	55	14	f2	c3	dc									
03f0	-	ab	5b	f0	98	28	83	7a	a6	9c	bf	4e	7f	2a	b1	76	ec									

Продолжение таблицы 6

Тег аутентификации	48 fa 4f 4c bb 63 fc 84 6e 9d 90 4b 50 b0 ce dc
--------------------	---

Контрольные примеры для проверки работы программы вычисления хэш-функции SHA-256 представлены в таблице 7.

Таблица 7 – Контрольные примеры для хэш-функции SHA-256

Текст	Хэш
xAGjCFfwGP3Eaxrl2JQroVZA0d9RxWZTKPLYWiLFil0MrcnJdq	43d57dca7ab905646073e19e3c511e4bfbcb66b5ea0b71614510cb20860273d9c
joORgPWYKOT3R2XJwnk28HWEjDici3N7DDVvLE8HP61naCYJk bHRYXGT9xJdUr38nJzjwLcBh3pZWJHQOJvqtAqAcln2cBLbjoxH8 x2FVWsIqvJd9yyYpNvMG7xgULsjhB8LuVuUXurSa5iRVs3oO47w VSN6ChhOhTpHN7lswaAGDqH09BLRInwnPUynfUb3zRX0JqC	28a780b265e476b58641156224bd85e4be7a3e64206f4c64f0a3e62350d89a3f
bKegnXS9FMkRyzww7D6YQ53ktZfYq5lljQQunq4apZV4dK2ItZER mLBFirH4OiSoPolAAZ987InmDjZYuax3Mzg1BTnBYEKhcJSYuaXI VUDmm2lK1q67Sf4CbhTxEPswOFp0wPEo5buX4xv72ypf1t7P3kfP 4fw3rmiRTjDUmI6wraRAYNPCXvsoZCc0xBelbeiJoa2nbGD9go0GE sitFengJeyiwcfcUarUbC4b1PiLbXU5n8GXVFYRNmn1QUVMrY Qmb6vfTin7DZod5YrF036X5yfxF0N22bKGD6ZXNI9IqtFfwZeCn4t an6nmuVniFjP7UzvWFOV1ZuEzDttNVO5Z12CjXEmYBPqbcOCmd tZgoZtaLYFWJKgTbiX1Q09T1nBZhJMsOsWzqXxwiVvlAP2kciVv2 4fiVn0FzBn0bGaKhO4Xm4SS5GYpRXEF1EvHx5OD4sZyHbKyNQ 9aT4fGHDe7YIvviAwSiMxOfFaSe7twjVeLk	6a9a7a5237f5660624dd96fc41027f102f1efcf90904e1fed966c7e76d5b17bc

Контрольный пример для проверки работы программы вычисления электронной подписи представлен в таблице 8.

Таблица 8 – Контрольный пример вычисления электронной подписи

Текст
Dd6bnl6Cl0cz9o4vsX7txjhdGUkve7PyKCfyRgxmnQ5917sFdZtfNWXTmdcJR1xOQunLHKWX tLv4v4wa6iuWW52ANZmTO4jAmdUheBuPdUAKT8aGCu0wzjHLQYNa6ZLGjCZbl6Y7tyCJt6 1xUmByp4QVGXPXYWmJelVocyvqdy6f4vZVHuvPwssmF2F77Df5pCYAajnZz3J3TOup2j9XPrg Kh84kwAs3L2TV4Tl3VFoiRxsbiX3AWD66oi7kZxQXbdfRefZtZdgG3p6BxGPwkO3aEApDl8d QWPRXe2D9zMxnAveqX4EUWa6yRQFJMX3vgY1mkPzl64k62w0yiZe864zV1VoHmuBDf2ip Qof4jA52yjQBdpJNi643yOWw4GQXKI1A60ihlewGX9JqWm3pEMZxpDz5Vy7yJNLViEqPJT2l 1GPcx6PmWNk1L5CZUj82YgV5PX1XuMiXgZx7fN7ME1cFLUldhGSsHga37vTlezDaJpcCUX DjHga9b3hGvZtQK08lqYNpYShAy6X4aQUkFpieeMNHSNq9qTd4Od3GJVUWGXlgoNfbKc0z 8Rh0wJy4UM37uKCK5VjviF9FzquCqzffeE4OpDtDNbJkU8OA5jrTwvCKIEckec8xjtWmpn5JYd Ga4hnXqbOL34YERYRntswfKCCUpdqem8hQT17xqZ90Ppw8FUShaWWMai6PnMNzpY3qKT AtEs4U66VQRnPr2ZG68iR6efi5DMTDjbcZvjxU4KjopYZWVRL9b80pVoPubZlYvd9A1yb5VitF RdReFzSPeBstGVYe6hxlZvTlfr9a45J5V2bqAfuyRVifrcVnAjcN04mwjYbJyzFnUd3iJUqOyZx cjLbwuUjcyZL1znEnxMiqPva1UnpHD7E7Y4Vm4KRJhBY7nB0zl3y2Edq626bbRb8pcNRZwi m4P3zv75pnQ6VtVERpD3Iqt1ZbvKaniN6ELZ4UIUkWoqdW1up8FXkaQIWFCODc3qeBMju7rF lyiOQIePjMMjbfRnBNrNz

Продолжение таблицы 8

Открытый ключ
<p>-----BEGIN RSA PUBLIC KEY-----</p> <p>MIIBCgKCAQEA+ir0Mew+REvyuSlt/jocqRjmUnu08d7t7oJ479UWeXuTWP/XXrOvHkenJ3W7y2FoLxDZY5hRdROtKBaF4Dq7a2dl7ZHTj1wql1k0ihjBP1pC44fVRbtXB9L/kZp9xyxm4rWkHoF7Oo31Vf/A+uYfWlqrMJVIan5PReAdEqSmRBVdKZzuU/NC/2bJhabNmGuRm8wwh00SkvWneff+5wDEaeht6UIZcaLiS+YdRjKLKOOwnkeWqDadel8R0NXxlSF1yUZPVDjRR2Qw9bCUi/t5MPnxb2w4deUrhQ4TglmnzCoRZwPB5qrdQ/EVIF1Nx/6DiB2SIAS4QZ+Mat+SaE/DwIDAQAB</p> <p>-----END RSA PUBLIC KEY-----</p>
Закрытый ключ
<p>-----BEGIN RSA PRIVATE KEY-----</p> <p>MIIEowIBAAKCAQEA+ir0Mew+REvyuSlt/jocqRjmUnu08d7t7oJ479UWeXuTWP/XrOvHkenJ3W7y2FoLxDZY5hRdROtKBaF4Dq7a2dl7ZHTj1wql1k0ihjBP1pC44fVRbtXB9L/kZp9xyxm4rWkHoF7Oo31Vf/A+uYfWlqrMJVIan5PReAdEqSmRBVdKZzuU/NC/2bJhabNmGuRm8wwh00SkvWneff+5wDEaeht6UIZcaLiS+YdRjKLKOOwnkeWqDadel8R0NXxlSF1yUZPVDjRR2Qw9bCUi/t5MPnxb2w4deUrhQ4TglmnzCoRZwPB5qrdQ/EVIF1Nx/6DiB2SIAS4QZ+Mat+SaE/DwIDAQABAoIBAQDUSDbHVizQT5I8PyDAJhfD9fkb7RVyWTWVuulcunlDxPUYBKGbHwButPoopeMhhhqeNbICrXIR4wyAg/w+dhgr36XCnofjktYhb/UBalgxp0YUM81P3pIcb0iyBILwlvP6twp3cmDOgufcQII9ZfGalCpKq6k0w5gx+ofkcIZVsUB/+K3DSE3qGM8BO59MojTB3UMV2MmB1UEpqku6LvFfX/OTWucIxVRxa8WdiiJg4Ln79AIK4yQ9H6cKxitNf8qRlpAbfwheYcMVYylX3P162VYS2eDUul1XnSjNoPsoWP+WLMFO0TOUSXTrNwmGEUMl4IjA/3iPXyFxx3LhAoGBAP7vufGM27h/5Ii8YVpU+GjWMiqOjC3aRShEwM1vm6XV6Y7XU4KVTuySsJqZg2C1T8P1EUpgj+zomrWTFw3WuMSfIPsQb4FAhNgIhJdJQz6WMlkBcAlhJYD2KrlwOF4ZDxR3BuaedmHh5dbEN9kQGCNCeP7BHjHQyy0cmf3UGjAoGBAPs2InWEefXJDNReAT5gN/TVoTHkUqmN9MyOUTMHkw3QcjROr3upZH3zU8S28kn3mib2zCVs2sVvWFufutB+Nkj0Krn7OrlekMf4n2tZ0M4JfeX3yo0DmU66YHFJe/3uz11ZEWSjZcp9BdKgsUSmYZomITSdVjxUxmjCBZ0mlulAoGAMp+8FuFXyF1tLkezXkC6LXZYbRCQmZfyP+x0cBC+Dgvzlcj4EEUxA7e17MFgHpK6cdlc15cUjD3JpNmFNFKxs6G6f+UpzC57G+L4ARhFFCDy2X2bz1Wld4ARHhDJ1StCzZt8CjGCyjaXCISYuRY4M0p9vNVxlWO6lwlal1hZgW0CgYBwBOvly5I277gw2fYUAoPxQXhN9ZCG2y5WKQA8aRxCBLloac4GrrqJD4BDL/FK2oZ66YKPDcNgVw5njQqY/i7myTpkOUKUY3JpjEzLFI/gYIsriRthVkF+wc1Ptxz7VwpQ7e67mcFCUg3Opqllm2vaeTZFaAOUU5IGDX7DXRZoQKBgDaUuER4cvk7YX58buIJTodOGsNB9zErNftZlQ2cxpdFvcRKUDhZUFIgHy5snVf5s9+VkiIISPaze2ffF+8sAaZDisvm9JjJatlyPvnogU5nXqibL5ztbjjFWLSVke7g4E4m0qhUCwNxSoHcR/8p3SKX7YgRw/KP/EAlllovBzXi</p> <p>-----END RSA PRIVATE KEY-----</p>
Электронная подпись (в шестнадцатеричном виде)
<p>E8D362F7742A3C60FA8BBA0A8218A7A0FE8C86B559A9D3F2E386097E961DF683EEC9779B2EAC039EDE732598C87F33EEB33AC84A4D25C7E4496AF9822EC63653B29C32F21FCF6F79C7BD82021E0B41507F47D66B265DD61321BECAD170E19C2EA8101C8243D2A9E9006F75096BFC974E54D125EE9DF34059951C8280B7D3D94E542200FBFFA84BF42B1B880F57D3798AF7D15EF33B051CB28E8D8C3FF8DC2E94F34A7125D9B799253FD74DA848FE6FC4B3B48C078E5D599DD36FF184A3EBE8EE4496E04375B3B5D8D0388BBA61D81393EB2136C0C3EF555B049D850D1F3824787DEF12EB453A1DB8E598B2974EA7030731A5398BDCFB2C009B77A19D87D4B4A3</p>

6.2 Разработка вариантов для проведения практикума

Для проведения практикума было разработано 4 группы по 20 вариантов задания.

Для программы шифрования AES-256 GCM необходимо зашифровать исходный текст, а затем расшифровать полученный шифротекст по вариантам, представленным в таблице 9.

Открытый текст №1:

VmXl3oSPjuZPdrZQ6NcTldhSMY8cnglmdlvF0LHxWHWmAmzdTl4Orm1K2EWBWgGwwa
9SBPUefJf7rGKCBivJAyYxOAzJypDEyK55dl9ubO1lgkxk09Q5AT2BxE473rAAFcL
vzjFIqykbSWQBOSg8R15YzVgZdvHreVVDNGk3slizZ9uCpGYO4DMJKvxgutJVNIBoV
Qr0YcYtmWXjM5PKUbpAyTPuvHxRuJfINjxGpEXmigVMEAIikeXNqEPcoXrCT9K0Oos
wDWfLtVioS9YC11gttJTe5pKdvunPYQc1TAUL8IFanWgfQoU15VZmBYys66akoWs8G
iI1Z7y4xb9ir70tAJqxxmomZ4HUbL21BB3fPBm1Ib5KMv6hqDyMhV5zHo6Gv2YgQrj
JWZpscqqp6jBEQomSlS2CZ45XDVL9U7i7DAvPIfZwUdeopMZR3MdC6ZO981npfHZVG
6GTSdFrC16bM3tItEtc0EJcpWUWFYOMNVdqeZgsm22RmurB0tQ9cDnzobAhZ1GZxDr
o76yQyRhAk9Ae8eOMRHf27ZqIw0YSxVAtuNVY8g2EvGzqCXf8cTaR7yMnrBx1T1CA6
EViTE0UNjoAiAE9S6rAiv658hDvzjDENccyyH20M7xI8QcKzhVbevcWqj3YMRC9Vx1
Rm2JqX5nosRgmumRJBuHvGbIo5oCbqIse10JGX8iZBj17L95CLiDoHM3XNBEd0gJKB
wFfh67W16TVcPRshresIw6MtaNHVmpPGNUo9UNNI2h96e5Uhlh1IAcoufTii28y3a
loVwqrnddo54o1H4McOLiwyMm2ouXmo5hN4QikL8DGHePNXsplcZYxsm0h2ONa2Xm
u56NuQslk0vMtOgqOQxIKxYCC2x8VfGh8ZGbeCXfrHWuGqhjVDBUrx6b2EwAvkdggj
astlAsn1AGlASMWRNXyLO4bzeiLSpSsVmFmlyav5BZfU7RdxbdhjK0JqiFxeEABA3n
PxQZyMwplkpcvHjR0irEihyacEfL02SGNI

Открытый текст №2:

FGMUTMMOot0oyWsF8M3PCTQx6Qs54CzVhRK9XJr8A2WIN8vkwFQKhSmAAg3OAgMsm8
XzCrXgQZN4KjyoirEJ42YFamayTPDE0BPPyELSyFTwVM2g00jcMKIjXSztKnwysnHF
DSLq3xPKaSzQBISs0YsyUJ1uqDQwOJR9gFCkgEXkvXmLkwsczharMaJqyy1BGVLrB
PIT0KJhLJ1ytFA3Bj3SfFFRr21CFv6rro7QZw8LDWnXhdUJTcdqMNFxbuggtwXB17u
Di8aCjltmJkzJM5lNmbSlxN8H4Chydi9OHFkgJOzJ1Uz1COYlqnCfwGvm5W6MU4ZMQ
kHbZL50N66X8Nwuukk3inJp99yyInk97eRrFZybMWA82L5gYvUnLdQjdWYkoA9sVSv
Jp5G0FupfY3LTsL1eCGMEN080j8Ns6OHF1ucMUts2IPUAKUZoXzGYYJju6E5zC6n6U
ofE9GHfQRabhhqH0tHbwlzVOUvOSE5gRWQwEzPakhTNSA1GRjQGQa1T0Ur2UZdbFZJ
gcUgiEnZv7hMrwP3J3mlKm7TuC3swEA70nyCAGCRi5nk4tY7de89LfY7duQSpWLREK
wtcLk5wUrI0KlkwswCcWzVhC1WZF0kk3ReiMfkBEeSFP9Y3CU2iJSn8TNegMBxGAc6
tPVblvt2JLXmCyAsRsKfnPJSpaUA6pr0lmaOnotGGd989Wd9Qtf9555aVT1gyfSC25
SJfDbZkyjqt5cibhhm7yplPJPdzV9sEf2rXPzw002rFEirrM8Sp1X580HQBstIzjUB
jAbT05FkcI9fTeXzXkFNzYKhYf5x20GfVaNq1AC5qDUGPWvSwGmJgWVl00lzDAujzm
2IoTzfn11kwXpIn2ThfcWzjfbCfm0aNTdPVGQbvNFZa43QONUvRJUmQvdZUt7V0xV4
2CkZqiwle1cA6E41G5nLrf6OwpkE7qjCGcpud815hRlEXsLWKWXAoHiTrHDpxPbx3J
QWNTbUmYOJ9PfVubGo5ao9njH6fkOdJH5Q

Таблица 9 – Варианты заданий для программы шифрования AES-256 GCM

Вариант	№ текста	Ключ	Вектор инициализации	AAD
1	1	zeIte9NPZ6j7Qhuw19MfE Qfb6wP3Hpaz	i8pwzIWdMEAJNrOV	Reference_example_ variant1
2	2	nQhFPkH1Nyiz2NaH0ZFK vEK79c26jYFJ	SsywkpuSHHrY5DSD	Reference_example_ variant2
3	1	itHV9NVTyIpKno4pxcWf8 mzf8jBVI9eF	DSta9StZDsubC8NU	Reference_example_ variant3
4	2	mAdzXkqW0BCazb2gKU Nz7AZnzh0EOG7V	Wprwbb53mGn8w3o7	Reference_example_ variant4
5	1	4OaHUpU9MjebunYQUU G3iGWLrgBfXxNC	uXGl6UocatIcpJ5m	Reference_example_ variant5
6	2	qGHQHhNler154wRTio0q 5k0pgs9g9ayu	YSrftp5VliKCfHMP	Reference_example_ variant6
7	1	wixtJ1ChXFb1hkDAXZZli Qr6PUHHlyqg	JsZedZgdo0a6aBF3	Reference_example_ variant7
8	2	cxF3btIMh43AKV1iQkf0Z Fnzi2IRqlsj	EvTbhUULkTxHYttg	Reference_example_ variant8
9	1	odAt6oQnNdrUaF3UtrC DstXhFA2vWhQ	dOLk5UQbh2aQIOHc	Reference_example_ variant9
10	2	kWqL2EhSyF24Pis1dQUB 2AnmmqnslyQ4	HQomrlkRL4CKktTm	Reference_example_ variant10
11	1	aFhDaS9qHKmzLZZZY71 CkkGqfZ2D0sjc	ClhdcOYEBwNDnCFM	Reference_example_ variant11
12	2	T42YB0FxzULv9jUakgXq X3bX4ecQLeqI	YaqwXIhr1sxD44PA	Reference_example_ variant12
13	1	eFW6AU6NRJmhMkJDQP EtYYbjnJM5UZZv	cg3AYsanvOBZpwuA	Reference_example_ variant13
14	2	MIModawFdfnsSHz1rhHM BiCZ8vF11uPY	OkLoKKA1fN855R7i	Reference_example_ variant14
15	1	IkrPtK5EfHGblWVf1VcpL rW47wTrvHbo	1XjU67LwKo5BGmEN	Reference_example_ variant15
16	2	syCYOUoqBKwXOIVigK YImqceH4dEAn55	Q8SDXL4Lmet4zFr6	Reference_example_ variant16
17	1	uxhGHIyJnL9as9ugi3FDY EwzZ4gm4JgC	pwSUN9zFlcI3sfrF	Reference_example_ variant17
18	2	JekkuHQ3bRtyDh7JGbr8x UJQQzWADgoX	xEwX8ZKcxI4Odugg	Reference_example_ variant18
19	1	KKKqQsSWYkedMen27sv GiLHLZNwz5VBI	8hyFSaop92MUUpHB	Reference_example_ variant19

Продолжение таблицы 9

Вариант	№ текста	Ключ	Вектор инициализации	AAD
20	2	Up9npsD8djxk034jyFDSpe kDcbIgSQVY	omeVhg4gkJiy6kW5	Reference_example_ variant20

Для программы вычисления хэш-функции SHA-256 необходимо проверить целостность текста, представленного в таблице 10 по вариантам.

Таблица 10 – Варианты заданий для проверки хэш-функции SHA-256

Вариант	Текст	Хэш
1	9pywSn0QomFUMk00QQURR6OxSkVsMzHe82E iwf6wnEUsEwxR9HIntPjUgYpvtbL9PbZLRLkQ yBIJLpT4S21EqLcokfhxVbsU7qhRMMdBtyjHzw4	355263f376ff5af3607399bb4f9ac 45bbe54daca28104d4bfb3aa6b11 b8a388f
2	twl29SoZdrUhfhdDyr8dgURsiGwWJaKWLvSBi8 AbgEtfS4Ery63PlNtwjSwzwr04d0dYjIB1ECgZJjC oE27182RGRwErkWhB2hX	a4b32628a50168d737d3b1d4f92d b24bd324f48ccd70c909a874b081 d4ccf01c
3	9PVS1drRGMtqLW3pMfxaLHCzSLAc45SE3Xaot 7bJugt0gBsjs0ELUaAqqIxsUbJTCRv2BOXaRDx 3JV8EIXEA89cnwaWag6MyVJkhFky1iGZEcTH6s	c8aa18997659af8a6b416f786bd5a a0d740075d4914cc307fa968d93b 184d712
4	ZBh8kRLzC99sL4BsYi4hwfu5fwBIZ8rIgOh6QyD oRLnsXQkBFnxmntnSJ9qRZ0XLil9urklCAfvvETg jwmUix1RzEb6K8Svi5xfbsg3GQIP14vSqDdhfzWr	abc918ccd35f6a5711a565ea470d8 ba567e4e666185ba057451a8c775 952ee73
5	GH793DhsBSvdl4pPKSsZB0WW5cZ4vUfWk6N5 ykoxgFQBSeBOdv4C6npVragiXjTjJYRDzuihV9m A6eBeHIO3aLEMs0EBA3fIAF2IEkArOA899FCX	b7ea526ffb6cddf6b70394f409ccf2 677a13023341b8388ec70c10c935 6f4c09
6	wvmFwL9esNq0XknTKrnUcDbqfIR5DChpj8yvIA X3jOg8Zz9fxFji5IVXKRbvEQZhjwe30faGsNUON k5YgOcuOMAtLE0ItYx3pisRb50ct1XECVH2NnU	6cc6165776b6f89ea0c8616b1d5d 357012ad9a648aa2e5c36fcd748f7 0df5bbe
7	YJQCTxMtceCbsty5qYiUkYkjyCOKwZTJjg54ug Vwd01kQy1vapqyWXj7eG33W4jdZ5vO	3606b496d44c85936c33e5725d77 4f191e02bd3a559f9d147dd08fb10 9fd741a
8	nHwVvnCnEvBTqeGHkWR9n3xBntE9TdiCeyft55 SK2jHg9NaW94yqpGMK1pVaCsHSLdFGud3TG9 JqlxCvDFzzn7diYJAhs5yYmoraabDXDTCi6cYU	ae1bb93400b8ba86d7aff48c4f287 b9f5f9fcdfe26427559e3b612abdd c3ab6e
9	TyKr1YePxTqLvrq5WU3yXmxaAl1llt9yc2bDaU8j oICVI9isfE9ImqAi0IwOSACCWkAgy60aCyznoUE BjSrbR8reryLP	db78d99909526c91675bd7a14a0c 21c60e40f91b95aa82d01dcdda4fa 2fea6f0
10	Zb5bgzoYPmmb7MEXInC9vU8WsMSzDeKO27m 1PZNsFMCj8HuxPnzgjJF62dFDF9OVGA3w082y CFTJ9xcFuXPA45riH44ucTU	a2afc207a199d233ec5193ee7f120 b923d4799a206f4e9a2f5e3700029 12615d
11	tVFbNmIYpwglcE3j pzLRJvTSsxCswYc4s9uOQN PKbYyZnj6RGZwLA3LLaCcUIIPV00ffBpvRAjmE rIw3nWrGmgTnLgOOXpfpfS	eb8a6408b72cdaed8ff60cbf2c127f d03686d4689c918e0e032dfb74ae b059e8

Продолжение таблицы 10

Вариант	Текст	Хэш
12	jr7RdnJUPnSN2YTmDtqfsRYypVDUoEa15nVYjZ SnvjuVc6PMPupvDY9qCEaGNBmswd4nV4f3kHb xoX9FPIGDA7z9gPepyPUQ	a3242d222498d74d60bdb6ca7ecb 5cfcc17d9d39a7f2a1ffe4b74bbc97 b5cd97
13	NBlTVJEXb7tCwHZqIqmAvGQOs5rLAIqzwA69z 2cupFLRTgJPGjiqtEWXw9	01fa845a4beebabe05825f084d47d b778886d876b9b84b6cf69fd787b 95b64e2
14	4tBEggrpQzSLCNX110PeC1JGmA0txneGgLUyO YDlqip85UZuQgNWcRGr7iqpWiNiBcamlyobnNp H6W92ItMYS	9ac074ad51b6336b342e8f845037 2b506cd4e0c1c179f5f9fe2401bb9 63cd52f
15	hH4NLwzcaflvn8QKSKDaNH2RB9aRKwxnxzaw 3s6YIMR4aCBx1A5tNBkVKDCWeIEi2VtWNkC5 kB7eFxS9csE1Cp03kP0o4KneF66iRN0K3m5rcWq 2S6CCzzhi8Q97J	df3646911cb1def13cc15fced2a22f f984437ce5586794f109bf1bd9b03 66e28
16	zUk3e6Jf5KKCzgrNTMji4sQNOBKgYdo7EGxDjl j3RU1Z2JiCYONdOPPkFZxduwulv4RwXuyKp30 GDHd2gJ06W17FTUNoeGU6rZr	e11a421017615c367c5052b8082d 51d7523302a8e3279b514348a88c cce50299
17	dr3EiWuumac90gSvF8G1iZH5ni4TwMu5loDre6H dilNZG0CjsSjgOQ0hE5KI	d2b652f0962c211a5ade333abfffe0 4e8832da96a487ac71f903006ae94 0dc4a
18	QkRfWMqHPrCiERxx4EbmirEcfWfD9J4WIPtKu wwFiMDGW0SXcVcXtPjbWx2RkxllbqlYJ6kY49 paycBial0yXDTUfCcPs9SSte96WzHL	6afe95c53ab19a75254a593c1e71f 8e836e66017312b0de7050880674 c99e66c
19	OVweDtnaDd4PLCiNhWu5neKnMsJAqmy3k8eqd c39WEmQYVlixmxWu1Hf	25e90a5e261078bcb5422e4f3b1a 6ce1020e8d89a16f614e22bd023ee 2f58a2e
20	SL5sbEcSlVsUjvDTZzqOO7uOMTr6rBB47QVW MnYo	242948bc81cbc1e5171238cac25b 09d702771c7474de621d0cdaf0a6 ebfafbcd

Для программы шифрования RSA необходимо сгенерировать ключи, зашифровать исходный текст, представленный в таблице 11, а затем расшифровать полученный шифротекст по вариантам.

Таблица 11 – Варианты заданий для шифрования RSA

Варианты	Размер ключа	Открытый текст
1	2048	hYseiInf5iqftKz2mVSgZwQQyuVIlhzetv1h1LvrWuR5b6GIi8dZSPcM2y4Wq hryOiRRfHnS9vDRYOgtXY5r2ngzSGf73MtDZgTcoNrN8QoN6lcxz6GYRz xxl0YDLqcZgR5gCop8dt9JQYETE4so5SMk5J72YWIhUPVWfK54psHU7 mg3W6aiSoAjVgYtBwIMDkvBIZgJqoX6R4ysJ0Wwq9

Продолжение таблицы 11

Варианты	Размер ключа	Открытый текст
2	1024	GzOBFJqbptMNzxQHqW0RWKkpGyNoFEsKncEelZS9Eli94Czwt7nFFPQsNAahRcEj000JabYPyGw3Bg30iA6Rh
3	2048	adKkQGAIXtoGBzloTwKcJwXEXzRApsLiy1GFkMqFHJn1loJm5R2WurDSkqhHDZK23XkDOrJiyNYPw2dm8u7U1wb05Gp9OabDjj4r1CBZwnraWN1M4v7UCmMPYzFavZBOBUs7SsZpsHsNkr6ZkODqZs5DQtFH5e6A2ztKD NjN4yIL0zULAgx212PFwGsKeOkOYzjMcXju14ERP3PcMfBwV
4	1024	mdmWhNj23DRLFWl3J84P3SSl54MhsuIM2UBqfIfKnS1exz6UvFNizkfSs9DRmo1cProXWQYiPKLTTH4ftdojR
5	2048	JWgAe0TP3kK1ZSIMJOuGFferxXCdjVAYTtaun70vYBKj6brFTgcUAEJE8izWeXjS0ejYAviZUaFd8sAcrHk97kWyPYEnhMIveIwS81OkHzbK2PP6pWYVRtGP0PCrznPVM47zkRGNaqc96WvFWoKJ1xhG2z2fGVnKIpoNLICre93o3VkmblHWuJskOSWpXBBTja6wqaIVnQhQ4Bhw8SK6E
6	1024	cMoCw8QvIGiRVCAVoWcKstF5f5KMqZo3wCuPI1PiJKsk33ZcSVX7vQod80Jf5hzI5MOvrrzgSwN9BXeyXWwLN
7	2048	fr1tZBzDavSQnKLKiEPybOUwazs37agvIe9acdtqlhRS0nkb6ljPCRlljen5o6a08qAMFI5BMmotRrONHz2sm8bCyVcL0AP65qlFXX1HfMjtHZwNnWJmlpqKOb5ldLIwRyXOtZmrRwPoeB9WZBeTaTwmcUdS7NX46SqGyBdf5csMXUBvM1ZgEnUot9ksPjqM9U7vR9KTpER1Ebcn3K3D
8	1024	Srzmp5PpBiAcO35L6JrhKGLdxwUaqHtJoQq3DoFj7WvrfTg3umPhTgDpmZLjz0k4jrWpgk3witxkQfEVrVXSz
9	2048	OBnGAlCECrRsHjDzbOQ66wQrL7HiK64TZVIDJ8IyxkJkVX1Me8QvvrrZ1JE5dFmZjfq3DooiMln5WZTSzKII70VLWg81eGZqpVRDyUsE2vw79LIaxtSzRHRWOgCmoLKzNbY7iHlbuESeNGLaKiZQITHyU2grzo3JBeBR0efpKuuqx5KYtD4pwJA98X8Q8KvLfe1DJhfbskIB2mHQBiad
10	1024	I4pbMNKxw8QgHpATV44YwOtslwAiYNGmO2e8QMxhcr42Cq7BkKQs6AAooKZoV8aoKQizuKEq6GLIMViQNvr4a
11	2048	T0BzGMU42j2fWYVgZaGdVGezVgALNpvxI5k4ppUQORiqsgKkulDFqcnmSkIIteqa9fp28lBRyhYwt7QGvniGhalbzD1VIbgMKhrXKBgLD1tJAWWeug1gOTQWQMYTlGNnOu9jsmOOHkSClLeNeKA2bPwkBGBCZPKkJeu6xHTPvqHQj8CSvE4QJpG3ftJjsBZwgcMz15jgT7wunb4u01OkJ
12	1024	wf2cpAwJBR41M05msO6ktExMl2HnLwZqvIYm36cKBHSao6X9lRXBu6GLAHgEgoilqQqRYtQZbXIW2vzNdn4of
13	2048	Vd6L4b2KuZcCsmHWJfeXhB16h1b90V6wfsjzhsegQzo4JaTiJiMjRL7A1sxXo9kMRcuWYDr6zYR1RcWAX5DAyW1LjzUvNm7z6iVqxoBeZtDiB8poU0Dd5UwXL6vqdeg1yM4fbrEI3H6wEqOMaMneTqHC1PLuAcxpHbUYOCsZKMGoBis352ItOvQmX7V3GZsSEjrvTQbW4ISRZzyymUgx
14	1024	61XKUMhTQdzscSIdC0Eypm599lpxbUWhUVxqFkES0TTWUAf0cqhmUIp8HwV7NXFeAlIEOyvYIRJ9p9itFcGAd

Продолжение таблицы 11

Варианты	Размер ключа	Открытый текст
15	2048	2ItJ8nowVG2XxTgP3bpyhfpUkcOJZRpFMIn2ScK8khTpkZpsbYpRUH5SdJzRpaLgRotazzxOcVQz59su8MA7j6vMMdmwA09nrtXJG6Wr2G923LUxw8ymeyP5IKU9gGHpLa8kgt9mUQ32DyaOKZJ69oLOfPFBWjdWnXfhoM7WH8RB7MARlftSdDbFmip4BhCyn4JFruVZWfXRIaCJB4FTZ
16	1024	TRP7gXnEeMT6fatFSdiO9MECO5sSoGrkOCBgHZajBHSt5HXvXqTnSdsvidjdTBN85fDfH3U2MMXNVD3IPRZG0
17	2048	i8rvAnNzhkh4MSbZrWIIdeHyC0G0cyQR06dNAJAsBpSQIV88yb4fZnvoI1I9S71f88EncwZdv3YHWAqZVln8PoFznXuYBBqj7JG6Yo5Dj9a6f3Ehia4U7MkHlr2Mrka2SoP0hUT0tNBiy1cTGQSOMIVJDiBXS0aidvfWegTYHStxOf7wa6B3Gom5ehk0RSPnFm9h7p2DCSqTRldpL1zg
18	1024	hBifDsU3kK1efXLUJdZnjDnVG99VMCTsmfKT3lyTJr10YETOzFn7ydzrux805Dy10wtkJ82WFHFDaZ73e37Q
19	2048	AwS3oPG9ZYyD8ByHswducKQJBkWYfizBsujRTFGQIILDsn807T2nNu7a7EHihISOpKrR89zf51TR8pVY0L3CGZeQaY719dOnmCXECETVl8OwYFSmzPYmKvDc8rHwyIvJDENWmAVxqjX4iSRdYat39KmgvX79CdduG3zXGUjLPUEkYCsVJqnrMMksn5TjpCD6hg5BQHmBWG2iN6lk5WC
20	1024	udGZqCdURnJX2fBdvkiYCEGIaYOsHEIEQ7YWle3u09bOTFFa8JtBzRn7eQlKTCjaPpamIv0EYZ1yFuWVzQpdt

Для программы вычисления электронной подписи необходимо создать и проверить электронную подпись текста, представленного в таблице 12, по вариантам.

Для программы работы с сертификатами необходимо создать сертификат и проверить электронную подпись текста, представленного в таблице 12, по вариантам.

Открытый ключ:

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEAr2WLDzCx3sgyosuj9+ZGRTjQiB6l6EVdLKYgI+A5nyYQBesEsq11
JWz1xaMl5IuQ3Gp/xX3ed3XUbm49aat8r4/m97722TIpVgbb6RUZtSuZRQ0Oraqo
XJptRh/t6QxNEbisL5bGMIMoso7pZslUofVzstYN4ndXB0eSfBBKwRXk3mliUTKi
F5o4KlFPY5cXhxH6TM7BmC6PqsqYGmwNcr083Vdggq/3zTNftmn1/ZE2jAHGjgz/
bfHhO1E8SNevzgywonoDirRQtRKtcx/SxauHA56RkaTBbFi8/OBXNkLCUtXF6FJs
+J5CCT7hvp3k+CSSoowaIcTCZ6+oZko31QIDAQAB
-----END RSA PUBLIC KEY-----
```

Закрытый ключ:

```
-----BEGIN RSA PRIVATE KEY-----
MIIEpQIBAAKCAQEAr2WLDzCx3sgyosuj9+ZGRTjQiB6l6EVdLKYgI+A5nyYQBesE
sq11JWz1xaMl5IuQ3Gp/xX3ed3XUbm49aat8r4/m97722TIpVgbb6RUZtSuZRQ0O
raqoXJptRh/t6QxNEbisL5bGMIMoso7pZslUofVzstYN4ndXB0eSfBBKwRXk3mli
```

UTKiF5o4KlFPY5cXhXh6TM7BmC6PqsqYGmwNCr083Vdgb/3zTNftmn1/ZE2jAHG
 jgz/bfHH01E8SnevzgywonoDIrRQtRKtcx/SxauHA56RkaTBbFi8/OBXNkLCUtXF
 6FJs+J5CCT7hvp3k+CSSoowaIcTCZ6+oZkO31QIDAQABAoIBAGKRcCqjNPJ8E2jF
 hGomMMtp+Fbm21ei04/p9Bn1QqybmvMaD9PSAw17P01oOD1N6aTU/OUhrUn5H93d
 sUmXI8UN28DSwqqgs/YyeJduUjosPEA/7EXQqDkSV+KWwv66+PzCiZFc7BgLPW3L
 PoOTFUrHQ7MNK5RfeLsQj9PE3B3Rn9gwfLleXp5PDNuvKFM+NT/LbUFE1FyNeQDM
 +poNyetGzTmf+ZEO8R0bZyxzLkNTib9KQYTB6gjc4zGJyEhwe8Xg0OvoCtoi6Bri
 qSlSABUqv35cq1bJy7kzRYDY9PlaOhhXWlmyuWNrXLF/0/6fYPaSmV2U900kvOxh
 iCPM7EECgYEA1BnTW3RDz1Cdnjn+UY29fyCuY7HJ8J1SGh0IwOqPF6Pj754VX04b
 t6FojNFyXzyLQwxFDIfcZWDpWbsApD2CNCtIVc8sIOZ3hmvOjnbjDDqWVMYDAN6k
 5vZD0mmEJ51re5E/34zJjMA5U+jJTQgnYqZ/YhA7u8/jSLMvEuq11SkCgYEA07Lx
 RXMT46mIA3KY1WuYcsjJYFQUXG+Msbt64/Dv4A4jExtwTboO+RA/2LFD0+M5C+B3
 HXvflIcGd2zmuEDESSE/1o88FsHFMSyYb0EAR4sFe42PsDRKld0AyCBZGdpEluG3
 Wz/uq6gsNd4WwrAGWQRsn8Ui6AP952jcIw/yIs0CgYEAydJ9PKPINZSeNsZmSgH+
 k/Sbfr/ZMTkjV7fS8YcZOWZ46TRaj+KKG1TaYeyYcsjyNu2A8qiAvnLV+M0pkh75
 O10IKqbFEBfoeh8xYHyeFa/WBoDIE1bIQQgKjbhbyTpQ4TbY0YcE7gqnQkHa8pLU
 r11REuXxgQgS9MWUAIMYXOkCgYEAmtZtxaymWWMJ7CzZDcTkPVax38SaQ9f7xHoC
 U8wBX2/71I1MNwTMZ1dqTeLlKEX/8eePbchD2c811eQkCv/c0JmSIS/i9jlhBNjY
 rdQmZqDn4+atMoJ15IZhVXWL5tkAQQ61DBZ2HsgEzyuf32FUBSLb9rnPLIoPDZbn
 k3ysPpUCgYEAqkWPUBnNEAo0shqqFt6vefVKoAudxxs4mPQaeu8SaxUTIozFkHXk
 FlGuUEv/XomU6LV5usNqScA77MDqQTdtZftT+186X9wH3EUuGPicCImySxqr9XLQ
 WD/CO1DfQY1jpDeFr9h84kxOG5G3zuqTdGECEJ4CtvUQQu1IxWfCh7Y=
 -----END RSA PRIVATE KEY-----

Таблица 12 – Варианты заданий для вычисления и проверки электронной подписи и работы с сертификатами

Вариант	Текст
1	hYseiInf5iqftKz2mVSgZwQQyuVlhzetvhl1LvrWuR5b6Gli8dZSPcM2y4WqhryOiRR fHnS9vDRYOGtXY5r2ngzSGf73MtDZgTcoNrN8QoN6lcxz6GYRzxxl0YDLqcZgR5 gCop8dt9JQYETE4so5SMk5J72YWlhUPVWfK54psHU7mg3W6aiSoAjVgYtBwIM DkvBIZgJqoX6R4ysJ0Wwq9
2	GzOBfJqbptMNzxQHqW0RWKkpGyNoFEsKncEelZS9Eli94Czwt7nFFPQsNAahRc Ej000JabYPyGw3Bg30iA6Rh
3	adKkQGAIXtoGBzloTwKcJwXEXzRApsLiy1GFkMqFHJn1loJm5R2WurDSkqhHD ZK23XkDOrJiyNYPw2dm8u7U1wb05Gp9OabDjj4r1CBZwnraWN1M4v7UCmMPY zFavZBOBUs7SsZpsHsNkr6ZkODqZs5DQtFH5e6A2ztkDNjN4yIL0zULAgx212PF wGsKeOkOYzjMcXju14ERP3PcMfBwV
4	mdmWhNj23DRLFwl3J84P3SSI54MhsuIM2UBqfIfKnS1exz6UvFNizkfSs9DRmo1c ProXWQYiPKLTTH4ftdojR
5	JWgAe0TP3kK1ZSIMJOuGFferxXCdjVAYTtaun70vYBKj6brFTgcUAEJE8izWeXjS 0ejYAviZUaFd8sAcrHk97kWyPYEnhMIveIwS81OkHzbK2PP6pWYVVRtGP0PCrznP VM47zkRGNaqc96WvFWoKJ1xhG2z2fGVnKIpoNLIcCrc93o3VkmblHWuJskOSWp XBBTja6wqaIVnQhQ4Bhw8SK6E
6	cMoCw8QvIGiRVCAVoWcKstF5f5KMqZo3wCuPI1PiJKsk33ZcSVX7vQod80Jf5hzl 5MOvrrzgSwN9BXeyXWwLN

Продолжение таблицы 12

Вариант	Текст
7	fr1tZBzDavSQnKLKiEPybOUwazs37agvIe9acdqlhRS0nkb6ljPCRlljen5o6a08qAMF I5BMmotRrONHz2sm8bCyVcL0AP65qlFXX1HfMjtHnX46SqGyBDf5csMXUBvM 1ZgEnUot9ksPjqM9U7vR9KTpER1Ebcn3K3D
8	Srzmp5PpBiAcO35L6JrhKGLdxwUaqHtJoQq3DoFj7WvrfTg3umPhTgDpmZLjz0k4j rWpgk3witxkQfEVrVXSz
9	OBnGAICECrRsHjDzbOQ66wQrL7HiK64TZVIDJ8IyxkJkVX1Me8QvvrZ1JE5dFm Zjfq3DooiMln5WZTSzKII70VLWg81eGZqpVRDyUsE2vw79LIaxtSzRHrWOgCmo LKzNbY7iHlbuESeNGLaKiZQITHyU2grzo3JBeBR0efpKuuqx5KYtD4pwJA98X8 Q8KvLfe1DJhfjbskIB2mHQBiad
10	I4pbMNKxw8QgHpATV44YwOtslwAiYNGmO2e8QMxhcr42Cq7BkKQs6AAooKZ oV8aoKQizuKEq6GLIMViQNvr4a
11	T0BzGMU42j2fWYVgZaGdVGezVgALNpvxI5k4ppUQORiqsgKkulDFqcnmSkllTeq a9fp28lBRyhYwt7QGvniGhalbzD1VlbgMKhrXKBgLD1tJAWWeug1gOTQWQMYT lGNnOu9jsmOOHkSCILeNeKA2bPwkBGBCPKkJeuxHTP
12	wf2cpAwJBR41M05msO6ktExMl2HnLwZqvIYm36cKBHSao6X9lRXBu6GLAHgEg oilqQqRYtQZbXIW2vzNdn4of
13	Vd6L4b2KuZcCsmHWJfeXhB16h1b90V6wfsjzhsegQzo4JaTiJiMjRL7A1sxXo9kMR cuWYDr6zYR1RcWAX5DAyW1LjzUvNm7z6iVqxoBeZtDiB8poU0Dd5UwXL6vqd eglyM4fbrEl3H6wEqOMaMneTqHC1PLuAcxpHbUYOCsZKMGoBis352ItOvQmX7 V3GZsSEjrvtTQbW4ISRZzyymUgx
14	61XKUMhTQdzscIdC0Eypm599lpxbUWhUVxqFkES0TTWUAf0cqhMUlp8HwV7 NXFeAlIEOyvYIRJ9p9itFcGAd
15	2ItJ8nowVG2XxTgP3bpyhfpUkcOJZRpFM9nrtXJG6Wr2G923LUxw8ymeyP5IKU9g GHPLa8kgt9mUQ32DyaOKZJ69oLOfPFBWjdWnXfhoM7WH8RB7MARlfTSdDbF mip4BhCyn4JFruVZWfXRIaCJB4FTZ
16	TRP7gXnEeMT6fatFSdiO9MECO5sSoGrkOCBgHZajBHSt5HXvXqTnSdsvidjdTBN 85fdFH3U2MMXNVD3IPRZG0
17	i8rvAnNzhkhn4MSbZrWIIdHyC0G0cyQR06dNAJAsBpSQIV88yb4fZnvoI1I9S71f8 8EncwZdv3YHWAqZVln8PoFznXuYBBqj7JG6Yo5Dj9a6f3Ehia4U7MkHlr2Mrka2S oP0hUT0tNBiy1cTGQSOMIVJDIBXS0aidvfWegTYHStxOf7wa6B3Gom5ehk0RSPn Fm9h7p2DCSqTRldpL1zg
18	hBifDsU3kK1efXLUJdZnjDnVG99VMCTsmfKT3lyTJr10YETOzFn7ydzrux805Dy1 0wtkJ82WFHFDaZ73e37Q
19	AwS3oPG9ZYyD8ByHswducKQJBkWYfizBsujRTFGQIILDsn807T2nNu7a7EHihS OpKrR89zf5lTR8pVY0L3CGZeQaY719dOnmCXECETvI8OwYFSmzPYmKvDc8rH wyIvJDENWmAVxqjuX4iSRdYat39KmgvX79CdduG3zXGUjLPUEkYCsvoJqnrMM ksn5TjpCD6hg5BQHmBWG2iN6lk5WC
20	udGZqCdURnJX2fBdvkiYCEGIaYOsHEIEQ7YWle3u09bOTFFa8JtBzRn7eQlkTCja PpamIv0EYZ1yFuWVzQpdt

7 Разработка примеров реализации практикума

7.1 Алгоритмы реализации функций по применению криптографической библиотеки

Была выполнена разработка примеров реализации практикума. Коды программ представлены в приложении А.

Алгоритмы программы шифрования AES-256 GCM представлены на рисунках 6 – 7.

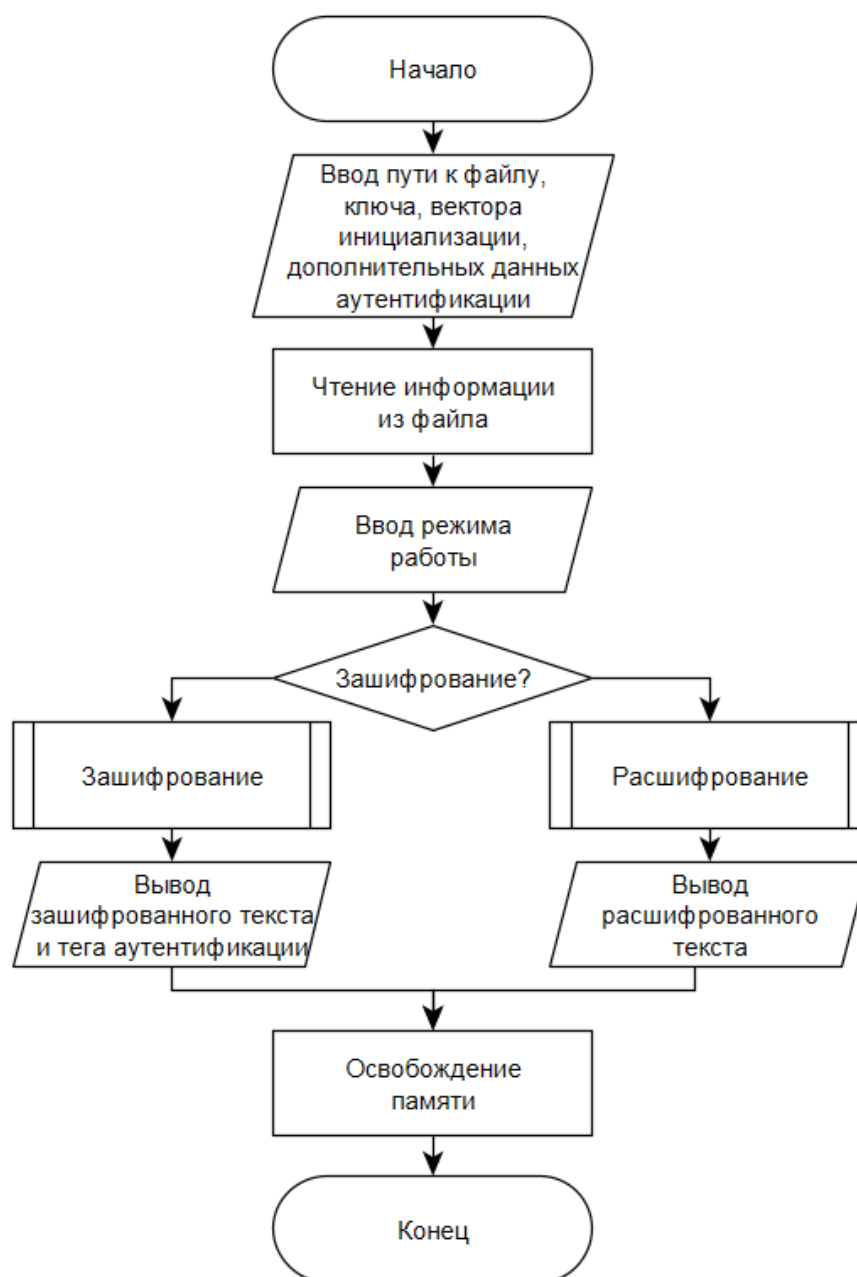


Рисунок 6 – Алгоритм программы шифрования AES-256 GCM

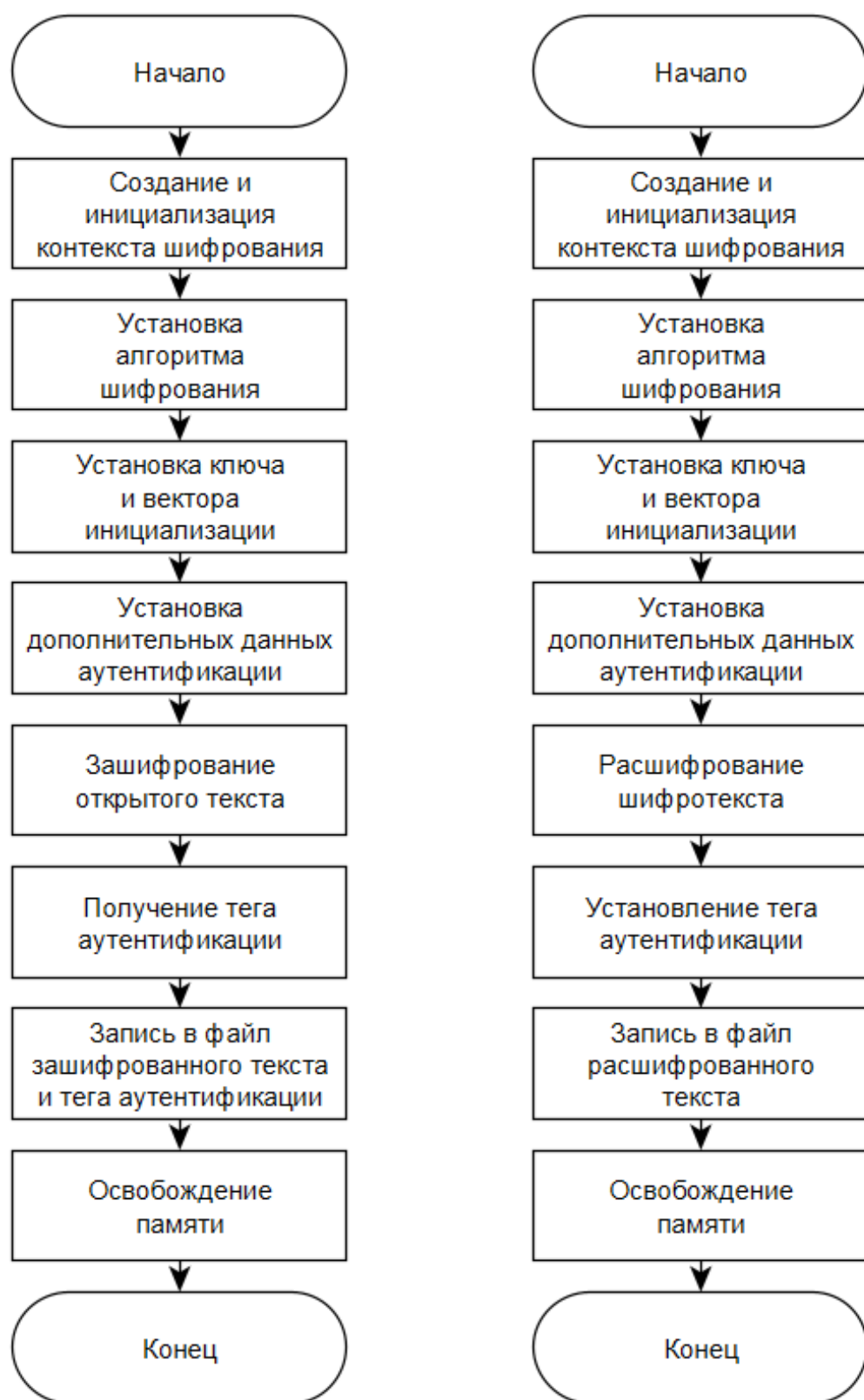


Рисунок 7 – Алгоритмы функций «Зашифрование» и «Расшифрование» соответственно

Алгоритмы программы вычисления хэш-функции SHA-256 представлены на рисунках 8 – 10.

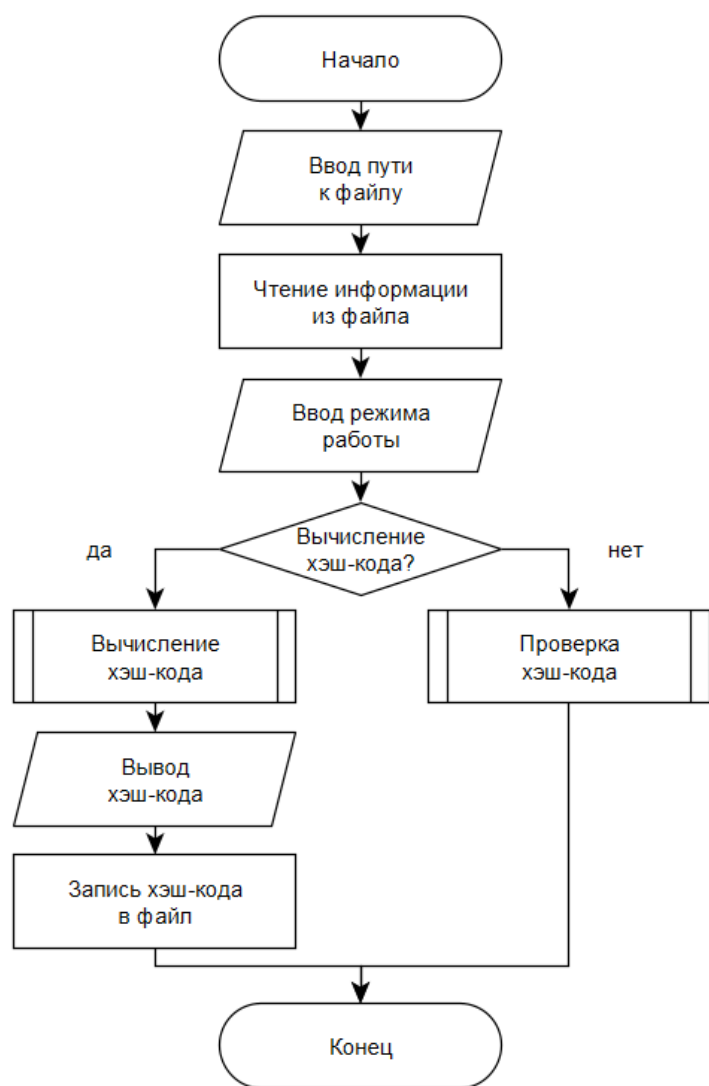


Рисунок 8 – Алгоритм программы вычисления хэш-функции SHA-256

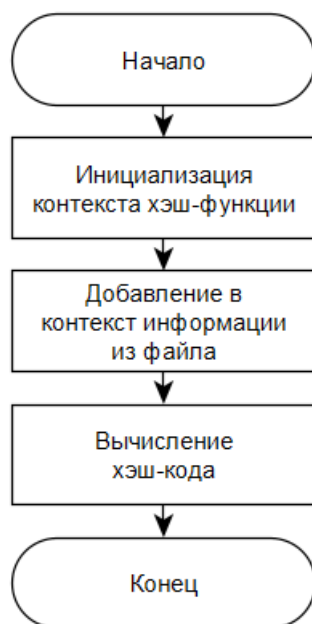


Рисунок 9 – Алгоритм функции «Вычисление хэш-кода»

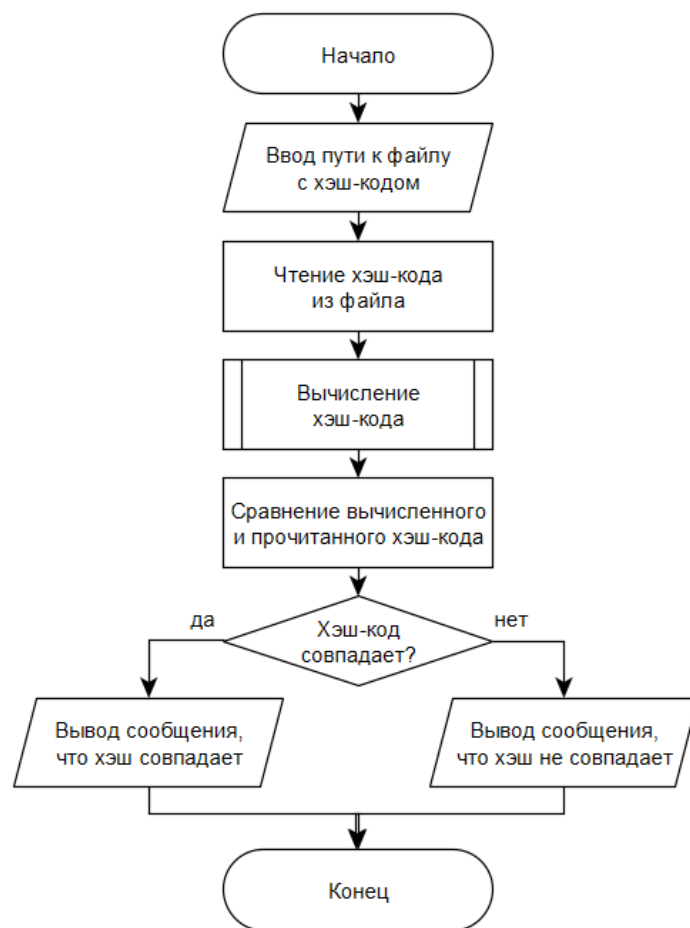


Рисунок 10 – Алгоритм функции «Проверка хэш-кода»

Алгоритмы программы шифрования и генерации ключей RSA представлены на рисунках 11 –13.

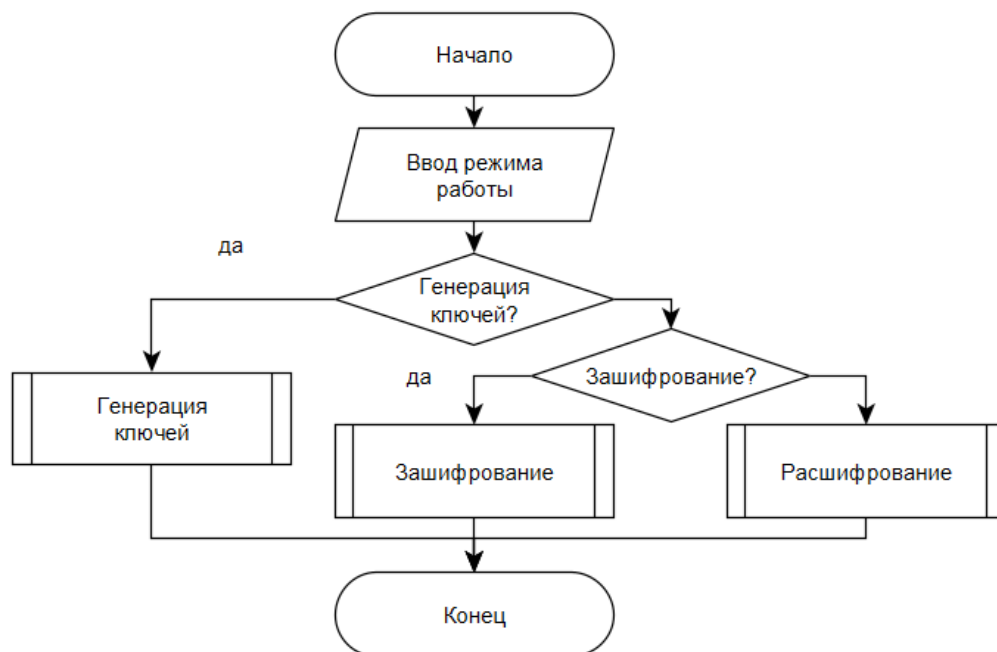


Рисунок 11 – Алгоритм программы шифрования и генерации ключей RSA

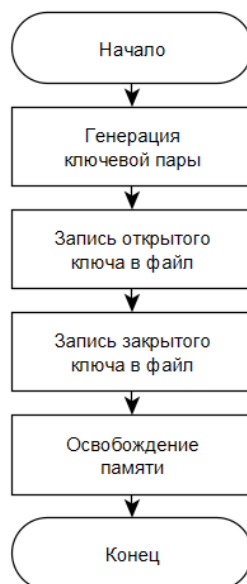


Рисунок 12 – Алгоритм функции «Генерация ключей»

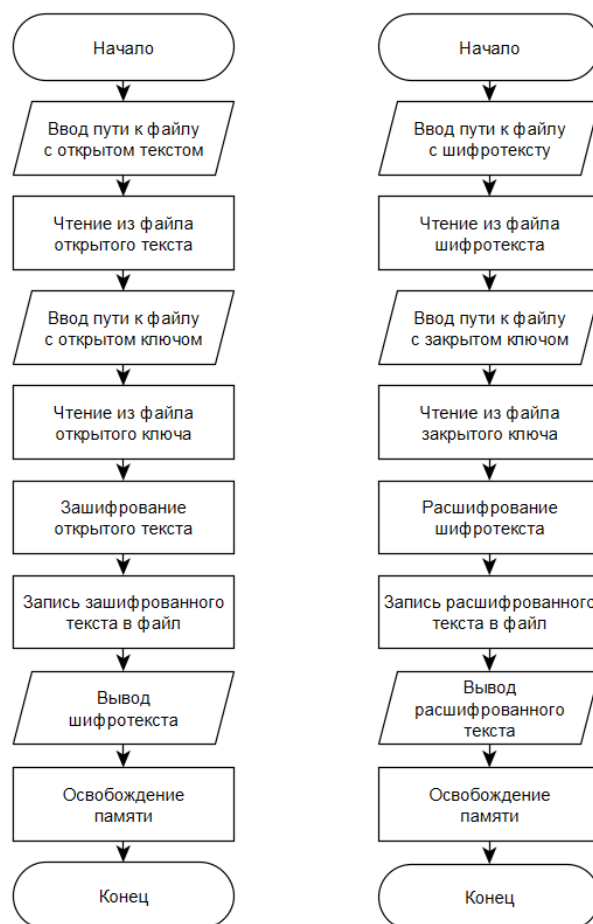


Рисунок 13 – Алгоритмы функций «Зашифрование» и «Расшифрование»

соответственно

Алгоритмы программы для работы с электронной подписью представлены на рисунках 14 – 15.

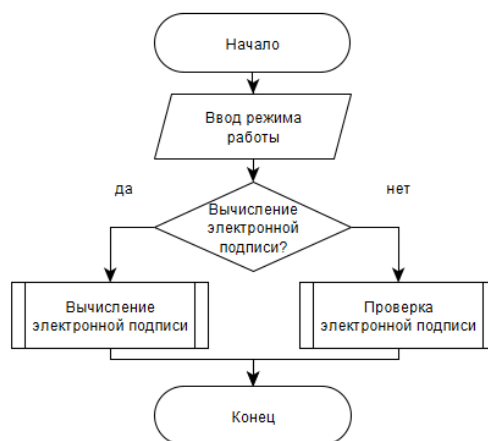


Рисунок 14 – Алгоритм программы работы с электронной подписи

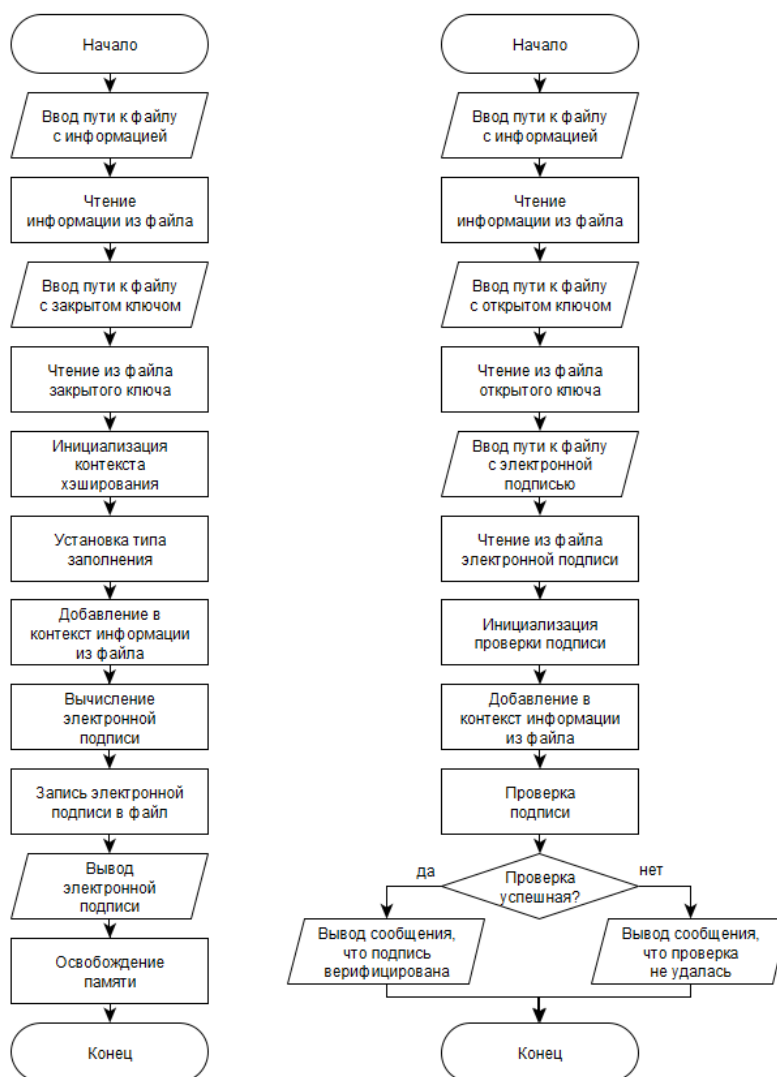


Рисунок 15 – Алгоритмы функций «Вычисление электронной подписи» и «Проверка электронной подписи» соответственно

Алгоритмы программы создания сертификата и проверки электронной подписи представлены на рисунках 16 – 17.

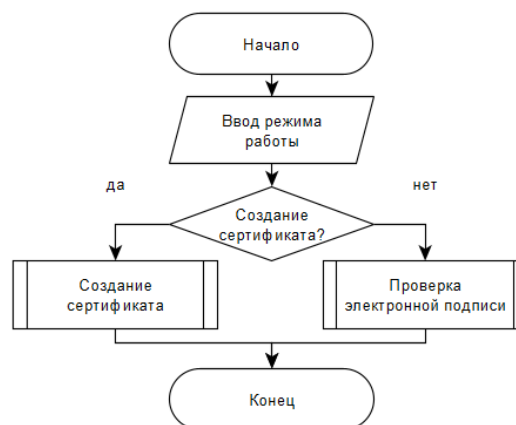


Рисунок 16 – Алгоритм программы создания сертификата и проверки электронной подписи

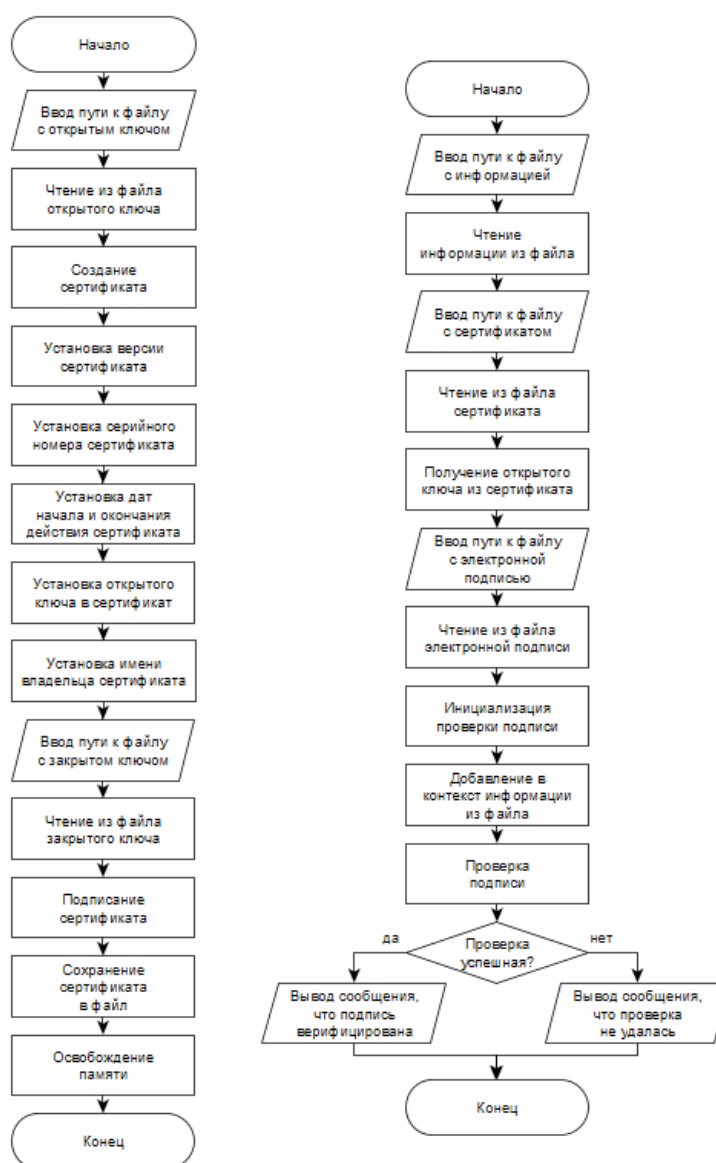


Рисунок 17 – Алгоритмы функций «Создание сертификата» и «Проверка электронной подписи» соответственно

7.2 Проверка работоспособности примеров реализации функций по применению криптографической библиотеки

7.2.1 Проверка работоспособности программы шифрования AES-256 GCM

В текстовый файл plaintext.txt был записан открытый текст.

Содержимое файла plaintext.txt:

```
GY3OQeG4ouXmj2m0nEeu3YzkC6RVDKIEi6QIyN7uQ9xBpZ5SgXqDAWJarMAvxFo3OT  
SHrdULuxYVuZe05BBw4r4Tnhw4ki68guKVOjnnrTcB3XxelAktrbSusGoXbUsPfsYV  
rphUj89GJOfkzLah7aP2fKr0rVGHBiSZreeTQOfFYmtYYq71Auc18xBfWHuNS9AYH6  
X3DdvlLxs4VLivIuKpZWpheIuR9R8GvaKGq96galF84VWyXrONhUufCIk1q198LJP5  
m1HLV5fQW6htEFiAtHE2Kn2TOw6UEebtZe0v5kXffskjFjDTsCbVHLZmqmjfYJNtIX  
A17C1j8y4Q3x6dUS8AxGw1wUpTcCefqhOlZOZirQFz2AjVUrNYvcsTljy0vCiYxqgq  
GVbOHVvvUgLfFIST6qtXBnuXivhkJLbnfNtsjniRUXAfOZfPn0ziTk3tYSFDPvuc0h  
kKam0XOUUnU198Tx7T6JFbiW9vLXBUuUs7uv8I42sw0bpU7on4Nm6BxBDPiWgAxGkGz  
tqxmWhCCsZZ7wONIXWFWQWNY0UrQvu03HmXOyGGlz305LK1g3JBhXA9dVBD1EBwQ6z  
aEkP053bLTjMZjQ8jlxmKZd19tbisTmew5tl4zprYxiYNRxR7SCViaZLjhb9H1X94  
H4uZJ1aGfiQ1GI1lXm478UrMZt2oENSf6JRVwR4S90i6eRDpPaxcfa2chdLHQH9bce  
h1IypJ6sAYcIiddhuKCZ0X0yYjSnBVpLPWFCaQIM3Dk9cmBDMw3aju7epvepoVGajD  
Jzm2Al6ZmSt3ONWinmKtWUIB1Q567K3sekuBk50f6D11lo99gcy8EEce7ewNaDTwIE  
TYY5utCwRHGke7ocIuzOB8dHvYrGv5TWiyuIc1LVHTO3N0OyfXzQQZGgyo67qAHois  
hlhLvrBBbG6m99OMDOXFncv12osQsmMraK8Pdl8wPjhrGEreZv7q4mKXefAg7DnyI1  
J8GUdoiRFxPijicdCuQyFZMoKB0AmcBaMq
```

После запуска программы были введены:

- ключ: HDtTfHff4Z3VFPTYpgHcYmaBblQSwtaj;
- вектор инициализации: k95U7ctB1pRonE62;
- дополнительные данные аутентификации: Reference_example_variant1;
- режим работы: 1 (для зашифрования), 2 (для расшифрования);
- путь к файлу: plaintext.txt.

После завершения работы функций зашифрования или расшифрования шифротекст или расшифрованный текст записываются в файлы cipher.txt и decrypt.txt соответственно.

Результат работы программы шифрования AES-256 GCM:

```
Введите ключ: HDtTfHff4Z3VFPTYpgHcYmaBblQSwtaj  
Введите вектор инициализации: k95U7ctB1pRonE62  
Введите дополнительные данные аутентификации:  
Reference_example_variant1  
Выберите режим работы:  
1 - зашифрование;  
2 - расшифрование;
```

0 - ВЫХОД.

1

Введите путь к файлу: plaintext.txt

Тег аутентификации:

0000 - 48 fa 4f 4c bb 63 fc 84-6e 9d 90 4b 50 b0 ce dc

Зашифрованный текст:

0000 - 27 ad 86 97 f6 8d e5 8e-47 07 6a 8a 60 88 a8 73
0010 - cd 17 08 b0 fe fc 22 d1-7d 8c ff 06 13 2c 69 6b
0020 - 55 1c e5 5a 44 fc 74 ce-a3 af 7e 50 7e af 10 9e
0030 - 3b a6 2f d2 16 d6 c4 b2-b2 98 c7 d5 24 4b f2 02
0040 - 52 d4 6b 9b 9e 4d 12 bb-8d dc 5b 94 af a4 e8 43
0050 - fe af 15 c7 be cf 65 bf-e0 29 27 eb 8a 87 73 0e
0060 - 8f 1a d9 a1 9e 92 3c 29-68 e1 e6 fb a0 e6 58 bc
0070 - 0f 2b 39 07 a0 d9 d2 4e-8c 28 55 77 8a 01 2b 01
0080 - b7 00 2a 8d 9c 28 22 c1-4b 56 cd 9e fe bf d8 64
0090 - f7 1a c1 ba bf 48 6e 11-34 e3 c5 ee 63 7b 61 12
00a0 - b0 4b 45 98 ed 35 45 97-d2 57 fc 0d a9 e6 bd 0a
00b0 - 18 b7 16 08 46 3b 6c e9-38 0a 85 2d da 45 a6 75
00c0 - 3e ac 7b a7 73 fd bc 8c-e7 ae 01 4e 04 af e8 44
00d0 - a0 1d ce 04 87 c8 84 eb-3b 16 d9 74 62 a6 84 ac
00e0 - 21 61 99 d3 76 29 fa 39-59 8b 63 6b 0e 77 3e aa
00f0 - 85 79 48 ad e1 ec 5c f9-38 60 36 08 e5 35 5d 75
0100 - 96 9f f2 be 2d bd 44 13-4e 5a 4d 0d 80 2f 12 a9
0110 - b9 51 88 21 1f 77 75 75-88 ba 4f 05 e8 d7 c0 50
0120 - dd 44 8d ca d0 1f 5b ad-16 bf 88 59 55 01 fe 6a
0130 - 85 b7 78 e4 d2 da 50 55-9a f8 8b c8 7d 23 b8 d0
0140 - dc 94 1f aa 7d 9a 33 2f-06 93 5a 14 e4 af 62 35
0150 - 5b f3 74 c4 6c 4e b0 b9-16 a5 08 ca 2d 3b 8b 3a
0160 - af 4c 5d a5 c8 09 e6 a3-87 de f6 81 1b f3 81 c1
0170 - 35 bc 54 a2 0d 9f 2e 04-cf 2d dc b0 ba 54 0e 3c
0180 - e3 1e b3 18 b2 de 50 f8-69 28 d3 a4 b7 99 78 70
0190 - 21 2f 0b bf 36 13 3c 23-ac fa 14 39 18 2f 34 32
01a0 - 97 39 c5 95 c1 15 86 bf-8a 53 75 fb 85 4e fb 5e
01b0 - e8 5f 0f 34 be aa f6 b2-a3 3a 82 69 64 f5 ef 7c
01c0 - 9b af 89 d1 56 7a 8f 1a-c1 3a c7 a1 eb d6 7b ad
01d0 - ef cf c6 e8 7a 6f 35 ab-dd a8 64 a3 3b 76 62 2e
01e0 - 47 7c 68 22 9a de b0 10-e7 61 dd d4 24 ab 56 33
01f0 - 50 f5 8c 48 3d 6b 46 64-19 18 31 61 5c 80 e4 62
0200 - 08 f0 e3 a4 5b 5c de 29-e9 e1 96 ca 49 7e d5 4c
0210 - d1 fa ce 8c c9 b0 cb 4e-37 2e 75 4b 5b 71 29 ea
0220 - ca dc 91 05 71 d4 8a f1-8b 8e 5c 06 eb c0 70 06
0230 - d8 43 38 d1 4e 87 ac 94-d2 c2 4c 8a 53 0b 9f 6c
0240 - cb fc f9 36 73 aa ba 5b-1d a6 e7 62 15 8a 2e 8d
0250 - 3c 93 67 33 1b fc 9e 55-4a 3b 56 fb a1 0e 66 88
0260 - e2 68 71 0b 78 e0 88 de-47 18 99 fa e7 42 28 80
0270 - cb ad 5f ed a1 59 c5 fd-7c cf 69 5c 37 d6 c2 b9
0280 - f9 ce 28 b7 23 92 c2 3a-b1 1c 3c a2 09 86 c5 a6
0290 - 8a b8 eb 2a 92 d0 50 44-46 4e a1 d3 64 38 4d 1a
02a0 - 62 22 be 56 76 fe 8b cc-24 2a c1 21 b6 86 20 25
02b0 - cb 1a 97 51 38 3c 10 d1-b6 a1 6a e6 eb 44 6d 64
02c0 - 57 e2 cb 57 71 fe df db-67 b1 11 13 bf 19 12 cd

02d0 - 49 15 b0 c1 a7 a8 12 6e-4b 45 49 0c e4 29 de 7d
 02e0 - 21 cd 48 fd de 62 1d e1-76 01 2f 34 d4 e8 60 9d
 02f0 - 22 a8 0f 8e f8 84 f3 23-08 1d cd 91 c9 07 02 04
 0300 - 5a 8f ba 70 a8 a4 ca 65-78 13 39 22 fa 3d 76 bd
 0310 - 9a 79 09 8c 05 c0 8f 5f-a6 4d 33 79 d9 89 10 6b
 0320 - 80 67 c6 56 35 02 cd d2-dc e0 f5 1f b8 cd c2 4b
 0330 - 96 b6 ae 9f 18 3b 54 37-39 8a a5 bc 85 0d df b0
 0340 - 53 c9 22 0b f7 4a bb c2-18 30 6c 98 49 f7 24 f1
 0350 - 25 34 df d4 ec 35 4a fc-23 a3 fb 0e a4 70 ea 4e
 0360 - 5e 20 be 8c 6c 36 93 20-cd a6 0d 4e ad ae fc a1
 0370 - 9b 9d f2 8c 91 d1 5d 8a-b2 d0 fa a3 35 81 dc 93
 0380 - b7 e4 da 4e c6 71 7c bf-a7 f6 41 f5 e3 fc 28 5f
 0390 - 9e c9 cd 77 cc ff f8 09-35 48 c6 a1 2d 53 1c 2d
 03a0 - e1 37 26 34 e8 20 9b b1-67 08 94 d0 e6 c0 87 ad
 03b0 - e6 31 e8 da 1c c2 bb 9b-b8 cb d7 84 6a 1b 41 f1
 03c0 - 3a 76 e9 8c b0 a5 28 7e-50 49 21 dc a4 38 a8 c5
 03d0 - 31 59 83 c0 b5 7e 10 44-67 1d 85 08 dd 80 b1 2d
 03e0 - bc 3a e4 c0 18 31 70 75-ab 2b e0 55 14 f2 c3 dc
 03f0 - ab 5b f0 98 28 83 7a a6-9c bf 4e 7f 2a b1 76 ec

Выберите режим работы:

- 1 - зашифрование;
- 2 - расшифрование;
- 0 - выход.

2

Расшифрованный текст:

GY30QeG4ouXmj2m0nEeu3YzkC6RVDKIEi6QIyN7uQ9xBpZ5SgXqDAWJarMAvxFo3OT
 SHrdULuxYVuZe05BBw4r4Tnhw4ki68guKVOjnnrTcB3XxelAktrbSusGoXbUsPfsYV
 rphUj89GJOfkzLah7aP2fKr0rVGHBiSZreeTQOFfYmtYYq71Auc18xBfWHuNS9AYH6
 X3Ddv1Lxs4VLivIuKpZWpheIuR9R8GvaKGq96galF84VWyXrONhUufCIk1q198LJP5
 m1HLV5fQW6htEFiAtHE2Kn2TOw6UEebtZe0v5kXffskjFjDTsCbVHLZmqmjfYJNtIX
 A17C1j8y4Q3x6dUS8AxGw1wUpTcCefqhOlZOZirQFz2AjVUrNYvcsTljy0vCiYxqgq
 GVbOHVvvUgLfFIST6qtXBnuXivhkJLbnfNtsjniRUXAfOZfPn0ziTk3tYSFDPvuc0h
 kKam0XOUUnU198Tx7T6JFbiW9vLXBuUUs7uv8I42sw0bpU7on4Nm6BxBDPiWgAxGkGz
 tqxmWhCCsZZ7wONIXWFWQWNY0UrQvu03HmXOyGGlz305LK1g3JBhXA9dVBD1EBwQ6z
 aEkP053bLTjMZjQ8jlxmKZd19tbisTmew5tl4zpRYxiYNRxR7SCViaZLjhb9H1X94
 H4uZJ1aGfiQ1GI1lXm478UrMZt2oENSf6JRVwR4S90i6eRDpPaxcfa2chdLHQH9bce
 h1IypJ6sAYcIiddhuKCZ0X0yYjSnBVpLPWFcAqIM3Dk9cmBDMw3aju7epvepoVGajD
 Jzm2Al6ZmSt3ONWinmKtWUIB1Q567K3sekuBk50f6D11lo99gcy8EEce7ewNaDTwIE
 TYY5utCwRHGke7ocIuzOB8dHvYrGv5TWiyuIc1LVHTO3N0OyfXzQQZGgyo67qAHois
 h1hLvrBBbG6m99OMDOXFncv12osQsmMraK8Pdl8wPjhrGEreZv7q4mKXefAg7DnyI1
 J8GUdoiRFxPijicdCuQyfZMoKB0AmcBaMQ

Файлы plaintext.txt и decrypt.txt имеют одинаковое содержимое.

7.2.2 Проверка работоспособности программы вычисления хэш-функции

SHA-256

В текстовый файл text.txt был записан текст, для которого будет

ВЫЧИСЛЯТЬСЯ ХЭШ-КОД.

Содержимое файла text.txt:

```
bKegnXS9FMkRyzww7D6YQ53ktZfYq511jQQunq4apZV4dK2ItZErmLBFirH4OiSoPo  
lAAZ987InmDjZYuax3Mzg1BTnBYEKhcJSYuaXIVUDmm2lK1q67Sf4CbhTxEPswOFp0  
wPEo5buX4xv72ypf1t7P3kfP4fw3rmiRTjDUmI6wraRAYNPCXvsoZCc0xBelbeiJoa  
2nbGD9go0GESitFengJeyiwcmfcnUarUbC4b1PiLbXU5n8GXVFYRNmn1QUVMrYQmb6  
vftin7DZod5ZyrF036X5yfxF0N22bKGD6ZXN19IqtFfwZeCn4tan6nmuVniFjP7Uzv  
WFOV1ZuEzDttNVO5Z12CjXEmYBPqbcOCmdtZgoZtaLYFWJKgTbiX1Q09T1nBZhJMSO  
sWzqXxwiVv1AP2kciVv24fiVn0FzBn0bGaKhO4Xm4SS5GYpRXEF1EvHx5OD4sZyHbK  
yNQ9aT4fGHDe7YIvviAwSiMxOfFaSe7twjVeLk
```

После запуска программы были введены:

- режим работы: 1 (для вычисления хэш-кода), 2 (для проверки хэш-кода);
- путь к файлу с хэш-кодом: hash.txt;
- путь к файлу: text.txt.

После завершения вычисления хэш-функции хэш-код записывается в файл hash.txt.

Результат работы программы вычисления хэш-функции SHA-256:

Выберите режим работы:

- 1 - вычисление хэш-кода;
- 2 - проверка хэш-кода;
- 0 - выход.

1

Введите путь к файлу: text.txt

Хэш-код:

6a9a7a5237f5660624dd96fc41027f102f1efcf90904e1fed966c7e76d5b17bc

Выберите режим работы:

- 1 - вычисление хэш-кода;
- 2 - проверка хэш-кода;
- 0 - выход.

2

Введите путь к файлу с хэш-кодом: hash.txt

Введите путь к файлу: text.txt

Хэш-код совпадает!

7.2.3 Проверка работоспособности программы шифрования и генерации ключей

RSA

В текстовый файл plaintext.txt был записан открытый текст.

Содержимое файла plaintext.txt:

```
hYseiInf5iqftKz2mVSgZwQQyuVIhzetvlh1LvRwUR5b6GIi8dZSPcM2y4WqhryOiR
RfHnS9vDRYOGtXY5r2ngzSGf73MtDZgTcoNrN8QoN6lcxz6GYRzxxl0YDLqcZgR5gC
op8dt9JQYETE4so5SMk5J72YWIhUPVWfK54psHU7mg3W6aiSoAjVgYtBwIMDkvBIZg
JqoX6R4ysJ0Wwq9
```

После запуска программы были введены:

– режим работы: 1 (для генерации ключевой пары), 2 (для зашифрования),
3 (для расшифрования);

– путь к файлу с открытым текстом: plaintext.txt;

– путь к файлу с открытым ключом: public.pem;

– путь к файлу с шифротекстом: encrypt.txt;

– путь к файлу с закрытым ключом: private.pem.

Сгенерированные открытый и закрытый ключи записываются в файлы public.pem и private.pem соответственно.

После завершения работы функций зашифрования или расшифрования шифротекст или расшифрованный текст записываются в файлы encrypt.txt и decrypt.txt соответственно.

Результат работы программы шифрования и генерации ключей RSA:

Выберите режим работы:

1 - генерация ключевой пары;

2 - зашифрование;

3 - расшифрование;

0 - выход.

1

Закрытый ключ:

-----BEGIN RSA PRIVATE KEY-----

```
MIIEowIBAAKCAQEAsK6UcpSCHvjIYfY9+KwbLOHhQd0ar2+4OdRxJtqD3EaknOyZ
Qj32vW00HBiUdRDP469qD9FuZmwkWXBSS30HKNWvJz2BMrjjHA0GGf0BIMAbBahK
2DzxoNjSt5qJ7LYWrL86sWFIvevsos77W07mMgutqzC6QclY1NS50rkkgxFUC0JA
L3rEc6KBHwqRtrINIupdEX9BSwyntxgWypZzdfApVwtsEqrdalPcKPTJt6za802b
sY0UBqbHROJxvYBAvhSw2f6jx+M3mGb++UmFa3Suj5dmZgMiyIEUKJ/VEdjASkyY
bbcCTARqGhfCxoFhAbknKW3t+RKu2hAHl20cowIDAQABAOIBAQCPh40PWAj6MJi
YNgItlp5PC1+bj7suOKSrh5PLBo++Vbl69+mwo07G9hnnh28yzasqW+QisYU4Ct/
PTR9BDYlhRwS/IbJcO5AzegQa1xbjGMxbSHcElplQt9Hbj+makbp/yrrf/PZ7njN
1j/V0o81ITK5B4sKObHlDsHNCr+iD7E0k34+T1sWoqSoJbE1Z433Vo/TCFjkSBHG
5GQ5J1+2EbUcN0NYe5v6Fo0lLCO0ZJAqJvJEupcnmMZBWLasRajoHhehYaKab+Jo
FqW/8j6sGU/CA7rAsjEyfaqC4OYzucaEfP7051YJHzaXzmI9u+iXZW3RKBmXN2g1
QnQ9gNexAoGBAOKWOGDoXZr7G8/Ng3waQoduoyxlnGUucsebbVO0oBOBQIj/TfUp
lFplqC56W28TnU8WohRyCAEwe9qtLz4lp5b8Q6DZABBiixj3bqejU0NjYHpsDlXR
SU2XIWLrb7x5TWZO+0BVJjhGpO+c4ENGqNRW3hQL4tuwOZeC8yF4HYA7AoGBAMed
9JkVY/QFGd5FYQ99ytnUt/Q1jR0ewH2N6oE1ZPM9czlVZkr3zlv/4SKQ5zoDFCiM
LdzShzqor6D+JneG3vg0NgxHaP/YJ+ZEsICeOr28X0ZRB4yhziJewo4kOd2Awc+e
```

```
w88MO+zqj5l4kZ0D0li3sStvacG+lY5E5Yq9PTa5AoGASoDUZuDgrm301h4gJrVH
vNpfIuECTK66JPHxP727iNbaoHE6A+u5EQlw1F99IwfTCMDLYeegxCtZ/ixFxWyC
L34hColvoPOWAw6QvuRyVZUeiwJusD8RmEaiPqjY0aMdFQ/RQ3T8DTMFVphBkT1x
k+hAO10xKcVHGo9/PDkPkjMCgYBlQC+DPqBWdAftAE/9JZcw6n7sNgV/0ctdZBCg
vU6Z0oApgrHCMPRS+/VcLRe8Cze8YW4Z5geFr252QL00STDrwL/xdn9B1ohWelz0
wE2XU+cXqZ4krSUos5wHsURQoDCueR9k9BWpVEjGVau2cp6zxWjuWTLRfTQq17N+
rgJQsQKBgF1OnWAGfKxR2UQuEuTLPB1OF1fmXMZPwI4/rfXDAsOJvYc+I+Ero5sP
kN3OuHBMz5BRCXzlpRe3k7ZHGS8YF5mB2UMT6M1nmOkd2IpLYeJfTKHdPwkSDgeN
RRfB9LYJ5xRCubkYe4wtIhOLUmQI/6o/btu/p1UjqGNd53G/Vlg/
-----END RSA PRIVATE KEY-----
```

Открытый ключ:

```
-----BEGIN RSA PUBLIC KEY-----
```

```
MIIBCgKCAQEAsK6UcpSCHvjIYfy9+KwbLOHhQd0ar2+4OdRxJtqD3EaknOyZQj32
vW00HBiUdRDP469qD9FuZmwkWxBSS30HKNWvJz2BMrjjHA0GGf0BIMAbBahK2Dzx
oNjSt5qJ7LYWrL86sWFIvevsos77W07mMgutqzC6QclY1NS50rkkgxFUC0JAL3rE
c6KBHwqRtrINIupdEX9BSwyntxgWypZzdfApVwtsEqrdalPcKPTJt6za802bsYOU
BqbHROjxvYBAvhSw2f6jx+M3mGb++UmFa3Suj5dmZgMiyIEUKJ/VEdjASkyYbbcc
TARqGhfcxOFhAbknKW3t+RKu2hAHl20cowIDAQAB
-----END RSA PUBLIC KEY-----
```

Выберите режим работы:

- 1 - генерация ключевой пары;
- 2 - шифрование;
- 3 - расшифрование;
- 0 - выход.

2

Введите путь к файлу с открытым текстом: plaintext.txt

Введите путь к файлу с открытым ключом: public.pem

Зашифрованное сообщение:

```
4db1c1a62900751332b92a76f4a58c6798f2ed4290f66271bc4aefd352e585e7fc
6613322948d4066fe20dcfdb4caed745287ac9c9af4359831937dd0578bb4ce236
bd73762b10d33d3593655ea3749cc4094dcc39a15948c9a49fb352d2e30838c475
fe909908c005da6cfe981dec3b1fd3063b6753672e04d30c17487e33cc0b406d77
dc1f5d5af4dd3f31597b3005a45ada8d824f58bbca55291d074babbb2a2deddf31d
9788e9e28ba95159d4862c3d09c56655c90b428ba2d7f40195e95a1c9c584c92a9
3997662aceabe05185e7ce8f20c2ca9b37a145c31f2c0c6c32c9f8cf26b224ffd7
6ea63aaff87db84cb3be0ff334118b58969e120ade8f1e9531
```

Выберите режим работы:

- 1 - генерация ключевой пары;
- 2 - шифрование;
- 3 - расшифрование;
- 0 - выход.

3

Введите путь к файлу с шифротексту: encrypt.txt

Введите путь к файлу с закрытым ключом: private.pem

Расшифрованное сообщение:

```
hYseiInf5iqftKz2mVSgZwQQyuVIhzetvlh1LvrWuR5b6GIi8dZSPcM2y4WqhryOiR
RfHnS9vDRYOGtXY5r2ngzSGf73MtDZgTcoNrN8QoN6lcxz6GYRzxxl0YDLqcZgR5gC
op8dt9JQYETE4so5SMk5J72YWIhUPVWfK54psHU7mg3W6aiSoAjVgYtBwIMDkvBIZg
JqoX6R4ysJ0Wwq9
```

Файлы plaintext.txt и decrypt.txt имеют одинаковое содержимое.

7.2.4 Проверка работоспособности программы для вычисления и проверки электронной подписи

В текстовый файл `message.txt` был записан текст, для которого будет вычисляться электронная подпись.

Содержимое файла `message.txt`:

```
Dd6bnl6Cl0cz9o4vsX7txjhdGUKve7PyKCfyRgxmnQ5917sFdZtfNWXTmdcJR1xOQu
nLHKWxtLv4v4wa6iuWW52ANZmTO4jAmdUheBuPdUAKT8aGcu0wzjHLQYNa6ZLGjCZb
I6Y7tyCJt61xUmByp4QVGXPXYWmJeIVocyvqdy6f4vZVHuvPwssmF2F77Df5pCYAajn
Zz3J3TOup2j9XPrghKh84kwAs3L2TV4Tl3VFoiRxsbi3AWD66oi7kZxQXbdfRefZtZ
dgG3p6BxGPwkO3aEApDl8dQWPRXe2D9zMxnAveqX4EUWa6yRQFJMX3vgY1mkPzl64k
62w0yiZe864zVlVoHmuBdf2ipQof4jA52yjqBdpJNi643yOWw4GQXKI1A60ihIewGX
9JqWm3pEMZxpDz5Vy7yJNLViEqPJT2l1GPcx6PmWNk1L5CZUj82YgV5PX1XuMiXgZx
7fN7ME1cFLUldhGSsHga37vTlezDaJpcCUXDjHga9b3hGvZtQK08lqYNpYShAy6X4a
QUkFpieeMNHSNq9qTd4Od3GJVUWGXIGoNfbKc0z8Rh0wJy4UM37uKck5VjviF9Fzqu
CqzffeE4OpDtDNbJkU8OA5jrTwvCKIEckec8xjtWmpn5JYdGa4hnXqbOL34YERYRnt
swfKCCUpdgem8hQT17xqZ90Ppw8FUShaWWMai6PnMNzpy3qKTatEs4U66VQRnPr2ZG
68iR6efi5DMTDjbcZvjxU4KjopYZWVRL9b80pVoPubzlyvd9Alyb5VitFRdReFzSpe
BstGVYe6hxlZvTl1fvR9a45J5V2bqAfuyRVIfrCvNAjcn04mwjYbJyzFnUd3iJUqOyZ
xcjLbwuUjcyZL1znENxMiqPValUnpHD7E7Y4Vm4KRJhBY7nB0zl3y2Edq626bbRb8
pcNRZwim4P3zv75pnQ6VtVERpD3Iqt1ZbvKaniN6ELZ4U1UkWoqdWlup8FXkaQIWFC
ODc3qeBMju7rF1yiOQIEpJmMjbfRnBNrNz
```

После запуска программы были введены:

– режим работы: 1 (для вычисления электронной подписи), 2 (для проверки электронной подписи);

- путь к файлу с информацией: `message.txt`;
- путь к файлу с закрытым ключом: `private.pem`;
- путь к файлу с открытым ключом: `public.pem`;
- путь к файлу с подписью: `signature.sig`.

После завершения функции вычисления электронной подписи результат записывается в файл `signature.sig`.

Результат функционирования программы для вычисления и проверки электронной подписи:

Выберите режим работы:

- 1 – вычисление электронной подписи;
- 2 – проверка электронной подписи;
- 0 – выход.

1

Введите путь к файлу с информацией: message.txt
 Введите путь к файлу с закрытым ключом: private.pem
 Электронная подпись сохранена в файл signature.sig
 Электронная подпись:
 e8d362f7742a3c60fa8bba0a8218a7a0fe8c86b559a9d3f2e386097e961df683ee
 c9779b2eac039ede732598c87f33eeb33ac84a4d25c7e4496af9822ec63653b29c
 32f21fcf6f79c7bd82021e0b41507f47d66b265dd61321becad170e19c2ea8101c
 8243d2a9e9006f75096bfc974e54d125ee9df34059951c8280b7d3d94e542200fb
 ffa84bf42b1b880f57d3798af7d15ef33b051cb28e8d8c3ff8dc2e94f34a7125d9
 b799253fd74da848fe6fc4b3b48c078e5d599dd36ff184a3ebe8ee4496e04375b3
 b5d8d0388bba61d81393eb2136c0c3ef555b049d850d1f3824787def12eb453a1d
 b8e598b2974ea7030731a5398bdcfb2c009b77a19d87d4b4a3
 Выберите режим работы:
 1 - вычисление электронной подписи;
 2 - проверка электронной подписи;
 0 - выход.
 2
 Введите путь к файлу с информацией: message.txt
 Введите путь к файлу с открытым ключом: public.pem
 Введите путь к файлу с подписью: signature.sig
 Подпись верифицирована!

7.2.5 Проверка работоспособности программы создания сертификата и проверки электронной подписи

В текстовый файл message.txt был записан текст, для которого будет вычисляться электронная подпись.

Содержимое файла message.txt:

Dd6bn16C10cz9o4vsX7txjhdGUKve7PyKCfyRgxmnQ5917sFdZtfnNWXtmdcJR1xOQu
 nLHKWxtLv4v4wa6iuWW52ANZmTO4jAmdUheBuPdUAKT8aGCU0wzjHLQYNa6ZLGjCZb
 I6Y7tyCJt61xUmByp4QVGPXYWmJeIVocyvqdy6f4vZVHuvPwssmF2F77Df5pCYAajn
 Zz3J3TOup2j9XPrgh84kwaS3L2TV4Tl3VFoiRxsbiX3AWD66oi7kZxQXbdfRefZtZ
 dgG3p6BxGPwkO3aEApDl8dQWPRXe2D9zMxnAveqX4EUWa6yRQFJMX3vgY1mkPzl64k
 62w0yiZe864zV1VoHmuBDf2ipQof4jA52yjqBdpJNi643yOWw4GQXKI1A60ihIewGX
 9JqWm3pEMZxpDz5Vy7yJNLViEqPJT2l1GPcx6PmWNk1L5CZUj82YgV5PX1XuMiXgZx
 7fN7ME1cFLUldhGSsHga37vTlezDaJpcCUXDjHga9b3hGvZtQK08lqYNpYShAy6X4a
 QUkFpieeMNHSNq9qTd4Od3GJVUWGXIGoNfbKc0z8Rh0wJy4UM37uKCk5VjviF9Fzqu
 CqzffeE4OpDtDNbJkU8OA5jrTwvCKIEckec8xjtWmpn5JYdGa4hnXqbOL34YERYRnt
 swfKCCUpdqem8hQT17xqZ90Ppw8FUShaWWmai6PnMNzpy3qKTAtEs4U66VQRnPr2ZG
 68iR6efi5DMTDjbcZvjxU4KjopYZWVRL9b80pVoPubzlyvd9A1yb5VitFRdReFzSPe
 BstGVYe6hxlZvTlfr9a45J5V2bqAfuyRVIfrCvNajcN04mwjYbJyzFnUd3iJUqOyZ
 xcjLbwIUUjcyZL1znENxMiqPValUnPHD7E7Y4Vm4KRJhBY7nB0zl3y2Edq626bbRb8
 pcNRZwim4P3zv75pnQ6VtVERpD3Iqt1ZbvKaniN6ELZ4U1UkWoqdWlup8FXkaQIWFC
 ODc3qeBMju7rF1yioQIEPjMMjbfRnBNrNz

После запуска программы были введены:

– режим работы: 1 (для создания сертификата), 2 (для проверки электронной подписи);

– путь к файлу с открытым ключом: public.pem;

– путь к файлу с закрытым ключом: private.pem;

– путь к файлу с информацией: message.txt;

– путь к файлу с сертификатом: certificate.crt;

– путь к файлу с подписью: signature.sig.

После завершения функции создания сертификата результат записывается в файл certificate.crt.

Результат функционирования программы для создания сертификата и проверки электронной подписи:

Выберите режим работы:

1 - создание сертификата;

2 - проверка электронной подписи;

0 - выход.

1

Введите путь к файлу с открытым ключом: public.pem

Введите путь к файлу с закрытым ключом: private.pem

Сертификат успешно создан и сохранен в файл certificate.crt

Выберите режим работы:

1 - создание сертификата;

2 - проверка электронной подписи;

0 - выход.

2

Введите путь к файлу с информацией: message.txt

Введите путь к файлу с сертификатом: certificate.crt

Введите путь к файлу с электронной подписью: signature.sig

Подпись верифицирована!

8 Разработка приложения для проверки результатов выполнения практикума

8.1 Словесное описание разрабатываемого приложения

Было разработано приложение для проверки результатов выполнения практикума.

У приложения пять вариантов использования:

- проверить правильность шифрования AES-256 GCM;
- проверить правильность вычисления хэш-кода;
- проверить правильность шифрования RSA;
- проверить правильность вычисления электронной подписи;
- проверить правильность создания сертификата.

При проверке правильности шифрования AES-256 GCM требуется ввести ключ, вектор инициализации, дополнительные данные аутентификации, пути к файлам с открытым текстом и с расшифрованным текстом от студента. После выполнения шифрования программа сравнивает результат работы с полученным от студента текстом. Если содержимое совпадает, то выводятся тег аутентификации и сообщение об успешно пройденной проверке, в противном случае появляется уведомление об ошибке.

При проверке правильности вычисления хэш-кода требуется ввести пути к файлам с хэш-кодом и к файлу, для которого необходимо вычислить хэш-код. Программа сравнивает вычисленный хэш-код с полученным от студента. Если содержимое совпадает, то выводится сообщение об успешно пройденной проверке, в противном случае появляется уведомление об ошибке.

При проверке правильности шифрования RSA требуется пути к файлам с шифротекстом, с закрытым ключом и с открытым текстом. После выполнения шифрования программа сравнивает результат расшифрования шифротекста с открытым текстом. Если содержимое совпадает, то выводится сообщение об успешно пройденной проверке, в противном случае появляется уведомление об ошибке.

При проверке правильности вычисления электронной подписи требуется ввести пути к файлам с информацией, с открытым ключом и с электронной подписью. Программа вычисляет электронную подпись для файла с помощью открытого ключа, а затем сравнивает с полученным от студента файлом подписи. Если содержимое совпадает, то выводится сообщение об успешно пройденной проверке, в противном случае появляется уведомление об ошибке.

При проверке правильности создания сертификата требуется ввести пути к файлам с информацией, с сертификатом и с электронной подписью. Программа вычисляет электронную подпись для файлом с помощью сертификата, а затем сравнивает с полученным от студента файла подписи. Если содержимое совпадает, то выводится сообщение об успешно пройденной проверке, в противном случае появляется уведомление об ошибке.

Диаграмма вариантов использования приложения для проверки результатов выполнения практикума представлена на рисунке 18.

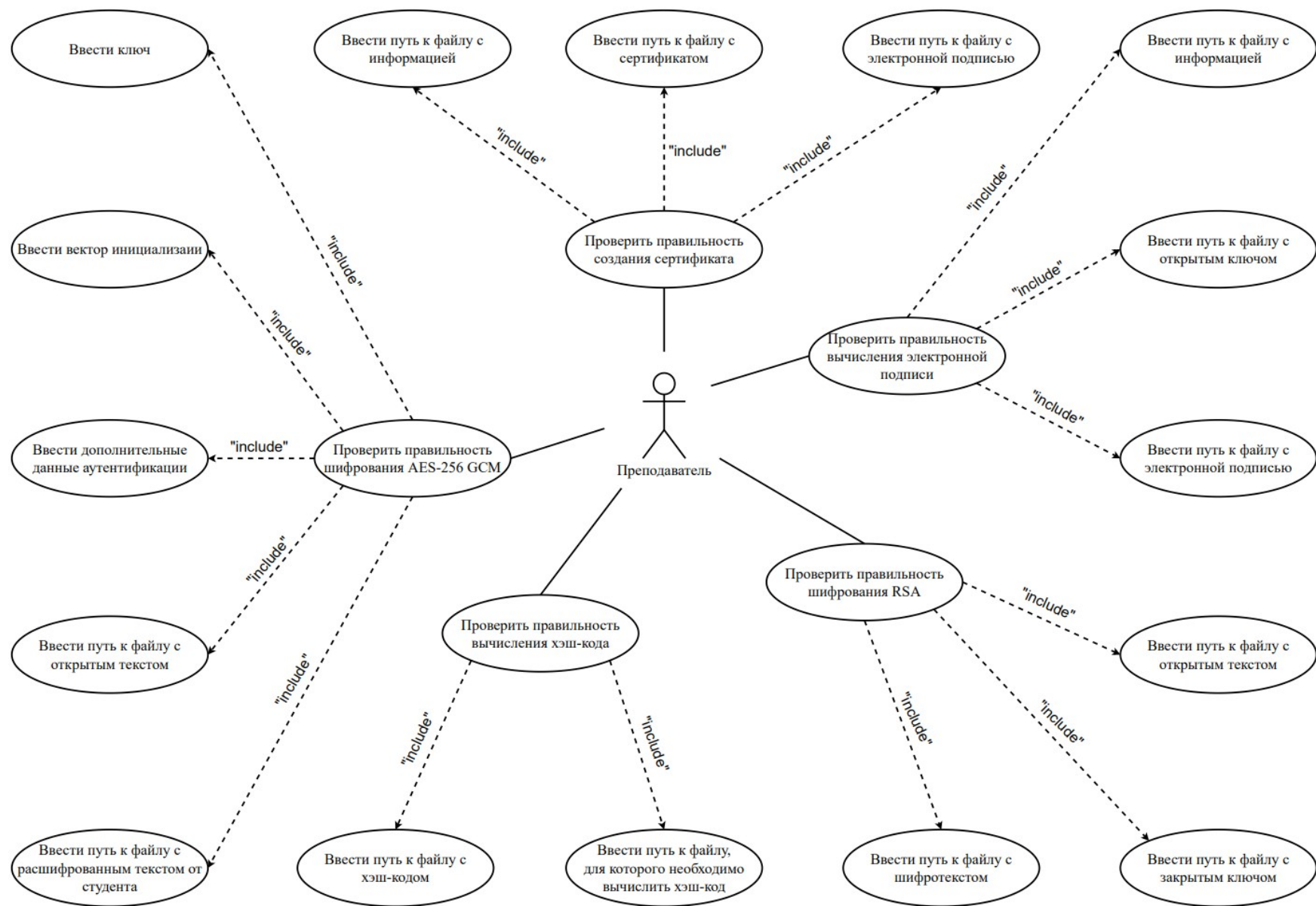


Рисунок 18 – Диаграмма вариантов использования приложения для проверки результатов выполнения практикума

8.2 Проверка работоспособности приложения для проверки результатов выполнения практикума

8.2.1 Проверка работоспособности приложения в режиме проверки шифрования AES-256 GCM

В текстовый файл plaintextAES.txt был записан текст, для которого будет вычисляться электронная подпись.

Содержимое файла plaintextAES.txt:

```
GY3OQeG4ouXmj2m0nEeu3YzkC6RVDKIEi6QIyN7uQ9xBpZ5SgXqDAWJarMAvxFo3OT
SHrdULuxYVuZe05BBw4r4Tnhw4ki68guKVOjnnrTcB3XxelAktrbSusGoXbUsPfsYV
rphUj89GJOfkzLah7aP2fKr0rVGHBiSZreeTQOFFYmtYYq71Auc18xBfWHuNS9AYH6
X3Ddv1Lxs4VLivIuKpZWpheIuR9R8GvaKGq96gal1f84VWyXrONhUufCIk1q198LJP5
m1HLV5fQW6htEFiAtHE2Kn2TOw6UEebtZe0v5kXffskjFjDTsCbVHLZmqmjfYJNtIX
A17C1j8y4Q3x6dUS8AxGw1wUpTcCefqhOlZOZirQFz2AjVUrNYvcsTljy0vCiYxqgq
GVbOHVvvUgLfFIST6qtXBnuXivhkJLbnfNtsjniRUXAfOZfPn0ziTk3tYSFDPvuc0h
kKam0XOUUnU198Tx7T6JFbiW9vLXBuUs7uv8I42sw0bpU7on4Nm6BxBDPiWgAxGkGz
tqxmWhCCsZZ7wONIXWFWQWNY0UrQvu03HmXOyGG1z305LK1g3JBhXA9dVBD1EBwQ6z
aEkP053bLTjMZjQ8j1xxmKZd19tbisTmew5tl4zpRYxiYNRxR7SCViaZLjhb9H1X94
H4uZJ1aGfiQ1GI11Xm478UrMZt2oENSf6JRVwR4S90i6eRDpPaxcfa2chdLHQH9bce
h1IypJ6sAYcIiddhuKCZ0X0yYjSnBVpLPWFcAqIM3Dk9cmBDMw3aju7epvepoVGajD
Jzm2Al6ZmSt3ONWinmKtWUIB1Q567K3sekuBk50f6D11lo99gcy8EEce7ewNaDTwIE
TYY5utCwRHGke7ocIuzOB8dHvYrGv5TWiyuIc1LVHTO3N0OyfXzQQZGgyo67qAHois
hlhLvrBBBg6m99OMDOXFncv12osQsmMraK8Pdl8wPjhrGEreZv7q4mKXefAg7DnyI1
J8GUdoiRfXpIjicdCuQyfZMoKB0AmcBaMq
```

Содержимое файла decryptAES.txt совпадает с содержимым plaintextAES.txt.

После запуска программы были введены:

- режим работы: 1 (AES);
- ключ: HDtTfHff4Z3VFPTYpgHcYmaBblQSwtaj;
- вектор инициализации: k95U7ctB1pRonE62;
- дополнительные данные аутентификации: Reference_example_variant1;
- путь к файлу с открытым: plaintextAES.txt;
- путь к файлу с расшифрованным текстом от студента: decryptAES.txt.

Результат функционирования программы в режиме работы AES:

Выберите режим работы:

- 1 - AES;
- 2 - HASH;

3 - RSA;
4 - SIGNATURE;
5 - CERTIFICATE;
0 - выход.

1

Введите ключ: HDtTfHff4Z3VFPTYpgHcYmaBblQSwtaj

Введите вектор инициализации: k95U7ctB1pRonE62

Введите дополнительные данные аутентификации:

Reference_example_variant1

Введите путь к файлу с открытым текстом: plaintextAES.txt

Тег аутентификации:

0000 - 48 fa 4f 4c bb 63 fc 84-6e 9d 90 4b 50 b0 ce dc

Введите путь к файлу с расшифрованным текстом от студента:

decryptAES.txt

Расшифрованный текст совпадает с открытым текстом

Проверка выполнена успешно.

В содержимом файла decryptAES.txt был изменён один символ, чтобы проверить работу программы проверки с отрицательным результатом. Все вводимые значения остались такими же.

Результат функционирования программы в режиме работы AES:

Введите ключ: HDtTfHff4Z3VFPTYpgHcYmaBblQSwtaj

Введите вектор инициализации: k95U7ctB1pRonE62

Введите дополнительные данные аутентификации:

Reference_example_variant1

Введите путь к файлу с открытым текстом: plaintextAES.txt

Введите путь к файлу с расшифрованным текстом от студента:

decryptAES.txt

Расшифрованный текст отличается от открытого текста

Программа выводит сообщение, что расшифрованный текст неверен.

8.2.2 Проверка работоспособности приложения в режиме проверки хэш-функции SHA-256

Содержимое файла textHASH.txt:

bKegnXS9FMkRyzww7D6YQ53ktZfYq51ljQQunq4apZV4dK2ItZErmLBFirH4OiSoPo
lAAZ987InmDjZYuax3Mzg1BTnBYEKhcJSYuaXIVUDmm2lK1q67Sf4CbhTxEPswOFp0
wPEo5buX4xv72ypf1t7P3kfP4fw3rmiRTjDUmI6wraRAYNPCXvsoZCc0xBelbeiJoa
2nbGD9go0GESitFengJeyiwcmfcUarUbC4b1PiLbXU5n8GXVFYRNmn1QUVMrYQmb6
vfTin7DZod5ZyrF036X5yfxF0N22bKGD6ZXNl9IqtFfwZeCn4tan6nmuVniFjP7Uzv
WFOV1ZuEzDttNVO5Zl2CjXEmYBPqbcOCmdtZgoZtaLYFWJKgTbiXlQ09T1nBZhJMSO
sWzqXxwiVv1AP2kciVv24fiVn0FzBn0bGaKh04Xm4SS5GYpRXEF1EvHx50D4sZyHbK
yNQ9aT4fGHDe7YIvviAwSiMxOfFaSe7twjVeLk

Содержимое файла hash.txt в шестнадцатеричном виде:

6A9A7A5237F5660624DD96FC41027F102F1EF90904E1FED966C7E76D5B17BC

После запуска программы были введены:

– режим работы: 2 (HASH);

– путь к файлу с хэш-кодом: hash.txt;

– путь к файлу, для которого необходимо вычислить хэш-код:
textHASH.txt.

Результат функционирования программы в режиме работы HASH:

Выберите режим работы:

1 - AES;

2 - HASH;

3 - RSA;

4 - SIGNATURE;

5 - CERTIFICATE;

0 - выход.

2

Введите путь к файлу с хэш-кодом: hash.txt

Введите путь к файлу, для которого необходимо вычислить хэш-код:

textHASH.txt

Хэш-код совпадает!

Проверка выполнена успешно.

Содержимое файла hash.txt было изменёно, чтобы проверить работу программы проверки с отрицательным результатом.

Содержимое файла hash.txt в шестнадцатеричном виде:

6A9A7A5237F5660624DD96FC41027F102F1EF90904E1FED96680E76D5B17BC

Все вводимые значения остались такими же.

Результат функционирования программы в режиме работы HASH:

Введите путь к файлу с хэш-кодом: hash.txt

Введите путь к файлу, для которого необходимо вычислить хэш-код:

textHASH.txt

Хэш-код не совпадает!

Программа выводит сообщение, что хэш-код неверен.

8.2.3 Проверка работоспособности приложения в режиме проверки шифрования

RSA

Содержимое файла encryptRSA.txt в шестнадцатеричном виде:

CE201459CAAF7B03E237BAFF6E8244D1E092D2A21CEB467F8D7E8280608777894C
72BA74EA4FD9B52F01DA0082CFF96F397F6122A8757F0DAF2A006D1EE306E7
60C456C8376B230ABFD70B35F3A8671D15D00CE01D0A0C16571622366345AEE4C9
E8F14D4477B593DCB9EB9FC202F6BF80F6E641A4EEE7C0A79E7E19A1A4E9D6
8D20109BB02758F0863FD9F46DEACF93355135CA624B26F62D6735D9FB2B9B5E47
6D63ED49A5FE5B8E1847440D7E3197D2D45A0220EDBA74EB06BA5A1594729C
D829529504103363092DF4E2C6C46BC52AD9BB26CFC7E90BF9FF9256B380133F90
B9E2378D370902CEF5217521206D0FB264B1D18DE4B429A0AD76268230F9A3

Содержимое файла private.pem:

-----BEGIN RSA PRIVATE KEY-----
MIIEowIBAAKCAQEA+ir0Mew+REvyuSit/jocqRjmUnu08d7t7oJ479UWeXuTWP/X
XrOvHkcJ3W7y2FoLxDZY5hRdROtKBaF4Dq7a2dl7ZHTj1wql1k0ihjBP1pC44fV
RbtXB9L/kZp9xyxm4rWkHoF7Oo31Vf/A+uYfWlqrMJVian5PReAdEqSmRBVdKZzu
U/NC/2bJhabNmGuRm8wwh00SkvWneff+5wDEaeah6t6UIZcaLiS+YdRjKLKOOwnke
WqDadel8R0NXx1SF1yUZPVDjRR2Qw9bCUi/t5MPnxb2w4deUrhQ4TglnnzCoRZwP
B5qrdQ/EV1F1Nx/6DiB2SlAS4QZ+Mat+SaE/DwIDAQABAoIBAQUdUSDbHVizQT5I8
PyDAJhfd9fkb7RVyWTWVuulcunlDxPUYBKBGBHwButPoopeMhhhqeNbICrXIR4wyA
g/w+dhgr36XCnofjktYhb/UBalgxp0YUM81P3pIcb0iyBILwlvP6twp3cmDOgu
fcQII9ZfGalCpKq6k0w5gx+ofkcIZVsUB/+K3DSE3qGM8BO59MojTB3UMV2MmBl
UEpqku6LvFfX/OTWucIxVRxa8WdiiJg4Ln79AIK4yQ9H6cKxitNf8qRlpAbfwheY
cMVYylX3P162VYS2eDUul1XnSjNoPsoWP+WLMFO0TOUSXTrNwmGEUML4IjA/3iPX
yFxx3LhAoGBAP7vufGM27h/5IiI8YVpU+GjWMiqOjC3aRShEwM1vm6XV6Y7XU4K
VTuySsJqZg2C1T8P1EUpgj+zomrWTFw3WuMSf1IPsQb4FAhNgIhJdJQz6WMIkBCA
lhJYD2KrIwOF4ZDxR3BuaedmHh5dbEN9kQGCNCeP7BHjHQyy0cmf3UGjAoGBAPs2
InWEefXJDNRCAT5gN/TVoThkUqmN9MyOUTMHkw3QcjRor3upZH3zU8S28kn3mib2
zCVs2sVvWFufutB+Nkj0Krn7OrlekMf4n2tZ0M4JfeX3yo0DmU66YHFJe/3uz1l
ZEWSjZcp9BdKgsUSmYZomITSdVjxUxmjCBZ0mlulAoGAMP+8FuFXyF1tLkezXkC6
LXZYbRCQmZfyP+x0cBC+Dgvzlcj4EEUxA7e17MFgHpK6cdlc15cUjD3JpNmFNFKx
s6G6f+Upzc57G+L4ARhFFCDy2X2bz1WId4ARhHDJ1StCzZt8CjGCyjaXCISYURY4
M0p9vNVxlWO6IwlalhzgW0CgYBwBOvIy5I277gw2fYUAoPxQXHn9ZCG2y5WKA8
aRxCBLLoac4GrrqJD4BDL/FK2oZ66YKPDcNgVw5njQqY/i7myTpkOUKUY3JpjEZL
FI//gYIsriRthVkf+wclPtzz7VwpQ7e67mcFCUg3Opql1m2vaeTZFaAOUU5lGDX7
DXRZoQKBgDaUuER4cvk7YX58buIJTodOGsNB9zErNftZlQ2cxpdFvcRKUDhZUFIg
Hy5snVf5s9+VkiiISPaze2fff+8sAaZDisvm9JjJatlyPvnogU5nXqibL5ztbjjF
WLSVke7g4E4m0qhUCwNxSoHcR/8p3SKX7YgRw/KP/EAILIovBzXi
-----END RSA PRIVATE KEY-----

Содержимое файла plaintextRSA.txt:

hYseiInf5iqftKz2mVSgZwQQyuVIhzetvlh1LvrWuR5b6GIi8dZSPcm2y4WqhryOiR
RfHnS9vDRYOGtXY5r2ngzSGf73MtDZgTcoNrN8QoN6lcxz6GYRzxxl0YDLqcZgR5gC
op8dt9JQYETE4so5SMk5J72YWIhUPVWfK54psHU7mg3W6aiSoAjVgYtBwIMdkvBIZg
JqoX6R4ysJ0Wwq9

После запуска программы были введены:

- режим работы: 3 (RSA);
- путь к файлу с шифротекстом: encryptRSA.txt;
- путь к файлу с закрытым ключом: private.pem;
- путь к файлу с открытым текстом: plaintextRSA.txt;

Результат функционирования программы в режиме работы RSA:

Выберите режим работы:

- 1 - AES;
- 2 - HASH;
- 3 - RSA;
- 4 - SIGNATURE;
- 5 - CERTIFICATE;
- 0 - выход.

3

Введите путь к файлу с шифротекстом: encryptRSA.txt

Введите путь к файлу с закрытым ключом: private.pem

Введите путь к файлу с открытым текстом: plaintextRSA.txt

Расшифрованный текст совпадает с открытым текстом

Проверка выполнена успешно.

Содержимое файла encryptRSA.txt было изменёно, чтобы проверить работу программы проверки с отрицательным результатом.

Содержимое файла encryptRSA.txt в шестнадцатеричном виде:

```
CE201459CAAF7B03E237BAFF6E8244D1E092D2A21CEB467F8D7E8280608777894C
72BA74EA4FD9B52F01DA0082CFF96F397F6122A8757F0DAF2A006D1EE306E7
60C456C8376B230ABFD70B35F3A8671D15D00CE01D0A0C16571622366345AEE4C9
E8F14D4477B59368B9EB9FC202F6BF80F6E641A4EEE7C0A79E7E19A1A4E9D6
8D20109BB02758F0863FD9F46DEACF93355135CA624B26F62D6735D9FB2B9B5E47
6D63ED49A5FE5B8E1847440D7E3197D2D45A0220EDBA74EB06BA5A1594729C
D829529504103363092DF4E2C6C46BC52AD9BB26CFC7E90BF9FF9256B380133F90
B9E2378D370902CEF5217521206D0FB264B1D18DE4B429A0AD76268230F9A3
```

Все вводимые значения остались такими же.

Результат функционирования программы в режиме работы RSA:

Введите путь к файлу с шифротекстом: encryptRSA.txt

Введите путь к файлу с закрытым ключом: private.pem

Ошибка расшифрования сообщения!

Программа выводит сообщение, что шифротекст неверен.

8.2.4 Проверка работоспособности приложения в режиме проверки электронной подписи с помощью открытого ключа

Содержимое файла messageSIGN.txt:

```
Dd6bn16Cl0cz9o4vsX7txjhdGUKve7PyKCfyRgxmnQ5917sFdZtfnNWXtmdcJR1xOQu
nLHKWxtLv4v4wa6iuWW52ANZmTO4jAmdUheBuPdUAKT8aGCu0wzjHLQYNa6ZLGjCZb
I6Y7tyCJt61xUmByp4QVGXPXYWmJeIVocyvqdy6f4vZVHuvPwssmF2F77Df5pCYAajn
Zz3J3TOup2j9XPrgKh84kAs3L2TV4Tl3VFoiRxsbiX3AWD66oi7kZxQXbdfRefZtZ
dgG3p6BxGPwkO3aEApDl8dQWPRXe2D9zMxnAveqX4EUWa6yRQFJMX3vgY1mkPz164k
```

```
62w0yiZe864zV1VoHmuBdf2ipQof4jA52yjQBdpJNi643yOWw4GQXKI1A60ihIewGX
9JqWm3pEMZxpDz5Vy7yJNLVIEqPJT211GPcx6PmWNk1L5CZUj82YgV5PX1XuMiXgZx
7fN7ME1cFLUldhGSsHga37vTlezDaJpcCUXDjHga9b3hGvZtQK08lqYNpYShAy6X4a
QUkFpieeMNHSNq9qTd4Od3GJVWGXlgoNfbKc0z8Rh0wJy4UM37uKck5VjviF9Fzqu
CqzffeE4OpDtDNbJkU80A5jrTwvCKIEckec8xjtWmpn5JYdGa4hnXqbOL34YERYRnt
swfKCCUpdqem8hQT17xqZ90Ppw8FUShaWWMai6PnMNzpy3qKTAtEs4U66VQRnPr2ZG
68iR6efi5DMTDjbcZvjxU4KjopYZWVRL9b80pVoPubzlYvd9A1yb5VitFRdReFzSPe
BstGVYe6hxlZvTlFvR9a45J5V2bqAfuyRVIfrcVnAjcn04mwjYbJyzFnUd3iJUqOyZ
xcjLbwIUUjcyZL1znENxMiqPValUnpHD7E7Y4Vm4KRJhBY7nB0zl3y2Edq626bbRb8
pcNRZwim4P3zv75pnQ6VtVERpD3Iqt1ZbvKaniN6ELZ4UlUkWoqdWlup8FXkaQIWFC
ODc3qeBMju7rF1yioQIEPjMMjbfRnBNrNz
```

Содержимое файла public.pem:

```
-----BEGIN RSA PUBLIC KEY-----
MIIBCgKCAQEA+ir0Mew+REvyuSIt/jocqRjmUnu08d7t7oJ479UWeXuTWP/XXrOv
HkcNJ3W7y2FoLxDZY5hRdROtKBaF4Dq7a2dl7ZHTj1wql1k0ihjBP1pC44fVRbtX
B9L/kZp9xyxm4rWkHoF7Oo31Vf/A+uYfWlqrMJVian5PReAdEqSmRBVdKZzuU/NC
/2bJhabNmGuRm8wwh00SkvWneff+5wDEaeahT6UIZcaLiS+YdRjKLK0OwnkeWqDa
del8R0NXxlSF1yUZPVDjRR2Qw9bCUi/t5MPnxb2w4deUrhQ4TglnnzCoRZwPB5qr
dQ/EVlF1Nx/6DiB2SlAS4QZ+Mat+SaE/DwIDAQAB
-----END RSA PUBLIC KEY-----
```

Содержимое файла signature.sig в шестнадцатеричном виде:

```
E8D362F7742A3C60FA8BBA0A8218A7A0FE8C86B559A9D3F2E386097E961DF683EE
C9779B2EAC039EDE732598C87F33EEB33AC84A4D25C7E4496AF9822EC63653
B29C32F21FCF6F79C7BD82021E0B41507F47D66B265DD61321BECAD170E19C2EA8
101C8243D2A9E9006F75096BFC974E54D125EE9DF34059951C8280B7D3D94E
542200FBFFA84BF42B1B880F57D3798AF7D15EF33B051CB28E8D8C3FF8DC2E94F3
4A7125D9B799253FD74DA848FE6FC4B3B48C078E5D599DD36FF184A3EBE8EE
4496E04375B3B5D8D0388BBA61D81393EB2136C0C3EF555B049D850D1F3824787D
EF12EB453A1DB8E598B2974EA7030731A5398BDCFB2C009B77A19D87D4B4A3
```

После запуска программы были введены:

- режим работы: 4 (SIGNATURE);
- путь к файлу с информацией: messageSIGN.txt;
- путь к файлу с открытым ключом: public.pem;
- путь к файлу с электронной подписью: signature.sig.

Результат функционирования программы в режиме работы SIGNATURE:

Выберите режим работы:

- 1 - AES;
- 2 - HASH;
- 3 - RSA;
- 4 - SIGNATURE;
- 5 - CERTIFICATE;
- 0 - выход.

4

Введите путь к файлу с информацией: messageSIGN.txt

Введите путь к файлу с открытым ключом: public.pem

Введите путь к файлу с электронной подписью: signature.sig
Подпись верифицирована!

Проверка выполнена успешно.

Содержимое файла signature.sig было изменено, чтобы проверить работу программы проверки с отрицательным результатом.

Содержимое файла signature.sig в шестнадцатеричном виде:

```
E8D362F7742A3C60FA8BBA0A8218A7A0FE8C86B559A9D3F2E386097E961DF683EE
C9779B2EAC039EDE732598C87F33EEB33AC84A4D25C7E4496AF9822EC63653
B29C32F21FCF6F79C7BD82021E0B41507F47D66B265DD61321BECAD170E19C2EA8
101C8243D2A9E9006F75096BFC974E54D125EE9DF34042951C8280B7D3D94E
542200FBFFA84BF42B1B880F57D3798AF7D15EF33B051CB28E8D8C3FF8DC2E94F3
4A7125D9B799253FD74DA848FE6FC4B3B48C078E5D599DD36FF184A3EBE8EE
4496E04375B3B5D8D0388BBA61D81393EB2136C0C3EF555B049D850D1F3824787D
EF12EB453A1DB8E598B2974EA7030731A5398BDCFB2C009B77A19D87D4B4A3
```

Все вводимые значения остались такими же.

Результат функционирования программы в режиме работы RSA:

Введите путь к файлу с информацией: messageSIGN.txt
Введите путь к файлу с открытым ключом: public.pem
Введите путь к файлу с электронной подписью: signature.sig
Проверка подписи не удалась.

Программа выводит сообщение, что проверка подписи не удалась.

8.2.5 Проверка работоспособности приложения в режиме проверки электронной подписи с помощью сертификата

Содержимое файла messageSIGN.txt:

```
Dd6bn16C10cz9o4vsX7txjhdGUKve7PyKCfyRgxmnQ5917sFdZtfNWXTmdcJR1xOQu
nLHKWxtLv4v4wa6iuWW52ANZmTO4jAmdUheBuPdUAKT8aGCu0wzjHLQYNa6ZLGjCZb
I6Y7tyCJt61xUmByp4QVGpXYWmJeIVocyvqdy6f4vZVHuvPwssmF2F77Df5pCYAajn
Zz3J3TOup2j9XPrghKh84kwaS3L2TV4Tl3VFoiRxsbiX3AWD66oi7kZxQXbdfRefZtZ
dgG3p6BxGPwkO3aEApDl8dQWPRXe2D9zMXnAveqX4EUWa6yRQFJMX3vgYlmpZl64k
62w0yize864zV1VoHmuBdf2ipQof4jA52yjQBdpJNi643yOWw4GQXKI1A60ihIewGX
9JqWm3pEMZxpDz5Vy7yJNLViEqPJT2l1GPcx6PmWNk1L5CZUj82YgV5PX1XuMiXgZx
7fN7ME1cFLUldhGSsHga37vTlezDaJpcCUXDjHga9b3hGvZtQK08lqYNpYShAy6X4a
QUkFpieeMNHSNq9qTd4Od3GJVUWGXIgoNfbKc0z8Rh0wJy4UM37uKcK5VjviF9Fzqu
CqzfFeE4OpDtDNbJkU8OA5jrTwvCKIEckec8xjtWmpn5JYdGa4hnXqbOL34YERYRnt
swfKCCUpdqem8hQT17xqZ90Ppw8FUShaWWmai6PnMNzpy3qKTAtEs4U66VQRnPr2ZG
68iR6efi5DMTDjbcZvjxU4KjopYZWVRL9b80pVoPubzlyvd9Alyb5VitFRdReFzSPe
BstGVYe6hxlZvTlFvR9a45J5V2bqAfuyRVIfrCvNAjcn04mwjYbJyzFnUd3iJUqOyZ
xcjLbwIUUjcyZL1znENxMiqPValUnpHD7E7Y4Vm4KRJhBY7nB0zl3y2Edq626bbRb8
pcNRZwim4P3zv75pnQ6VtVERpD3Iqt1ZbvKaniN6ELZ4U1UkWoqdWlup8FXkaQIWFC
ODc3qeBMju7rF1yioQIEPjMMjbfRnBNrNz
```

Содержимое файла certificate.crt:

```
-----BEGIN CERTIFICATE-----
MIIClzCCAX+gAwIBAgIBATANBgkqhkiG9w0BAQsFADAAMB4XDTIzMDUwOTA4NDZl
MVoXDTI0MDUwODA4NDZlMVowHjEcmBoGA1UEAwwTTGFiV29yayBTZXJ0aWZpY2F0
ZTCCASIwDQYJKoZIhvcNAQEBBQADggEPADCCAQoCggEBAPoq9DHsPkRL8rkiLf46
HKkY5lJ7tPHe7e6CeO/VFn17klj/116zrx5HJydlu8thaC8Q2WOYUXUTrSgWheA6
u2tnZe2R049cKpdZNIoYwT9aQuOH1UW7cQfS/5GafccsZuK1pB6BezqN9VX/wPrm
H1paqzCVSGp+T0XgHRKkpkQVXSmc7lPzQv9myYWmzZhrkZvMMIdNEpL1p3n3/ucA
xGnmobelCGXGi4kvmHUYyiyjjsJ5Hlqg2nXpfEdDV8ZUhdclGT1Q40UdkMPWwlIv
7eTD58W9sOHXlK4UOE4JZp8wqEWcDweaq3UPxFZRdTcf+g4gdkpQEuEGfjGrfkmh
Pw8CAwEAATANBgkqhkiG9w0BAQsFAAOCAQEAAuCK3hdHqmk5WEp+CpC2V7L5NZOr7
NVQ1soqWwP+Vd8+ulMoVoDw3SbG5/7KmanaPKzSQFjL2BZ45YDiZb6GCFIapbkLP
KESSRG+Z8eg5BlRTH2vRC6BV3XgPL3d8QXOrxcEr8nL5lip7mydstfA5Did+eNGI
Y4c8CuC7R605qNex0/fKlmQrEQLYDqkKnLHiwE9wxpohHkqWcYOZM/Kstzfedo+C
1i8eY4hrys/c3JdXHg8kFXEqK+cCSelmVWHjw+TlNjRrCh6XQk8hoMoaMmtKeilL8
6cfgARsTazgDsOIeuzXcLkv/vr97aFt+P7nVdccZh/0WzSp0+qdJEZiptw==
-----END CERTIFICATE-----
```

Содержимое файла signature.sig в шестнадцатеричном виде:

```
E8D362F7742A3C60FA8BBA0A8218A7A0FE8C86B559A9D3F2E386097E961DF683EE
C9779B2EAC039EDE732598C87F33EEB33AC84A4D25C7E4496AF9822EC63653
B29C32F21FCF6F79C7BD82021E0B41507F47D66B265DD61321BECAD170E19C2EA8
101C8243D2A9E9006F75096BFC974E54D125EE9DF34059951C8280B7D3D94E
542200FBFFA84BF42B1B880F57D3798AF7D15EF33B051CB28E8D8C3FF8DC2E94F3
4A7125D9B799253FD74DA848FE6FC4B3B48C078E5D599DD36FF184A3EBE8EE
4496E04375B3B5D8D0388BBA61D81393EB2136C0C3EF555B049D850D1F3824787D
EF12EB453A1DB8E598B2974EA7030731A5398BDCFB2C009B77A19D87D4B4A3
```

После запуска программы были введены:

- режим работы: 5 (CERTIFICATE);
- путь к файлу с информацией: messageSIGN.txt;
- путь к файлу с сертификатом: certificate.crt;
- путь к файлу с электронной подписью: signature.sig.

Результат функционирования программы в режиме работы CERTIFICATE:

- 1 - AES;
- 2 - HASH;
- 3 - RSA;
- 4 - SIGNATURE;
- 5 - CERTIFICATE;
- 0 - выход.

```
5
Введите путь к файлу с информацией: messageSIGN.txt
Введите путь к файлу с сертификатом: certificate.crt
Введите путь к файлу с электронной подписью: signature.sig
Подпись верифицирована!
```

Проверка выполнена успешно.

Содержимое файла signature.sig было изменёно, чтобы проверить работу программы проверки с отрицательным результатом.

Содержимое файла signature.sig в шестнадцатеричном виде:

```
E8D362F7742A3C60FA8BBA0A8218A7A0FE8C86B559A9D3F2E386097E961DF683EE
C9779B2EAC039EDE732598C87F33EEB33AC84A4D25C7E4496AF9822EC63653
B29C32F21FCF6F79C7BD82021E0B41507F47D66B265DD61321BECAD170E19C2EA8
101C8243D2A9E9006F75096BFC974E54D125EE9DF34042951C8280B7D3D94E
542200FBFFFA84BF42B1B880F57D3798AF7D15EF33B051CB28E8D8C3FF8DC2E94F3
4A7125D9B799253FD74DA848FE6FC4B3B48C078E5D599DD36FF184A3EBE8EE
4496E04375B3B5D8D0388BBA61D81393EB2136C0C3EF555B049D850D1F3824787D
EF12EB453A1DB8E598B2974EA7030731A5398BDCFB2C009B77A19D87D4B4A3
```

Все вводимые значения остались такими же.

Результат функционирования программы в режиме работы RSA:

```
Введите путь к файлу с информацией: messageSIGN.txt
Введите путь к файлу с сертификатом: certificate.crt
Введите путь к файлу с электронной подписью: signature.sig
Проверка подписи не удалась.
```

Программа выводит сообщение, что проверка подписи не удалась.

9 Вредные психофизиологические факторы и методы их снижения в работе специалиста по защите информации

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

Мышечно-скелетные нарушения у работающих за ЭВМ приводят к нарушению трудоспособности и нарушению жизненной активности. Рабочая поза за компьютером во время рабочей смены является для пользователей не физиологической и вызывает определенные нарушения: застой крови, мышечные напряжения, в последующем – дегенеративные изменения позвоночника. Важной проблемой для здоровья пользователей компьютеров является статичность позы. Длительная неподвижность может способствовать формированию мышечно-скелетных нарушений [8].

При плохой осанке или «несущественной» дисплазии скелета даже обычные статические нагрузки могут оказаться чрезмерными, способными вызвать заболевание опорно-двигательной системы. Монотонные, стереотипные движения при работе с клавиатурой способствуют перегрузке суставов верхних конечностей; работа с манипулятором вызывает хроническое растяжение сухожилий. Кроме того, патология формируется вследствие постоянного постукивания подушечками пальцев по клавиатуре и нефизиологического положения кисти [8].

Боль в суставах (артралгия) профессионального, или нет, происхождения возникает не сразу. Возникновение боли – сигнал о том, что начался асептический воспалительный процесс. В целом для костно-мышечной патологии характерен механический ритм болей – возникновение боли под влиянием рабочей нагрузки и стихание за период отдыха [8].

9.2 Методы и средства физической культуры, снижающие воздействие психофизиологических факторов

Состояние позвоночника для специалиста по защите информации, как впрочем, и для любого человека играет важную роль. Поэтому необходимо уделять большое внимание осанке.

Основное упражнение для осанки.

Встаньте спиной к стене, ноги слегка расставлены, руки свободно опущены. Затылок, плечи, икры, и пятки касаются стены. А теперь постарайтесь прислониться к стене так, чтобы расстояние между стеной и поясницей было не больше толщины пальца. Подберите живот, вытяните немного шею вверх и разверните плечи. Проанализируйте "чувства" всех частей тела, особенно мускулов спины и живота. Другими словами, начните программирование своего позвоночного компьютера в положении нормальной осанки. Выполняйте это упражнение как можно чаще в течение дня. Как только вы сможете удерживать такое положение тела у стены в течение минуты без утомления, тогда, сохраняя ту же осанку, идите вперед [9]!

Приступая к выполнению оздоровительных упражнений, следует руководствоваться следующими правилами:

- не прилагать резких усилий к заостеневшим местам;
- выполнять упражнения, соизмеряя нагрузки со своими физическими возможностями;
- не стремиться выполнять упражнения с максимальной амплитудой движения.

Вначале надо делать каждое упражнение не более двух-трех раз. Через день можно увеличить до пяти раз и более. Учтите, что изменения в позвоночнике происходили в течении многих лет и нельзя ничего исправить за один день. Только постоянная тренировка будет стимулировать рост хряща и сделает позвоночник растянутым и эластичным [9].

Оздоровительные упражнения:

– вращение головой опустите подбородок на грудь и поворачивайте голову так, чтобы ухо коснулось плеча, затылок-спины, другое ухо-второго плеча, подбородок груди. Делайте это упражнение медленно, растягивая шейные мускулы и позвонки. Вращайте голову 20 раз в одну сторону и столько же в другую;

– растяжение шеи и усиление верхней части спины. Встаньте в положение правильной осанки, раздвиньте ноги на ширину ступни, расслабьте мускулы. Сомкните руки за головой, наклоните голову вперед, а затем попытайтесь вернуть ее в исходное положение, оказывая сопротивление руками. Делайте это в течении 6 сек. Повторите упражнение, откинув голову назад, наклонив влево и вправо;

– усиление верхней части позвоночника. Поднимите плечи как можно выше и плавно отведите их назад, а затем вперед как можно дальше. Повторите это упражнение 15 раз. После небольшой паузы сделайте эти же движения 15 раз в обратном направлении;

– усиление всего позвоночника. Встаньте прямо, ноги на ширине плеч, руки опущены и совершенно расслаблены. Поворачивайтесь всем телом то в одну, то в другую сторону, стараясь заглянуть через плечо как можно дальше;

– развитие гибкости позвоночника. Встаньте прямо, ноги вместе. Поднимите прямые руки над головой. Нагнитесь вперед и попытайтесь пальцами рук коснуться пальцев ног, ноги не сгибайте. Вернитесь в исходное положение и затем с поднятыми руками отклонитесь назад как можно дальше, руки и голова также откинута назад. Вернитесь в исходное положение [9].

Заключение

Криптографические библиотеки предоставляют множество инструментов для защиты данных и информации, поэтому важно уметь использовать их возможности для создания качественных приложений.

Умение использовать криптографические библиотеки – необходимый навык для любого специалиста по информационной безопасности.

В результате выпускной квалификационной работы были выполнены следующие задачи:

- выполнен обзор криптографических библиотек в операционных системах семейства Linux;
- выполнен анализ возможностей криптографических библиотек;
- разработана структура практикума;
- разработан порядок проведения практикума;
- разработаны рекомендации по проведению практикума;
- разработаны варианты и контрольные примеры для 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; методы и средства физической культуры, снижающие их воздействие.

Поставленные задачи решены в полном объёме. Таким образом, техническое задание выполнено полностью.

Список использованных источников

1. Botan: Crypto and TLS for Modern C++ — Botan : [сайт]. URL: <https://botan.randombit.net/> (дата обращения: 15.02.2023, 11.03.2023).
2. Cryptlib Encryption Toolkit : [сайт]. URL: <https://www.cs.auckland.ac.nz/~pgut001/cryptlib/> (дата обращения: 13.02.2023, 12.03.2023).
3. Crypto++ Library 8.7 | Free C++ Class Library of Cryptographic Schemes : [сайт]. URL: <https://www.cryptopp.com/> (дата обращения: 14.02.2023, 03.03.2023).
4. GnuPG – Libraries – Libgcrypt : [сайт]. URL: <https://www.gnupg.org/software/libgcrypt/index.html> (дата обращения: 23.02.2023, 02.03.2023).
5. Nettle: a low-level cryptographic library : [сайт]. URL: <https://www.lysator.liu.se/~nisse/nettle/nettle.html> (дата обращения: 25.02.2023, 09.03.2023).
6. OpenSSL Cryptography and SSL/TLS Toolkit : [сайт]. URL: <https://www.openssl.org/docs/manmaster/man1/openssl.html> (дата обращения: 24.02.2023, 01.03.2023).
7. WolfCrypt Embedded Cryptography Engine | wolfSSL Products : [сайт]. URL: <https://www.wolfssl.com/products/wolfcrypt-2/> (дата обращения: 17.02.2023, 03.03.2023).
8. Костно-мышечные нарушения у работающих за компьютером / Е.М. Власова, В.Б. Алексеев, Н.Н. Малютина, В.А. Хорошавин – Текст : электронный // Медицина труда и промышленная экология. – 2011. – №11. – ISSN 1026-9428. – URL: <https://cyberleninka.ru/article/n/kostno-myshechnye-narusheniya-u-rabotayuschih-za-kompyuterom> (дата обращения: 09.05.2023, 19.05.2023). – Режим доступа: Научная электронная библиотека Киберленинка.
9. Суворов, Ю. А. Профессионально-прикладная физическая подготовка

студентов : учеб.-метод. Пособие / Ю. А. Суворов, В. А. Платонова. – Санкт-Петербург : СПб ГУИТМО, 2006. – 90 с. – ISBN 5-88306-314-2.

Приложение А
(обязательное)

Коды примеров реализации практикума

Код программы шифрования AES-256 GCM:

```
#include <openssl/evp.h>
#include <string.h>
#include <iostream>
#include <fstream>

using namespace std;

// Функция зашифрования
int encrypt(unsigned char *aad, int aad_len, unsigned char *key,
            unsigned char *iv,
            unsigned char *ciphertext, unsigned char *tag)
{
    string path;
    // Получение пути к файлу
    cout << "Введите путь к файлу: ";
    //cin >> path;
    path = "plaintext.txt";

    string text;
    string kont;
    ifstream file(path);
    file.seekg(0, ios::end);
    int size = file.tellg();
    file.seekg(0);
    unsigned char plaintext[size]= {};
    int i = 0;
    while(getline(file, text)) {
        if (i > 0)
            kont += '\n';
        i = 1;
        kont += text;
    }
    file.close();
    strcpy((char*)plaintext, kont.c_str());
    int plaintext_len = strlen((char *) plaintext);

    cout << "Исходный текст:\n" << plaintext << endl;
    EVP_CIPHER_CTX *ctx;
    int len;
    int ciphertext_len;

    // Создание и инициализация контекста
    if (!(ctx = EVP_CIPHER_CTX_new())) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка алгоритма шифрования
    if (1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_gcm(), NULL,
```

```

NULL, NULL)) {
    EVP_CIPHER_CTX_free(ctx);
    return -1;
}

// Установка ключа и IV
if (1 != EVP_EncryptInit_ex(ctx, NULL, NULL, key, iv)) {
    EVP_CIPHER_CTX_free(ctx);
    return -1;
}

// Установка AAD (дополнительные данные аутентификации)
if (aad && aad_len > 0) {
    if (1 != EVP_EncryptUpdate(ctx, NULL, &len, aad, aad_len))
    {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }
}

// Зашифрование данных
if (1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext,
plaintext_len)) {
    EVP_CIPHER_CTX_free(ctx);
    return -1;
}
ciphertext_len = len;

// Завершение шифрования и получение тега
if (1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len)) {
    EVP_CIPHER_CTX_free(ctx);
    return -1;
}
ciphertext_len += len;

if (1 != EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16,
tag)) {
    EVP_CIPHER_CTX_free(ctx);
    return -1;
}

// Запись тега и шифротекста в файлы
FILE *tag_file = fopen("tag.txt", "w");
BIO_dump_fp(tag_file, (const char *) tag, 16);
FILE *cipher_file = fopen("cipher.txt", "w");
BIO_dump_fp(cipher_file, (const char *) ciphertext,
ciphertext_len);

// Освобождение контекста
EVP_CIPHER_CTX_free(ctx);

```

```

    return ciphertext_len;
}

// Функция расшифрования
int decrypt(unsigned char *ciphertext, int ciphertext_len,
            unsigned char *aad, int aad_len,
            unsigned char *tag,
            unsigned char *key,
            unsigned char *iv,
            unsigned char *plaintext)
{
    EVP_CIPHER_CTX *ctx;
    int len;
    int plaintext_len;
    int ret;

    // Создание и инициализация контекста
    if (!(ctx = EVP_CIPHER_CTX_new())) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка алгоритма шифрования
    if (!EVP_DecryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL,
NULL)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка ключа и IV
    if (!EVP_DecryptInit_ex(ctx, NULL, NULL, key, iv)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка AAD (дополнительные данные аутентификации)
    if (aad && aad_len > 0) {
        if (!EVP_DecryptUpdate(ctx, NULL, &len, aad, aad_len)) {
            EVP_CIPHER_CTX_free(ctx);
            return -1;
        }
    }

    // Расшифрование данных
    if (!EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext,
ciphertext_len)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }
    plaintext_len = len;
}

```

```

// Установка тега
if (!EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_TAG, 16, tag))
{
    EVP_CIPHER_CTX_free(ctx);
    return -1;
}

// Завершение расшифрования
ret = EVP_DecryptFinal_ex(ctx, plaintext + len, &len);

// Освобождение контекста
EVP_CIPHER_CTX_free(ctx);

if (ret > 0) {
    plaintext_len += len;
    plaintext[plaintext_len] = '\0';
    // Открытие файла для записи расшифрованного текста
    ofstream decrypt_file("decrypt.txt");
    if (decrypt_file.is_open()) {
        decrypt_file << plaintext;
        decrypt_file.close();
    }
    return plaintext_len;
} else {
    return -1;
}
}

int main()
{
    unsigned char ciphertext[1280];
    unsigned char decryptedtext[1280];
    unsigned char tag[16];
    //unsigned char key[] = "HDtTfHff4Z3VFPTYpgHcYmaBblQSwtaj";
    //unsigned char iv[] = "k95U7ctB1pRonE62";
    //unsigned char aad[] = "Reference_example_variant1";
    int decryptedtext_len, ciphertext_len;

    unsigned char key[33];
    unsigned char iv[17];
    unsigned char* aad = new unsigned char[50];

    cout << "Введите ключ: ";
    cin >> key;
    cout << "Введите вектор инициализации: ";
    cin >> iv;
    cout << "Введите дополнительные данные аутентификации: ";
    cin >> aad;

    int mode = 1;
    while (mode != 0) {

```

```

        cout << "Выберите режим работы:\n1 - зашифрование;\n2 -
расшифрование;\n0 - выход.\n";
        cin >> mode;

        switch (mode) {
        case 1:
            // Зашифрование данных
            ciphertext_len = encrypt(aad, strlen((char *) aad),
key, iv, ciphertext, tag);
            cout << "Тег аутентификации:\n";
            BIO_dump_fp(stdout, (const char *) tag, sizeof(tag));
            cout << "Зашифрованный текст:\n";
            BIO_dump_fp(stdout, (const char *) ciphertext,
ciphertext_len);
            break;
        case 2:
            // Расшифрование данных
            decryptedtext_len = decrypt(ciphertext,
ciphertext_len, aad, strlen((char *) aad), tag, key, iv,
decryptedtext);
            // Добавление нулевого символа в конец расшифрованного
текста
            decryptedtext[decryptedtext_len] = '\0';
            cout << "Расшифрованный текст:\n" << decryptedtext <<
endl;
            break;
        default:
            break;
        }
    }

    delete[] aad;
    return 0;
}

```

Код программы вычисления хэш-функции SHA-256:

```

#include <iostream>
#include <fstream>
#include <cstring>
#include <openssl/sha.h>

using namespace std;

// Вычисление хэш-кода
int calcHash (unsigned char *hash)
{
    string text, kont;
    string path;

    // Получение пути к файлу
    cout << "Введите путь к файлу: ";

```

```

cin >> path;
//path = "text.txt";

// Открытие и чтение информации из файла
ifstream file(path, ios::binary);
int i = 0;
while(getline(file, text)) {
    if (i > 0)
        kont += '\n';
    i = 1;
    kont += text;
}
file.close();

// Вычисление хэш-кода
SHA256_CTX sha256;
SHA256_Init(&sha256);
SHA256_Update(&sha256, kont.c_str(), kont.size());
SHA256_Final(hash, &sha256);
return 0;
}

// Проверка хэш-кода
int checkHash(unsigned char *hash)
{
    string path;
    // Получение пути к файлу
    cout << "Введите путь к файлу с хэш-кодом: ";
    cin >> path;
    //path = "hash.txt";

    // Чтение хэш-кода из файла
    ifstream hashFile2(path, ios::binary);
    unsigned char hash2[SHA256_DIGEST_LENGTH];
    hashFile2.read((char*)hash2, SHA256_DIGEST_LENGTH);

    calcHash(hash);

    // Проверка хэш-кодов
    if (memcmp(hash, hash2, SHA256_DIGEST_LENGTH) == 0) {
        cout << "Хэш-код совпадает!" << endl << endl;
    } else {
        cout << "Хэш-код не совпадает!" << endl << endl;
    }
    return 0;
}

int main()
{
    unsigned char hash[SHA256_DIGEST_LENGTH];

```

```

int mode = 1;

while (mode != 0) {
    cout << "Выберите режим работы:\n1 - вычисление хэш-кода;\n2 - проверка хэш-кода;\n0 - выход.\n";
    cin >> mode;

    switch (mode) {
    case 1: {
        // Вычисление хэша
        calcHash(hash);
        cout << "Хэш-код: ";
        for (int i = 0; i < SHA256_DIGEST_LENGTH; i++) {
            printf("%02x", hash[i]);
        }
        cout << endl << endl;
        // Запись хэша в файл
        ofstream hashFile("hash.txt", ios::binary);
        hashFile.write((char*)hash, SHA256_DIGEST_LENGTH);
        hashFile.close();
        break;
    }
    case 2:
        // Проверка хэша
        checkHash(hash);
        break;
    default:
        break;
    }
}
return 0;
}

```

Код программы шифрования и генерации ключей RSA:

```

#include <iostream>
#include <fstream>
#include <openssl/rsa.h>
#include <openssl/pem.h>

using namespace std;

int generateKey()
{
    // Генерация ключевой пары
    RSA *keypair = RSA_new();
    BIGNUM *e = BN_new();
    BN_set_word(e, RSA_F4);
    RSA_generate_key_ex(keypair, 2048, e, NULL);

    if (!keypair) {
        cout << "Ошибка генерации ключевой пары" << endl;
    }
}

```

```

        BN_free(e);
        RSA_free(keypair);
        return 1;
    }

    // Получение открытого и закрытого ключей
    BIO *priv = BIO_new(BIO_s_mem());
    BIO *publ = BIO_new(BIO_s_mem());
    PEM_write_bio_RSAPrivateKey(priv, keypair, NULL, NULL, 0,
    NULL, NULL);
    PEM_write_bio_RSAPublicKey(publ, keypair);

    // Получение длины ключа
    int priv_len = BIO_pending(priv);
    int publ_len = BIO_pending(publ);

    // Чтение ключей в буфер
    char *priv_key = new char[priv_len + 1];
    char *publ_key = new char[publ_len + 1];
    BIO_read(priv, priv_key, priv_len);
    BIO_read(publ, publ_key, publ_len);
    priv_key[priv_len] = '\0';
    publ_key[publ_len] = '\0';

    // Запись ключей в файлы
    FILE* private_key_file = fopen("private.pem", "wb");
    PEM_write_RSAPrivateKey(private_key_file, keypair, NULL, NULL,
    0, NULL, NULL);
    fclose(private_key_file);

    FILE* public_key_file = fopen("public.pem", "wb");
    PEM_write_RSAPublicKey(public_key_file, keypair);
    fclose(public_key_file);

    // Вывод ключей
    cout << "Закрытый ключ:" << endl;
    cout << priv_key << endl;
    cout << "Открытый ключ:" << endl;
    cout << publ_key << endl;
    delete[] priv_key;
    delete[] publ_key;
    BN_free(e);
    BIO_free_all(priv);
    BIO_free_all(publ);
    RSA_free(keypair);
    return 0;
}

int encrypt()
{
    string plaintext;

```

```

string text;
string path;
// Получение пути к файлу
cout << "Введите путь к файлу с открытым текстом: ";
cin >> path;
//path = "plaintext.txt";

// Считывание открытого текста
ifstream file(path, ios::binary);
int i = 0;
while(getline(file, text)) {
    if (i > 0)
        plaintext += '\n';
    i = 1;
    plaintext += text;
}
file.close();
// Получение пути к файлу
cout << "Введите путь к файлу с открытым ключом: ";
cin >> path;
//path = "public.pem";
FILE* fp = fopen(path.c_str(), "r");
if (!fp) {
    cerr << "Ошибка открытия файла с открытым ключом." <<
endl;
    return 1;
}
RSA *rsapubl = PEM_read_RSAPublicKey(fp, NULL, NULL, NULL);
fclose(fp);

// Зашифрование
unsigned char *encrypted = new unsigned
char[RSA_size(rsapubl)];
int encrypted_length = RSA_public_encrypt(plaintext.length() +
1,
                                (unsigned char *) plaintext.c_str(),
                                encrypted,
                                rsapubl,
                                RSA_PKCS1_OAEP_PADDING);
if (encrypted_length == -1) {
    cout << "Ошибка шифрования сообщения" << endl;
    delete[] encrypted;
    RSA_free(rsapubl);
    return 1;
}

fp = fopen("encrypt.txt", "wb");
// Вывод зашифрованного сообщения и запись в файл
cout << "Зашифрованное сообщение:" << endl;
for (int i = 0; i < encrypted_length; ++i) {
    printf("%02x", encrypted[i]);
}

```

```

        fwrite(&encrypted[i], sizeof(encrypted[i]), 1, fp);
    }
    cout << endl;
    fclose(fp);
    delete[] encrypted;
    RSA_free(rsapubl);
    return 0;
}

int decrypt()
{
    string path;
    // Получение пути к файлу
    cout << "Введите путь к файлу с шифротексту: ";
    cin >> path;
    //path = "encrypt.txt";

    // Чтение шифротекста из файла
    ifstream file(path, ios::binary);
    file.seekg(0, file.end);
    int length = file.tellg();
    file.seekg(0, file.beg);
    unsigned char *encrypted = new unsigned char[length];
    file.read((char *)encrypted, length);
    file.close();

    // Получение пути к файлу
    cout << "Введите путь к файлу с закрытым ключом: ";
    cin >> path;
    //path = "private.pem";

    //Загрузка ключей
    FILE* fp = fopen(path.c_str(), "r");
    if (!fp) {
        cerr << "Ошибка открытия файла с закрытым ключом." <<
endl;
        delete[] encrypted;
        return 1;
    }
    RSA *rsapriv = PEM_read_RSAPrivateKey(fp, NULL, NULL, NULL);
    fclose(fp);

    // Расшифрование сообщения
    unsigned char *decrypted = new unsigned char[length];
    int decrypted_length = RSA_private_decrypt(length, encrypted,
decrypted, rsapriv, RSA_PKCS1_OAEP_PADDING);
    if (decrypted_length == -1) {
        cout << "Ошибка расшифрования сообщения" << endl;
        delete[] encrypted;
        delete[] decrypted;
        return 1;
    }
}

```

```

    }

    // Вывод расшифрованного сообщения
    cout << "Расшифрованное сообщение:" << endl;
    cout << decrypted << endl;
    fp = fopen("decrypt.txt", "wb");

    // Запись в файл расшифрованного сообщения
    ofstream File("decrypt.txt", ios::binary);
    File.write((char*)decrypted, decrypted_length-1);
    File.close();

    // Очистка памяти
    delete[] encrypted;
    delete[] decrypted;
    return 0;
}

int main()
{
    int mode = 3;
    while (mode != 0) {
        cout << "Выберите режим работы:\n1 - генерация ключевой
пары;\n2 - зашифрование;\n3 - расшифрование;\n0 - выход.\n";
        cin >> mode;

        switch (mode) {
            case 1: {
                generateKey();
                break;
            }
            case 2: {
                encrypt();
                break;
            }
            case 3: {
                decrypt();
                break;
            }
        }
    }
}

```

Код программы для вычисления и проверки электронной подписи:

```

#include <openssl/pem.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstring>

using namespace std;

```

```

int calculateSignature ()
{
    string message;
    string path;
    stringstream buffer;
    // Получение пути к файлу
    cout << "Введите путь к файлу с информацией: ";
    cin >> path;
    // path = "message.txt";

    // Чтение текста для подписи
    ifstream file(path, ios::binary);
    if (file.is_open()) {
        buffer.str("");
        buffer.clear();
        buffer << file.rdbuf();
        message = buffer.str();
        file.close();
    } else {
        cerr << "Не удалось открыть файл." << endl;
        return 1;
    }

    cout << "Введите путь к файлу с закрытым ключом: ";
    cin >> path;
    //path = "private.pem";

    // Загрузка закрытого ключа
    FILE* fp = fopen(path.c_str(), "r");
    if (!fp) {
        cerr << "Ошибка открытия файла с закрытым ключом." <<
endl;
        return 1;
    }
    RSA* rsa = PEM_read_RSAPrivateKey(fp, NULL, NULL, NULL);
    fclose(fp);
    // Создание объекта EVP_PKEY из объекта RSA
    EVP_PKEY* privkey = EVP_PKEY_new();
    EVP_PKEY_assign_RSA(privkey, rsa);
    // Создание контекста EVP_MD_CTX
    EVP_MD_CTX* mdctx = EVP_MD_CTX_new();
    if (!mdctx) {
        cerr << "Ошибка создания EVP_MD_CTX." << endl;
        RSA_free(rsa);
        EVP_PKEY_free(privkey);
        return 1;
    }
    // Создание объекта EVP_PKEY_CTX
    EVP_PKEY_CTX* pctx;
    // Инициализация контекста хэширования

```

```

        if (EVP_DigestSignInit(mdctx, &pctx, EVP_sha256(), NULL,
privkey) != 1) {
            cerr << "Инициализация контекста хеширования не удалась."
<< endl;
            RSA_free(rsa);
            EVP_PKEY_free(privkey);
            return 1;
        }
        // Установка параметров шифрования RSA
        if (EVP_PKEY_CTX_set_rsa_padding(pctx, RSA_PKCS1_PADDING) !=
1) {
            cerr << "Ошибка установки параметров шифрования RSA." <<
endl;
            RSA_free(rsa);
            EVP_PKEY_free(privkey);
            return 1;
        }
        // Добавление данных для подписи
        if (EVP_DigestSignUpdate(mdctx, message.c_str(),
message.length()) != 1) {
            cerr << "Добавление данных для подписи не удалось." <<
endl;
            RSA_free(rsa);
            EVP_PKEY_free(privkey);
            return 1;
        }
        size_t siglen;
        // Получение подписи
        if (EVP_DigestSignFinal(mdctx, NULL, &siglen) != 1) {
            cerr << "Получение длины подписи не удалось." << endl;
            RSA_free(rsa);
            EVP_PKEY_free(privkey);
            return 1;
        }
        unsigned char* sign = new unsigned char[siglen];
        if (EVP_DigestSignFinal(mdctx, sign, &siglen) != 1) {
            cerr << "Получение подписи не удалось." << endl;
            RSA_free(rsa);
            EVP_PKEY_free(privkey);
            delete[] sign;
            return 1;
        }
        // Запись подписи в файл
        ofstream out("signature.sig");
        out.write((char*)sign, siglen);
        out.close();
        cout << "Электронная подпись сохранена в файл signature.sig\
nЭлектронная подпись:\n";
        int chet = static_cast<int>(siglen);
        for (int i = 0; i < chet; ++i) {
            printf("%02x", sign[i]);

```

```

    }
    cout << endl;

    // Освобождение ресурсов
    RSA_free(rsa);
    EVP_PKEY_free(privkey);
    delete[] sign;
    return 0;
}

int checkSignature()
{
    string message;
    string path;
    stringstream buffer;
    // Получение пути к файлу
    cout << "Введите путь к файлу с информацией: ";
    cin >> path;
    // path = "message.txt";

    // Чтение текста для подписи
    ifstream file(path, ios::binary);
    if (file.is_open()) {
        buffer.str("");
        buffer.clear();
        buffer << file.rdbuf();
        message = buffer.str();
        file.close();
    } else {
        cerr << "Не удалось открыть файл." << endl;
        return 1;
    }

    cout << "Введите путь к файлу с открытым ключом: ";
    cin >> path;
    //path = "public.pem";

    // Загрузка открытого ключа
    FILE *fp = fopen(path.c_str(), "r");
    if (!fp) {
        cerr << "Ошибка открытия файла с открытым ключом." <<
endl;
        return 1;
    }
    RSA *pubrsa = PEM_read_RSAPublicKey(fp, NULL, NULL, NULL);
    fclose(fp);
    EVP_PKEY *pubkey = EVP_PKEY_new();
    EVP_PKEY_assign_RSA(pubkey, pubrsa);

    cout << "Введите путь к файлу с подписью: ";
    cin >> path;

```

```

//path = "signature.sig";

// Чтение подписи из файла
string siglstring;
ifstream Signature(path, ios::binary);
if (Signature.is_open()) {
    stringstream buffer;
    buffer << Signature.rdbuf();
    siglstring = buffer.str();
    Signature.close();
} else {
    cerr << "Не удалось открыть файл." << endl;
    RSA_free(pubrsa);
    EVP_PKEY_free(pubkey);
    return -1;
}

EVP_MD_CTX *mdctx = EVP_MD_CTX_new();
EVP_PKEY_CTX *pctx;
pctx = EVP_PKEY_CTX_new(pubkey, NULL);

unsigned char* sig = new unsigned char[siglstring.size() + 1];
memcpy(sig, siglstring.c_str(), siglstring.size() + 1);
strcpy((char*)sig, siglstring.c_str());

// Проверка подписи
int verified = EVP_DigestVerifyInit(mdctx, &pctx,
EVP_sha256(), NULL, pubkey);

if (verified <= 0) {
    cerr << "Инициализация проверки подписи не удалась." <<
endl;
    RSA_free(pubrsa);
    EVP_PKEY_free(pubkey);
    EVP_MD_CTX_free(mdctx);
    delete[] sig;
    return -1;
}

if (EVP_DigestVerifyUpdate(mdctx, message.c_str(),
message.length()) <= 0) {
    cerr << "Добавление данных для проверки подписи не
удалось." << endl;
    RSA_free(pubrsa);
    EVP_PKEY_free(pubkey);
    EVP_MD_CTX_free(mdctx);
    delete[] sig;
    return -1;
}

verified = EVP_DigestVerifyFinal(mdctx, sig,
siglstring.size());

```

```

    if (verified == 1) {
        cout << "Подпись верифицирована!" << endl;
    } else {
        cout << "Проверка подписи не удалась." << endl;
    }

    // Освобождение ресурсов
    RSA_free(pubrsa);
    EVP_PKEY_free(pubkey);
    EVP_MD_CTX_free(mdctx);
    delete[] sig;

    return 0;
}

int main()
{
    int mode = 1;

    while (mode != 0) {
        cout << "Выберите режим работы:\n1 - вычисление
электронной подписи;\n2 - проверка электронной подписи;\n0 -
выход.\n";
        cin >> mode;

        switch (mode) {
            case 1: {
                calculateSignature();
                break;
            }
            case 2: {
                checkSignature();
                break;
            }
            default:
                break;
        }
    }
    return 0;
}

```

Код программы создания сертификата и проверки электронной подписи:

```

#include <openssl/pem.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstring>

using namespace std;

```

```

int generateSertificate()
{
    string path;
    cout << "Введите путь к файлу с открытым ключом: ";
    cin >> path;
    //path = "public.pem";

    // Чтение ключа RSA из файла
    FILE* fp = fopen(path.c_str(), "r");
    if(!fp) {
        cerr << "Ошибка открытия файла с открытым ключом" << endl;
        return 1;
    }
    RSA* rsa = PEM_read_RSAPublicKey(fp, NULL, NULL, NULL);
    if(!rsa) {
        cerr << "Ошибка чтения открытого ключа" << endl;
        fclose(fp);
        return 1;
    }
    fclose(fp);
    EVP_PKEY *pubkey = EVP_PKEY_new();
    EVP_PKEY_assign_RSA(pubkey, rsa);

    // Создание сертификата
    X509* x509 = X509_new();
    if(!x509) {
        cerr << "Ошибка создания сертификата" << endl;
        EVP_PKEY_free(pubkey);
        return 1;
    }
    X509_set_version(x509, 2);
    ASN1_INTEGER_set(X509_get_serialNumber(x509), 1);
    X509_gmtime_adj(X509_get_notBefore(x509), 0);
    X509_gmtime_adj(X509_get_notAfter(x509), (long) 60*60*24*365);
    X509_set_pubkey(x509, pubkey);

    // Установка имени владельца
    X509_NAME* name = X509_get_subject_name(x509);
    if(!name) {
        cerr << "Ошибка получения имени субъекта" << endl;
        X509_free(x509);
        EVP_PKEY_free(pubkey);
        return 1;
    }
    int res = X509_NAME_add_entry_by_txt(name, "CN", MBSTRING_ASC,
    (const unsigned char*)"LabWork Sertificate", -1, -1, 0);
    if(res != 1) {
        cerr << "Ошибка добавления имени субъекта" << endl;
        X509_free(x509);
        EVP_PKEY_free(pubkey);
        return 1;
    }
}

```

```

}

cout << "Введите путь к файлу с закрытым ключом: ";
cin >> path;
//path = "private.pem";

// Загрузка закрытого ключа
fp = fopen(path.c_str(), "r");
if(!fp) {
    cerr << "Ошибка открытия файла с закрытым ключом" << endl;
    X509_free(x509);
    EVP_PKEY_free(pubkey);
    return 1;
}
RSA *rsak = PEM_read_RSAPrivateKey(fp, NULL, NULL, NULL);
if(!rsak) {
    cerr << "Ошибка чтения закрытого ключа" << endl;
    X509_free(x509);
    EVP_PKEY_free(pubkey);
    fclose(fp);
    return 1;
}
fclose(fp);
EVP_PKEY *privkey = EVP_PKEY_new();
EVP_PKEY_assign_RSA(privkey, rsak);

// Подписание сертификата ключом RSA из файла
res = X509_sign(x509, privkey, EVP_sha256());
if(res == 0) {
    cerr << "Ошибка подписи сертификата" << endl;
    X509_free(x509);
    EVP_PKEY_free(pubkey);
    EVP_PKEY_free(privkey);
    return 1;
}

// Сохранение сертификата в файл certificate.crt
fp = fopen("certificate.crt", "w");
if(!fp) {
    cerr << "Ошибка открытия файла для записи сертификата" <<
endl;
    X509_free(x509);
    EVP_PKEY_free(pubkey);
    EVP_PKEY_free(privkey);
    return 1;
}
res = PEM_write_X509(fp, x509);
fclose(fp);
if(res != 1) {
    cerr << "Ошибка записи сертификата" << endl;
    X509_free(x509);
}

```

```

        EVP_PKEY_free(publkey);
        EVP_PKEY_free(privkey);
        return 1;
    }

    cout << "Сертификат успешно создан и сохранен в файл
certificate.crt\n";

    EVP_PKEY_free(publkey);
    EVP_PKEY_free(privkey);
    X509_free(x509);

    return 0;
}

int checkSignature()
{
    string path;
    string message;
    stringstream buffer;
    // Получение пути к файлу
    cout << "Введите путь к файлу с информацией: ";
    cin >> path;
    //path = "message.txt";

    // Чтение текста для подписи
    ifstream file(path, ios::binary);
    if (file.is_open()) {
        buffer.str("");
        buffer.clear();
        buffer << file.rdbuf();
        message = buffer.str();
        file.close();
    } else {
        cerr << "Не удалось открыть файл." << endl;
        return 1;
    }

    cout << "Введите путь к файлу с сертификатом: ";
    cin >> path;
    //path = "certificate.crt";

    // Чтение сертификата
    FILE* fp = fopen(path.c_str(), "r");
    if (!fp) {
        cerr << "Ошибка открытия файла с сертификатом." << endl;
        return 1;
    }
    X509* x509 = PEM_read_X509(fp, NULL, NULL, NULL);

    // Получение открытого ключа

```

```

EVP_PKEY* pubkey = X509_get_pubkey(x509);
fclose(fp);

cout << "Введите путь к файлу с электронной подписью: ";
cin >> path;
//path = "signature.sig";

// Чтение подписи из файла
string siglstring;
ifstream Signature(path, ios::binary);
if (Signature.is_open()) {
    stringstream buffer;
    buffer << Signature.rdbuf();
    siglstring = buffer.str();
    Signature.close();
} else {
    cerr << "Не удалось открыть файл." << endl;
    return -1;
}

EVP_MD_CTX *mdctx = EVP_MD_CTX_new();
EVP_PKEY_CTX *pctx;
pctx = EVP_PKEY_CTX_new(pubkey, NULL);

unsigned char* sig = new unsigned char[siglstring.size() + 1];
memcpy(sig, siglstring.c_str(), siglstring.size() + 1);
strcpy((char*)sig, siglstring.c_str());

// Проверка подписи
int verified = EVP_DigestVerifyInit(mdctx, &pctx,
EVP_sha256(), NULL, pubkey);

if (verified <= 0) {
    cerr << "Инициализация проверки подписи не удалась." <<
endl;
    return -1;
}

if (EVP_DigestVerifyUpdate(mdctx, message.c_str(),
message.length()) <= 0) {
    cerr << "Добавление данных для проверки подписи не
удалось." << endl;
    return -1;
}
verified = EVP_DigestVerifyFinal(mdctx, sig,
siglstring.size());

if (verified == 1) {
    cout << "Подпись верифицирована!" << endl;
} else {
    cout << "Проверка подписи не удалась." << endl;
}

```

```

    }

    // Освобождение ресурсов
    EVP_PKEY_free(pubkey);
    EVP_MD_CTX_free(mdctx);
    delete[] sig;

    return 0;
}

int main()
{
    int mode = 2;
    while (mode != 0) {
        cout << "Выберите режим работы:\n1 - создание
сертификата;\n2 - проверка электронной подписи;\n0 - выход.\n";
        cin >> mode;

        switch (mode) {
            case 1: {
                generateSertificate();
                break;
            }
            case 2: {
                checkSignature();
                break;
            }
            default:
                break;
        }
    }
    return 0;
}

```

Приложение Б
(обязательное)

Код приложения для проверки результатов выполнения практикума

AES.h:

```
#pragma once
#include <openssl/evp.h>
#include <string.h>
#include <iostream>
#include <fstream>

using namespace std;

// Функция зашифрования
int encrypt(unsigned char *aad, int aad_len, unsigned char *key,
            unsigned char *iv,
            unsigned char *ciphertext, unsigned char *tag)
{
    string path;
    // Получение пути к файлу
    cout << "Введите путь к файлу с открытым текстом: ";
    cin >> path;
    //path = "plaintextAES.txt";
    string kont;
    string text;
    ifstream file(path);
    file.seekg(0, ios::end);
    int size = file.tellg();
    file.seekg(0);
    unsigned char plaintext[size]= {};
    int i = 0;
    while(getline(file, text)) {
        if (i > 0)
            kont += '\n';
        i = 1;
        kont += text;
    }
    file.close();
    strcpy((char*)plaintext, kont.c_str());
    int plaintext_len = strlen((char *) plaintext);

    EVP_CIPHER_CTX *ctx;
    int len;
    int ciphertext_len;

    // Создание и инициализация контекста
    if (!(ctx = EVP_CIPHER_CTX_new())) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка алгоритма шифрования
    if (1 != EVP_EncryptInit_ex(ctx, EVP_aes_256_gcm(), NULL,
    NULL, NULL)) {
```

```

        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка ключа и IV
    if (1 != EVP_EncryptInit_ex(ctx, NULL, NULL, key, iv)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка AAD (дополнительные данные аутентификации)
    if (aad && aad_len > 0) {
        if (1 != EVP_EncryptUpdate(ctx, NULL, &len, aad, aad_len))
        {
            EVP_CIPHER_CTX_free(ctx);
            return -1;
        }
    }

    // Зашифрование данных
    if (1 != EVP_EncryptUpdate(ctx, ciphertext, &len, plaintext,
plaintext_len)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }
    ciphertext_len = len;

    // Завершение шифрования и получение тега
    if (1 != EVP_EncryptFinal_ex(ctx, ciphertext + len, &len)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }
    ciphertext_len += len;

    if (1 != EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_GET_TAG, 16,
tag)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Освобождение контекста
    EVP_CIPHER_CTX_free(ctx);

    return ciphertext_len;
}

// Функция расшифрования
int decrypt(unsigned char *ciphertext, int ciphertext_len,
            unsigned char *aad, int aad_len,
            unsigned char *tag,
            unsigned char *key,

```

```

        unsigned char *iv,
        unsigned char *plaintext)
{
    EVP_CIPHER_CTX *ctx;
    int len;
    int plaintext_len;
    int ret;

    // Создание и инициализация контекста
    if (!(ctx = EVP_CIPHER_CTX_new())) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка алгоритма шифрования
    if (!EVP_DecryptInit_ex(ctx, EVP_aes_256_gcm(), NULL, NULL,
NULL)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка ключа и IV
    if (!EVP_DecryptInit_ex(ctx, NULL, NULL, key, iv)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Установка AAD (дополнительные данные аутентификации)
    if (aad && aad_len > 0) {
        if (!EVP_DecryptUpdate(ctx, NULL, &len, aad, aad_len)) {
            EVP_CIPHER_CTX_free(ctx);
            return -1;
        }
    }

    // Расшифрование данных
    if (!EVP_DecryptUpdate(ctx, plaintext, &len, ciphertext,
ciphertext_len)) {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }
    plaintext_len = len;

    // Установка тега
    if (!EVP_CIPHER_CTX_ctrl(ctx, EVP_CTRL_GCM_SET_TAG, 16, tag))
    {
        EVP_CIPHER_CTX_free(ctx);
        return -1;
    }

    // Завершение расшифрования

```

```

ret = EVP_DecryptFinal_ex(ctx, plaintext + len, &len);

// Освобождение контекста
EVP_CIPHER_CTX_free(ctx);

if (ret > 0) {
    plaintext_len += len;
    plaintext[plaintext_len] = '\0';
    return plaintext_len;
} else {
    return -1;
}
}

```

hash.h:

```

#pragma once
#include <iostream>
#include <fstream>
#include <cstring>
#include <openssl/sha.h>

using namespace std;

// Вычисление хэш-кода
int calcHash (unsigned char *hash)
{
    string text, kont;
    string path;

    // Получение пути к файлу
    cout << "Введите путь к файлу, для которого необходимо  

    вычислить хэш-код: ";
    cin >> path;
    //path = "textHASH.txt";

    // Открытие и чтение информации из файла
    ifstream file(path, ios::binary);
    int i = 0;
    while(getline(file, text)) {
        if (i > 0)
            kont += '\n';
        i = 1;
        kont += text;
    }
    file.close();

    // Вычисление хэш-кода
    SHA256_CTX sha256;
    SHA256_Init(&sha256);
    SHA256_Update(&sha256, kont.c_str(), kont.size());
    SHA256_Final(hash, &sha256);
}

```

```

        return 0;
    }

    // Проверка хэш-кода
    int checkHash()
    {
        unsigned char hash[SHA256_DIGEST_LENGTH];
        string path;
        // Получение пути к файлу
        cout << "Введите путь к файлу с хэш-кодом: ";
        cin >> path;
        //path = "hash.txt";

        // Чтение хэш-кода из файла
        ifstream hashFile2(path, ios::binary);
        unsigned char hash2[SHA256_DIGEST_LENGTH];
        hashFile2.read((char*)hash2, SHA256_DIGEST_LENGTH);

        calcHash(hash);

        // Проверка хэш-кодов
        if (memcmp(hash, hash2, SHA256_DIGEST_LENGTH) == 0) {
            cout << "Хэш-код совпадает!" << endl;
        } else {
            cout << "Хэш-код не совпадает!" << endl;
        }
        return 0;
    }
}

```

RSA.h:

```

#pragma once
#include <iostream>
#include <fstream>
#include <openssl/rsa.h>
#include <openssl/pem.h>

using namespace std;

int decryptRSA()
{
    string path;
    // Получение пути к файлу
    cout << "Введите путь к файлу с шифротекстом: ";
    cin >> path;
    //path = "encryptRSA.txt";

    // Чтение шифротекста из файла
    ifstream file(path, ios::binary);
    file.seekg(0, file.end);
    int length = file.tellg();
    file.seekg(0, file.beg);
}

```

```

unsigned char *encrypted = new unsigned char[length];
file.read((char *)encrypted, length);
file.close();

// Получение пути к файлу
cout << "Введите путь к файлу с закрытым ключом: ";
cin >> path;
//path = "private.pem";

//Загрузка ключей
FILE* fp = fopen(path.c_str(), "r");
if (!fp) {
    cerr << "Ошибка открытия файла с закрытым ключом." <<
endl;
    delete[] encrypted;
    return 1;
}
RSA *rsapriv = PEM_read_RSAPrivateKey(fp, NULL, NULL, NULL);
fclose(fp);

// Расшифрование сообщения
unsigned char *decrypted = new unsigned char[length];
int decrypted_length = RSA_private_decrypt(length, encrypted,
decrypted, rsapriv, RSA_PKCS1_OAEP_PADDING);
if (decrypted_length == -1) {
    cout << "Ошибка расшифрования сообщения" << endl;
    delete[] encrypted;
    delete[] decrypted;
    return 1;
}

// Получение пути к файлу
cout << "Введите путь к файлу с открытым текстом: ";
cin >> path;
//path = "plaintextRSA.txt";
string kont;
string text;
ifstream file1(path);
file1.seekg(0, ios::end);
int size = file1.tellg();
file1.seekg(0);
unsigned char plaintext[size]= {};
int i = 0;
while(getline(file1,text)) {
    if (i > 0)
        kont += '\n';
    i = 1;
    kont += text;
}
file1.close();
strcpy((char*)plaintext, kont.c_str());

```

```

int plaintext_len = strlen((char *) plaintext);

if (memcmp(plaintext, decrypted, plaintext_len) == 0) {
    cout << "Шифротекст верен\n";
} else {
    cout << "Шифротекст не верен\n";
}

// Очистка памяти
delete[] encrypted;
delete[] decrypted;
return 0;
}

```

signature.h:

```

#pragma once
#include <openssl/pem.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstring>

using namespace std;

int checkSignature()
{
    string message;
    string path;
    stringstream buffer;
    // Получение пути к файлу
    cout << "Введите путь к файлу с информацией: ";
    cin >> path;
    //path = "messageSIGN.txt";

    // Чтение текста для подписи
    ifstream file(path, ios::binary);
    if (file.is_open()) {
        buffer.str("");
        buffer.clear();
        buffer << file.rdbuf();
        message = buffer.str();
        file.close();
    } else {
        cerr << "Не удалось открыть файл." << endl;
        return 1;
    }

    cout << "Введите путь к файлу с открытым ключом: ";
    cin >> path;
    //path = "public.pem";
}

```

```

// Загрузка открытого ключа
FILE *fp = fopen(path.c_str(), "r");
if (!fp) {
    cerr << "Ошибка открытия файла с открытым ключом." <<
endl;
    return 1;
}
RSA *pubrsa = PEM_read_RSAPublicKey(fp, NULL, NULL, NULL);
fclose(fp);
EVP_PKEY *pubkey = EVP_PKEY_new();
EVP_PKEY_assign_RSA(pubkey, pubrsa);

cout << "Введите путь к файлу с электронной подписью: ";
cin >> path;
//path = "signature.sig";

// Чтение подписи из файла
string siglstring;
ifstream Signature(path, ios::binary);
if (Signature.is_open()) {
    stringstream buffer;
    buffer << Signature.rdbuf();
    siglstring = buffer.str();
    Signature.close();
} else {
    cerr << "Не удалось открыть файл." << endl;
    RSA_free(pubrsa);
    EVP_PKEY_free(pubkey);
    return -1;
}

EVP_MD_CTX *mdctx = EVP_MD_CTX_new();
EVP_PKEY_CTX *pctx;
pctx = EVP_PKEY_CTX_new(pubkey, NULL);

unsigned char* sig = new unsigned char[siglstring.size() + 1];
memcpy(sig, siglstring.c_str(), siglstring.size() + 1);
strcpy((char*)sig, siglstring.c_str());

// Проверка подписи
int verified = EVP_DigestVerifyInit(mdctx, &pctx,
EVP_sha256(), NULL, pubkey);

if (verified <= 0) {
    cerr << "Инициализация проверки подписи не удалась." <<
endl;
    RSA_free(pubrsa);
    EVP_PKEY_free(pubkey);
    EVP_MD_CTX_free(mdctx);
    delete[] sig;
    return -1;
}

```

```

    }

    if (EVP_DigestVerifyUpdate(mdctx, message.c_str(),
message.length()) <= 0) {
        cerr << "Добавление данных для проверки подписи не
удалось." << endl;
        RSA_free(pubrsa);
        EVP_PKEY_free(pubkey);
        EVP_MD_CTX_free(mdctx);
        delete[] sig;
        return -1;
    }
    verified = EVP_DigestVerifyFinal(mdctx, sig,
siglstring.size());

    if (verified == 1) {
        cout << "Подпись верифицирована!" << endl;
    } else {
        cout << "Проверка подписи не удалась." << endl;
    }

    // Освобождение ресурсов
    RSA_free(pubrsa);
    EVP_PKEY_free(pubkey);
    EVP_MD_CTX_free(mdctx);
    delete[] sig;

    return 0;
}

```

certificate.h:

```

#pragma once
#include <openssl/pem.h>
#include <iostream>
#include <fstream>
#include <sstream>
#include <cstring>

using namespace std;

int checkSignatureCertificate()
{
    string path;
    string message;
    stringstream buffer;
    // Получение пути к файлу
    cout << "Введите путь к файлу с информацией: ";
    cin >> path;
    //path = "messageSIGN.txt";

    // Чтение текста для подписи

```

```

ifstream file(path, ios::binary);
if (file.is_open()) {
    buffer.str("");
    buffer.clear();
    buffer << file.rdbuf();
    message = buffer.str();
    file.close();
} else {
    cerr << "Не удалось открыть файл." << endl;
    return 1;
}

cout << "Введите путь к файлу с сертификатом: ";
cin >> path;
//path = "certificate.crt";

// Чтение сертификата
FILE* fp = fopen(path.c_str(), "r");
if (!fp) {
    cerr << "Ошибка открытия файла с сертификатом." << endl;
    return 1;
}
X509* x509 = PEM_read_X509(fp, NULL, NULL, NULL);

// Получение открытого ключа
EVP_PKEY* pubkey = X509_get_pubkey(x509);
fclose(fp);

cout << "Введите путь к файлу с электронной подписью: ";
cin >> path;
//path = "signature.sig";

// Чтение подписи из файла
string siglstring;
ifstream Signature(path, ios::binary);
if (Signature.is_open()) {
    stringstream buffer;
    buffer << Signature.rdbuf();
    siglstring = buffer.str();
    Signature.close();
} else {
    cerr << "Не удалось открыть файл." << endl;
    return -1;
}

EVP_MD_CTX *mdctx = EVP_MD_CTX_new();
EVP_PKEY_CTX *pctx;
pctx = EVP_PKEY_CTX_new(pubkey, NULL);

unsigned char* sig = new unsigned char[siglstring.size() + 1];
memcpy(sig, siglstring.c_str(), siglstring.size() + 1);

```

```

    strcpy((char*)sig, siglstring.c_str());

    // Проверка подписи
    int verified = EVP_DigestVerifyInit(mdctx, &pctx,
    EVP_sha256(), NULL, pubkey);

    if (verified <= 0) {
        cerr << "Инициализация проверки подписи не удалась." <<
endl;
        return -1;
    }

    if (EVP_DigestVerifyUpdate(mdctx, message.c_str(),
message.length()) <= 0) {
        cerr << "Добавление данных для проверки подписи не
удалось." << endl;
        return -1;
    }

    verified = EVP_DigestVerifyFinal(mdctx, sig,
siglstring.size());

    if (verified == 1) {
        cout << "Подпись верифицирована!" << endl;
    } else {
        cout << "Проверка подписи не удалась." << endl;
    }

    // Освобождение ресурсов
    EVP_PKEY_free(pubkey);
    EVP_MD_CTX_free(mdctx);
    delete[] sig;

    return 0;
}

```

main.cpp:

```

#include <AES.h>
#include <RSA.h>
#include <hash.h>
#include <signature.h>
#include <certificate.h>

using namespace std;

int main()
{
    int mode = 1;
    while (mode != 0) {
        cout << "Выберите режим работы:\n1 - AES;\n2 - HASH;\n3 -
RSA;\n4 - SIGNATURE;\n5 - CERTIFICATE;\n0 - выход.\n";
    }
}

```

```

cin >> mode;

switch (mode) {
case 1: {
    unsigned char ciphertext[1280];
    unsigned char decryptedtext[1280];
    unsigned char tag[16];
    //unsigned char key[] =
    "HDtTfHff4Z3VFPTYpgHcYmaBblQSwtaj";
    //unsigned char iv[] = "k95U7ctBlpRonE62";
    //unsigned char aad[] = "Reference_example_variant1";
    int decryptedtext_len, ciphertext_len;
    string path;

    unsigned char key[33];
    unsigned char iv[17];
    unsigned char* aad = new unsigned char[50];

    cout << "Введите ключ: ";
    cin >> key;
    cout << "Введите вектор инициализации: ";
    cin >> iv;
    cout << "Введите дополнительные данные аутентификации:
";

    cin >> aad;

    // Зашифрование данных
    ciphertext_len = encrypt(aad, strlen((char *) aad),
key, iv, ciphertext, tag);

    // Расшифрование данных
    decryptedtext_len = decrypt(ciphertext,
ciphertext_len, aad, strlen((char *) aad), tag, key, iv,
decryptedtext);
    // Добавление нулевого символа в конец расшифрованного
текста
    decryptedtext[decryptedtext_len] = '\0';

    // Получение пути к файлу
    cout << "Введите путь к файлу с расшифрованным текстом
от студента: ";
    cin >> path;
    //path = "decryptAES.txt";
    string kont;
    string text;
    ifstream file(path);
    file.seekg(0, ios::end);
    int size = file.tellg();
    file.seekg(0);
    unsigned char plaintext[size]= {};
    int i = 0;

```

```

        while(getline(file,text)) {
            if (i > 0)
                kont += '\n';
            i = 1;
            kont += text;
        }
        file.close();
        strcpy((char*)plaintext, kont.c_str());

        if (memcmp(plaintext, decryptedtext,
decryptedtext_len) == 0) {
            cout << "Тег аутентификации:\n";
            BIO_dump_fp(stdout, (const char *) tag,
sizeof(tag));
            cout << "Расшифрованный текст совпадает с открытым
текстом\n";
        } else {
            cout << "Расшифрованный текст отличается от
открытого текста\n";
        }

        delete[] aad;
        break;
    }
    case 2: {
        checkHash();
        break;
    }
    case 3: {
        decryptRSA();
        break;
    }
    case 4: {
        checkSignature();
        break;
    }
    case 5: {
        checkSignatureCertificate();
        break;
    }
    default:
        break;
    }
}
return 0;
}

```