МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ
УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
Кафедра «Интеллектуальные информационные технологии»

«К защите допускаю»
Заведующий кафедрой
_____ В.А. Головко
«____»_____ 2023 г.

ШИФРОВАНИЕ ДАННЫХ, ПЕРЕДАВАЕМЫХ ПО КОМПЬЮТЕРНЫМ СЕТЯМ, НА ПРИМЕРЕ ДАННЫХ ПРИЛОЖЕНИЯ JUNIORGRAM

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА К ДИПЛОМНОМУ ПРОЕКТУ

ДП.АС55.190232-05 81 00

Листов 61

Руководитель Ю. И. Давидюк

Выполнил В. И. Лепешев

Консультанты:

по ЕСПД Ю. И. Давидюк

по экономическому разделу А. Г. Проровский

Рецензент Л. В. Лизун

2023

Аннотация

61 с./65 с., 32 рис., 4 табл., 13 исп. ист., 5 граф. матер.

Цель данной дипломной работы состоит в обеспечении шифрованием приложения Juniorgram. В данном мессенджере отсутствует серьезная криптографическая защита, поэтому обеспечение шифрования потребует реализации дополнительных элементов защиты.

Для обеспечения шифрования были спроектированы, реализованы и внедрены в приложение модули симметричного и асимметричного шифрования, генерации и подтверждения ключа, верификации соединения. Также была произведена модификация существующего в Juniorgram модуля авторизации для использования разработанной защиты, были созданы генератор псевдослучайных чисел и хранилище для ключей симметричного шифрования.

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ
«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Факультет ФЭИС Кафедра ИИТ

УТВЕРЖДАЮ
Зав. кафедрой _____
(подпись)
«21» 04 2023 г.

ЗАДАНИЕ

по дипломному проектированию

Студенту Лепешеву Вячеславу Ильичу

1. Тема проекта Шифрование данных, передаваемых по компьютерным сетям, на примере данных приложения Juniorgram

(Утверждена приказом по вузу от 20.03.2023 № 266-С

2. Сроки сдачи студентом законченного проекта 20.06.2023

3. Исходные данные к проекту Разработать модуль шифрования для клиент-серверного приложения Juniorgram, а также модули, необходимые для функционирования шифрования в приложении. Рассмотреть подходящие алгоритмы шифрования, реализовать наиболее эффективный и менее ресурсоемкий.
Приложение должно быть кроссплатформенным и использовать следующие технологии:
ЯВУ C++, CMake, система контроля версий Git с использованием платформы GitHub, обязательным требованием является использование библиотеки CryptoPP

4. Содержание расчетно-пояснительной записки (перечень подлежащих разработке вопросов)

Введение
1. Системный анализ и постановка задачи
2. Проектирование системы
3. Реализация и испытание
4. Технико-экономическое обоснование
Заключение
Список сокращений
Список использованных источников
Приложение А. Текст программы

5. Перечень графического материала (с точным указанием обязательных чертежей и графиков)

1. Постановка задачи (плакат А1)
2. Выбор алгоритма шифрования(плакат А1)
3. Результаты работы системы (плакат А1)
4. Схема взаимодействия программ (чертеж А1)
5. Схема программы (чертеж А1)

6. Консультанты по проекту (с указанием относящихся к ним разделов проекта)
Экономический раздел — Проровский А. Г.

7. Дата выдачи проекта 20.04.23

8. Календарный график работы над проектом на весь период проектирования (с указанием сроков выполнения и трудоемкость отдельных этапов)

Раздел 1: 21.04.2023 — 29.04.2023 20%
Раздел 2: 30.04.2023 — 10.05.2023 20%
Раздел 3: 19.05.2023 — 26.05.2023 20%
Раздел 4: 27.05.2023 — 01.06.2023 20%

Оформление проекта: 01.06.2023 —10.06.2023 20%

Руководитель _____ Ю. И. Давидюк
(подпись)

Задание принял к исполнению (дата) 21.04.2023

(подпись студента) _____ В. И. Лепешев

Содержание

Введение

Интернет – это одно из самых значимых изобретений в истории человечества. Инструмент, некогда используемый лишь некоторыми людьми для передачи информации, сегодня доступен повсеместно и каждому и оказал огромное влияние на жизни людей.

Появление сети, доступной для каждого человека, сопровождалось ее сильным влиянием на различные сферы жизни: образование, здравоохранение, наука, культура, коммуникация, бизнес и т.д. Возможность выйти в Интернет позволила людям иметь доступ к огромному количеству информации и знаний, которые ранее были недоступны.

С каждым годом Интернет становится все более мобильным, распространенным удобным в использовании и интегрированным в жизни людей. Развитие Всемирной паутины и стремительный рост пользователей Сети дали начало множеству продуктов, предназначенных для повседневного использования: электронная почта, поисковые сети, онлайн-магазины, облачные сервисы, социальные сети и мессенджеры, с помощью которых люди стали использовать Интернет для обучения, общения и работы. На способе общения людей друг с другом оказало большое влияние появление приложений для коммуникации, что позволило в кратчайшие сроки донести информацию текстовой, визуальной или любой другой формы до получателей.

Однако, несмотря на все преимущества и удобства, которые дает Всемирная паутина, она также имеет свои недостатки и опасности. Одна из них – это угроза нарушения конфиденциальности и безопасности данных, которые передают через Интернет. Любая информация, передаваемая посредством Глобальной сети, может быть перехвачена, прочитана, изменена или украдена злоумышленниками. Любое негативное воздействие, оказываемое на данные, может привести к самым разным последствиям, вплоть до рассекречивания и публикации личных данных, кражи денежных средств или распространения дезинформации.

Уже на начальных этапах развития сети Интернет стало ясно, что необходимо обеспечить защиту передаваемых и хранимых в Сети данных от лиц, которые не должны иметь к ним доступ. Так появилась криптография – наука о защите информации от изменения или несанкционированного доступа.

Для сокрытия данных от третьих лиц в криптографии используется процесс шифрования. Шифрование – процесс преобразования данных таким образом, что их реальный смысл и содержание скрыто от посторонних. В совокупности с другими криптографическими приемами обеспечивается защита передаваемых данных.

В данном дипломном проекте реализуется криптографическая защита, включающая в себя симметричный и асимметричный алгоритмы шифрования, протоколы генерации и подтверждения ключа шифрования, в рамка мессенджера Juniorgram.

1 Системный анализ и постановка задачи

1.1 Анализ приложения Juniorgram

Приложение Juniorgram – это мессенджер, разрабатываемый для группового общения пользователей друг с другом.

Мессенджер работает по принципу двух уровней, где клиент и сервер – это независимые элементы. Информация передается между сторонами по определенным правилам, где она превращается в битовую последовательность перед отправкой и восстанавливается в формат, удобный для чтения человеком, после получения. Для передачи данных между элементами клиент-серверной архитектуры используется протокол TCP/IP с побайтовой обработкой, где сначала оправляется количество байт для чтения, а затем читается указанный объем данных. Для взаимодействия сторон в приложении используются запросы, которые отправляет клиент, и ответы, которые отправляет сервер при получении клиентского запроса.

Приложение функционирует на операционных системах Windows и Linux.

1.2 Постановка задачи

Данное приложение не имеет серьезной криптографической защиты. В нем имеется аутентификация пользователя по логину и паролю, к которому была применена хэш-функция, поэтому для обеспечения конфиденциальности переписки между пользователями необходимо реализовать шифрование данных. Разработка модуля шифрования также подразумевает реализацию как конкретного алгоритма, так и компенсацию слабых сторон выбранного метода шифрования, а также включает разработку всех необходимых криптографических модулей, требуемых для функционирования выбранного метода шифрования.

1.3 Анализ популярных алгоритмов шифрования и защиты данных

В настоящее время существует множество алгоритмов, используемых для защиты данных от третьих лиц. Постепенно появляются все более быстрые и более эффективные

способы шифрования информации, однако и современные варианты защиты данных, как и их предшественники, имеют свои особенности реализации, достоинства и недостатки.

Алгоритмы защиты данных чаще всего классифицируют по количеству ключей (см. рисунок 1.1). Далее рассмотрены симметричные и ассиметричные алгоритмы.

Рисунок 1.1 – Классификация криптографических алгоритмов

Симметричные алгоритмы – алгоритмы шифрования, в котором для шифрования и расшифрования применяется один и тот же криптографический ключ. Симметричные алгоритмы подразделяются на блочные и поточные.

Блочные шифры обрабатывают информацию блоками определенной длины, которые применяют к блоку ключ в установленном порядке, как правило, несколькими циклами перемешивания и подстановки, называемыми раундами. Результатом повторения раундов является лавинный эффект – нарастающая потеря соответствия битов между блоками открытых и зашифрованных данных. Алгоритмы блочного шифрования:

1. AES (англ. Advanced Encryption Standard; также Rijndael) – алгоритм, созданный в 1998 году, имеющий огромное распространение в различных сферах деятельности, включая правительства, военную сферу и банки. Алгоритм одобрен Национальным институтом стандартов и технологий (англ. The National Institute of Standards and Technology; далее – NIST) в «FIPS 197, Advanced Encryption Standard» [1]. При использовании AES алгоритма обычно указывается режим работы и, при необходимости, схема заполнения.

Некоторые режимы используют вектор инициализации, который формируется по зависящим от режима правилам, такие как использование только один раз (одноразовый номер), непредсказуемость перед его публикацией и т.д.

Возможны следующие категории режимы работы алгоритма:

1.1. Режимы, обеспечивающие только конфиденциальность:

1.1.1. Режим ECB (англ. electronic codebook)

Режим представляет собой электронную кодовую книгу, указан NIST в FIPS 81 [4]. Процессы шифрования и дешифрования в режиме ECB изображены на рисунках 1.2 и 1.3.

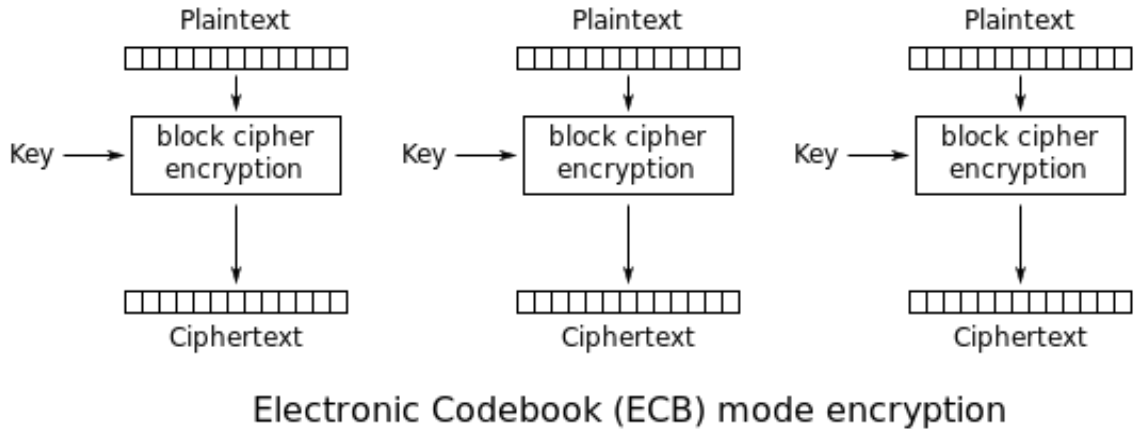


Рисунок 1.2 – Шифрование в режиме ECB

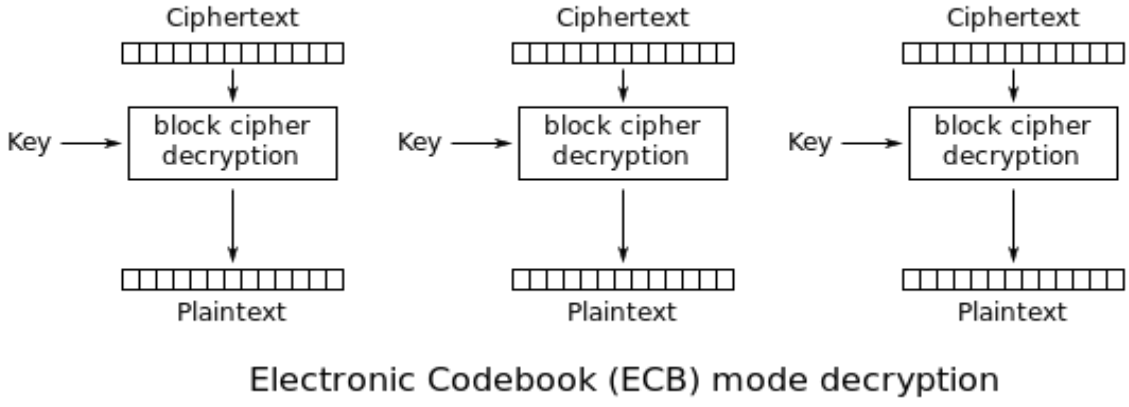


Рисунок 1.3 – Дешифрование в режиме ECB

Достоинства:

- простота;
- шифрование с возможностью распараллеливания;
- дешифрование с возможностью распараллеливания;

• доступен произвольный доступ для чтения.

Недостатки:

• отсутствие диффузии, т.е. происходит шифрование идентичных блоков исходной информации в идентичные блоки зашифрованной информации (см. рисунок 1.4);

• в протоколах без защиты целостности режим делает их более восприимчивым к атакам повторного воспроизведения, поскольку каждый блок расшифровывается точно таким же образом;

• сообщение должно быть дополнено до размера, кратного размеру блока шифрования;

• устарел и больше не используется.

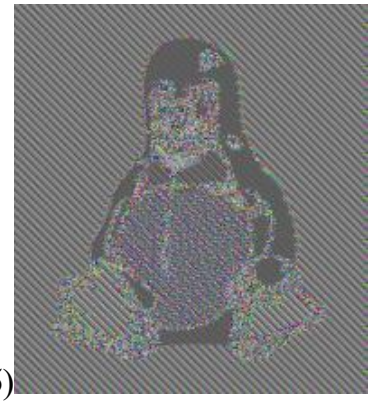
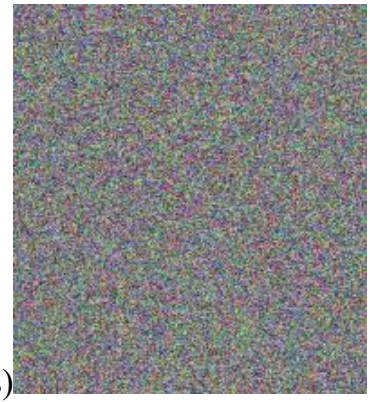
а)  б)  в) 

Рисунок 1.4 – а) оригинальное изображение; б) использование ECB режима; в) использование режимов, отличных от ECB

1.1.2. Режим CBC.

Режим представляет собой цепочку блоков шифрования (англ. cipher block chaining), где каждый блок открытого текста перед шифрованием связывается с предыдущим блоком зашифрованного текста, что обеспечивает зависимость блока зашифрованного текста от всех блоков открытого текста, обработанных до этого момента. Алгоритм указан NIST в FIPS 81 [4]. Процессы шифрования и дешифрования в режиме CBC изображены на рисунках 1.5 и 1.6.

Достоинства:

• дешифрование с возможностью распараллеливания;

• доступен произвольный доступ для чтения.

Недостатки:

• шифрование без возможности распараллеливания, т.к. все блоки обрабатываются последовательно;

• сообщение должно быть дополнено до размера, кратного размеру блока шифрования.

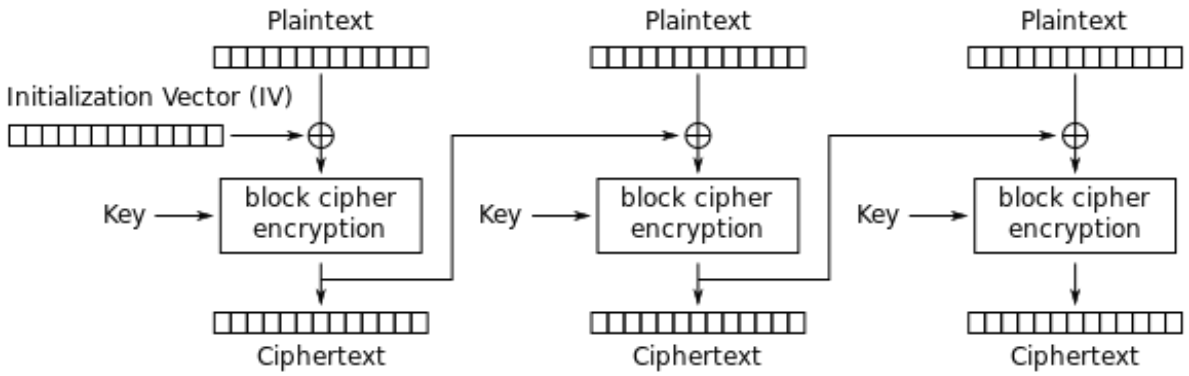


Рисунок 1.5 – Шифрование в режиме CBC

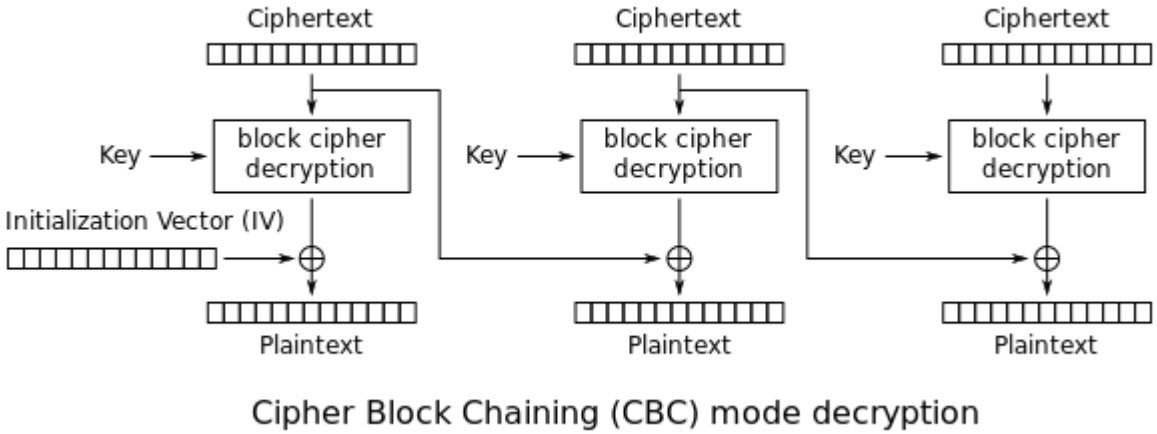


Рисунок 1.6 – Дешифрование в режиме CBC

1.1.3. Режим CFB (англ. cipher feedback).

Режим обратной связи по шифрованию в своей простейшей форме использует весь вывод блочного шифра. В этом варианте он очень похож на CBC, при этом блочный шифр превращается в самосинхронизирующийся потоковый шифр. Алгоритм указан NIST в FIPS 81 [4]. Процессы шифрования и дешифрования в режиме CFB изображены на рисунках 1.7 и 1.8.

Достоинства:

- дешифрование с возможностью распараллеливания;
- доступен произвольный доступ для чтения;
- сообщение не нужно дополнять до размера, кратного размеру блока шифрования.
- в определенных реализациях алгоритм самосинхронизируется и устойчив к потере зашифрованного текста.

Недостатки:

• шифрование без возможности распараллеливания.

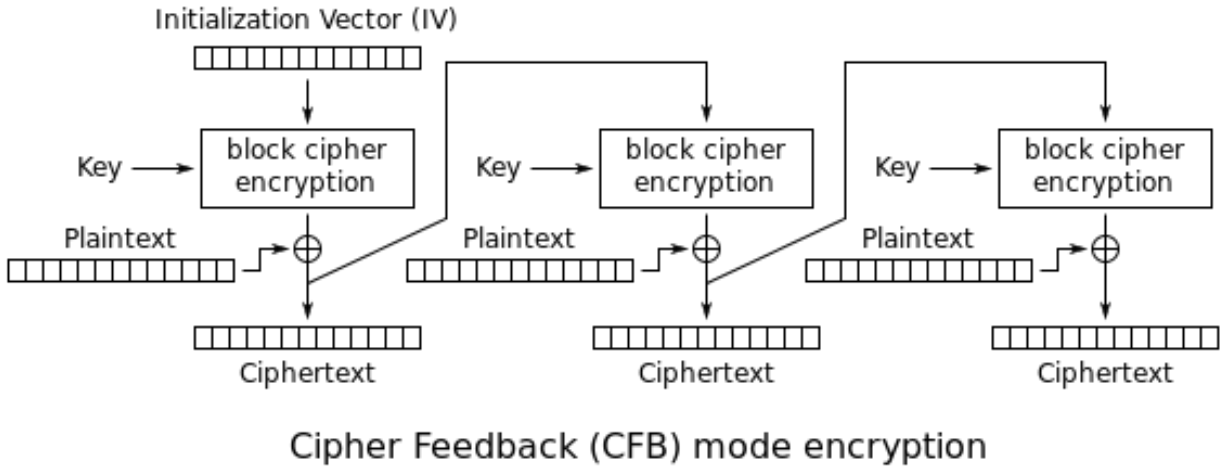


Рисунок 1.7 – Шифрование в режиме CFB

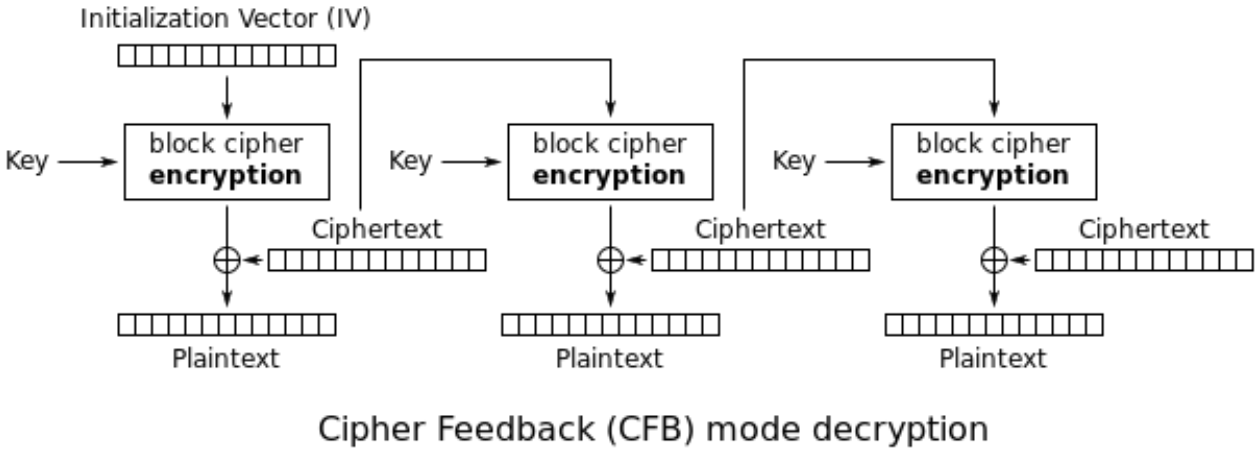


Рисунок 1.8 – Дешифрование в режиме CFB

1.1.4. Режим OFB (англ. output feedback).

Режим обратной связи на выходе преобразует блочный шифр в синхронный потоковый шифр. Он генерирует блоки ключевого потока, которые затем преобразуются в блоки открытого текста для получения зашифрованного текста. Алгоритм указан NIST в FIPS 81 [4]. Процессы шифрования и дешифрования в режиме OFB изображены на рисунках 1.9 и 1.10.

Достоинства:

• алгоритм помехоустойчив;

• сообщение не нужно дополнять до размера, кратного размеру блока шифрования.

Недостатки:

• шифрование без возможности распараллеливания;

• дешифрование без возможности распараллеливания;

• использование одного и того же вектора инициализации с одним и тем же ключом шифрования приводит к нарушению секретности;

• нет произвольного доступа для чтения.

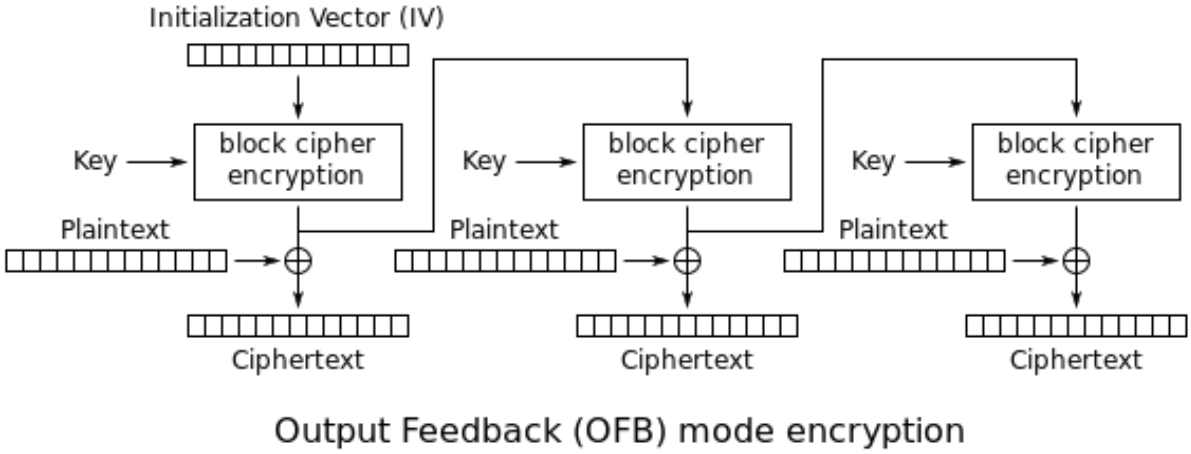


Рисунок 1.9 – Шифрование в режиме OFB

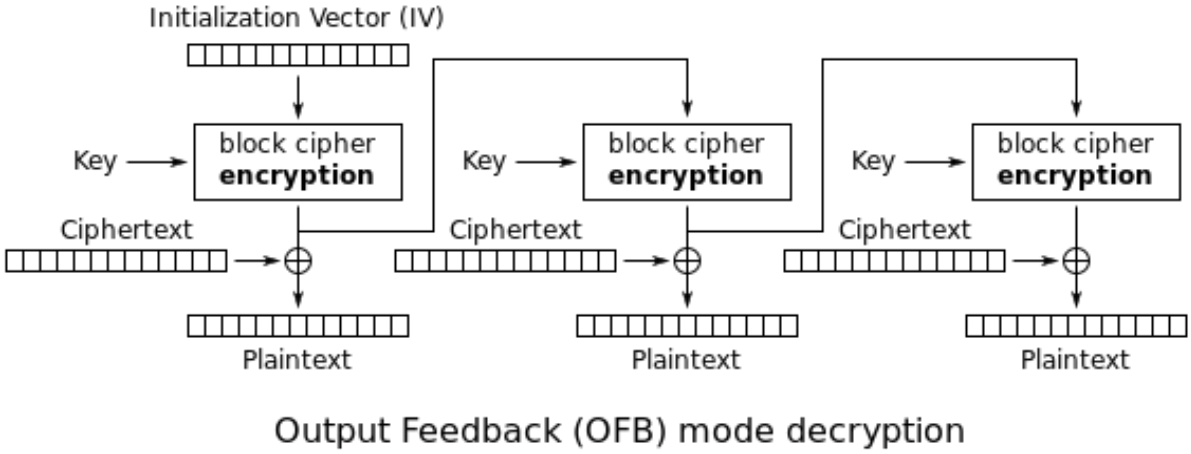


Рисунок 1.10 – Дешифрование в режиме OFB

1.1.5. Режим CTR.

Режим счетчика, также известный как режим целочисленного счетчика и режим сегментированного целочисленного, превращает блочный шифр в потоковый шифр. Алгоритм генерирует следующий блок ключевого потока путем шифрования последовательных значений счетчика. Режим CTR был стандартизирован в 2001 году NIST в SP 800-38A [5]. Процессы шифрования и дешифрования в режиме CTR изображены на рисунках 1.11 и 1.12.

Достоинства:

• шифрование с возможностью распараллеливания;

• дешифрование с возможностью распараллеливания;

• сообщение не нужно дополнять до размера, кратного размеру блока шифрования;

• доступен произвольный доступ для чтения

• широко распространен.

Недостатки:

• алгоритм не является помехоустойчивым, т.к. если информация о смещении или местоположении повреждена, то будет невозможно частично восстановить такие данные из-за зависимости от смещения в байтах;

• использование одного и того же вектора инициализации с одним и тем же ключом шифрования приводит к нарушению секретности.

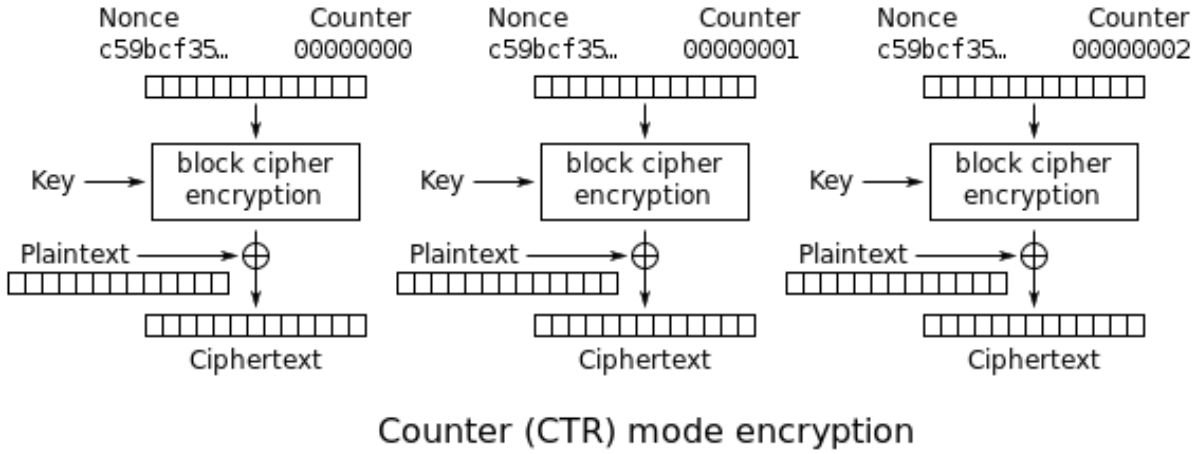


Рисунок 1.11 – Шифрование в режиме CTR

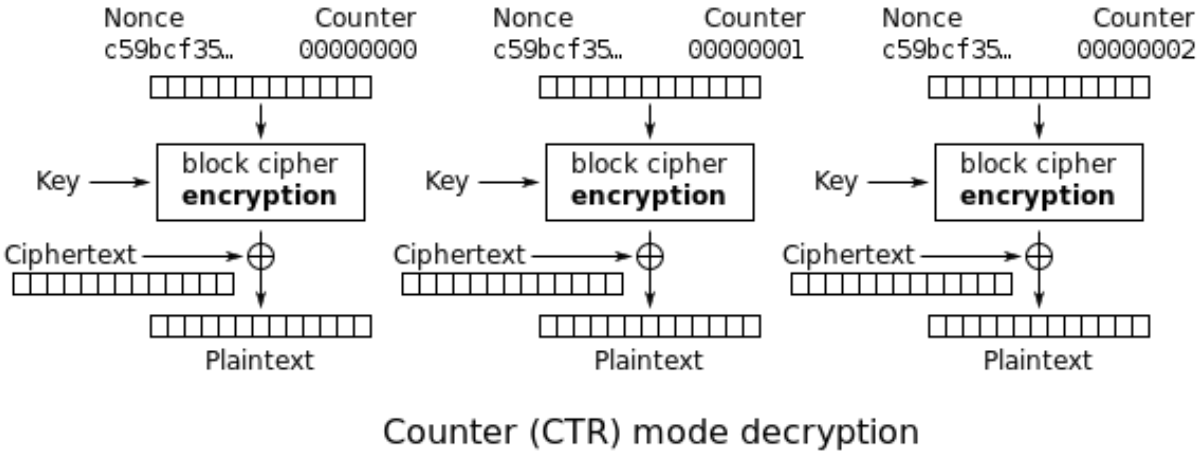


Рисунок 1.12 – Дешифрование в режиме CTR

1.2. Режимы аутентифицированного шифрования. Далее представлены режимы шифрования 4-го поколения: CCM, EAX, GCM и CWC. Данные режимы обеспечивают как конфиденциальность, целостность и аутентификацию данных:

1.2.1. Режим CCM (англ. counter with CBC-MAC, счетчик с кодом аутентификации сообщения цепочки блоков шифрования).

Данный алгоритм аутентифицированного шифрования, разработан для обеспечения как аутентификации, так и конфиденциальности. Режим CCM определен только для блочных шифров с длиной блока 128 бит. Алгоритм сочетает в себе режим счетчика (CTR) для обеспечения конфиденциальности с кодом аутентификации сообщения с цепочкой блоков шифрования для аутентификации. Эти два примитива применяются способом «аутентификация, затем шифрование».

Достоинства:

- перенимает преимущества CTR режима.

Недостатки:

- нужно заранее знать размер данных для шифрования;
- проверка целостности происходит после дешифровки.

1.2.2. Режим EAX (англ. encrypt-then-authenticate-then-translate).

Этот режим блочного шифра выполняет два прохода при шифровании: первый – для обеспечения конфиденциальности, второй – для аутентификации для каждого блока. Режим EAX был представлен 3 октября 2003 года вниманию NIST для замены CCM. Алгоритм работы изображен на рисунке 1.13.

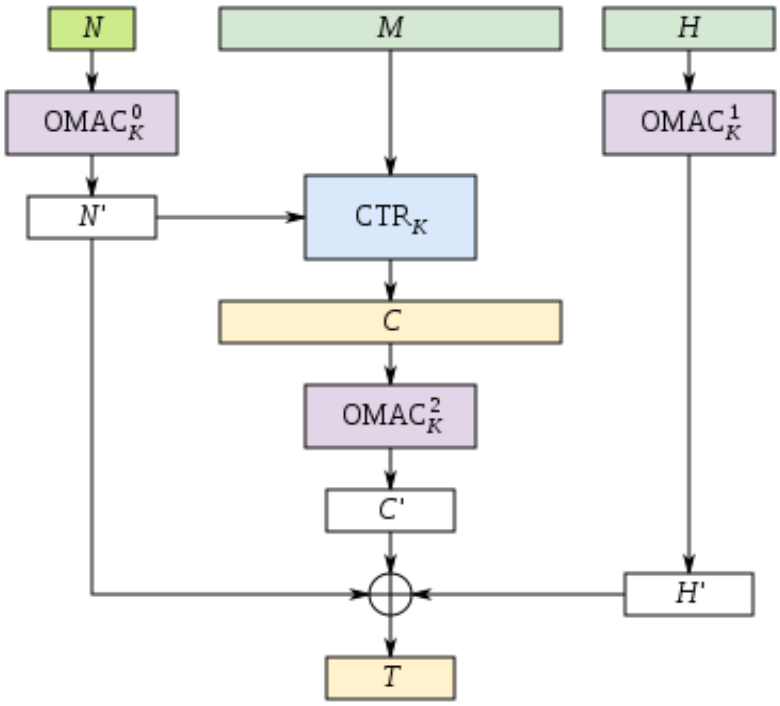


Рисунок 1.13 – Алгоритм работы режима EAX

Достоинства:

• доказуемо безопасный;

• расширение сообщения минимально после шифрования из-за ограничения на размер тега;

• алгоритм является «on-line», т.е. нет нужды знать заранее размер данных;

• не защищен патентом.

Недостатки:

• работает медленнее, чем хорошо разработанная однопроходная схема, основанная на тех же примитивах.

1.2.3. Режим GCM (англ. Galois/counter mode).

Режим комбинирует идею режима CTR и режим аутентификации Галуа. Ключевой особенностью является простота параллельного вычисления умножения поля Галуа, используемого для аутентификации. Эта функция обеспечивает более высокую пропускную способность, чем многие другие алгоритмы шифрования. GCM определен для блочных шифров с размером блока 128 бит. Алгоритм работы представлен ниже (см. рисунок 1.14).

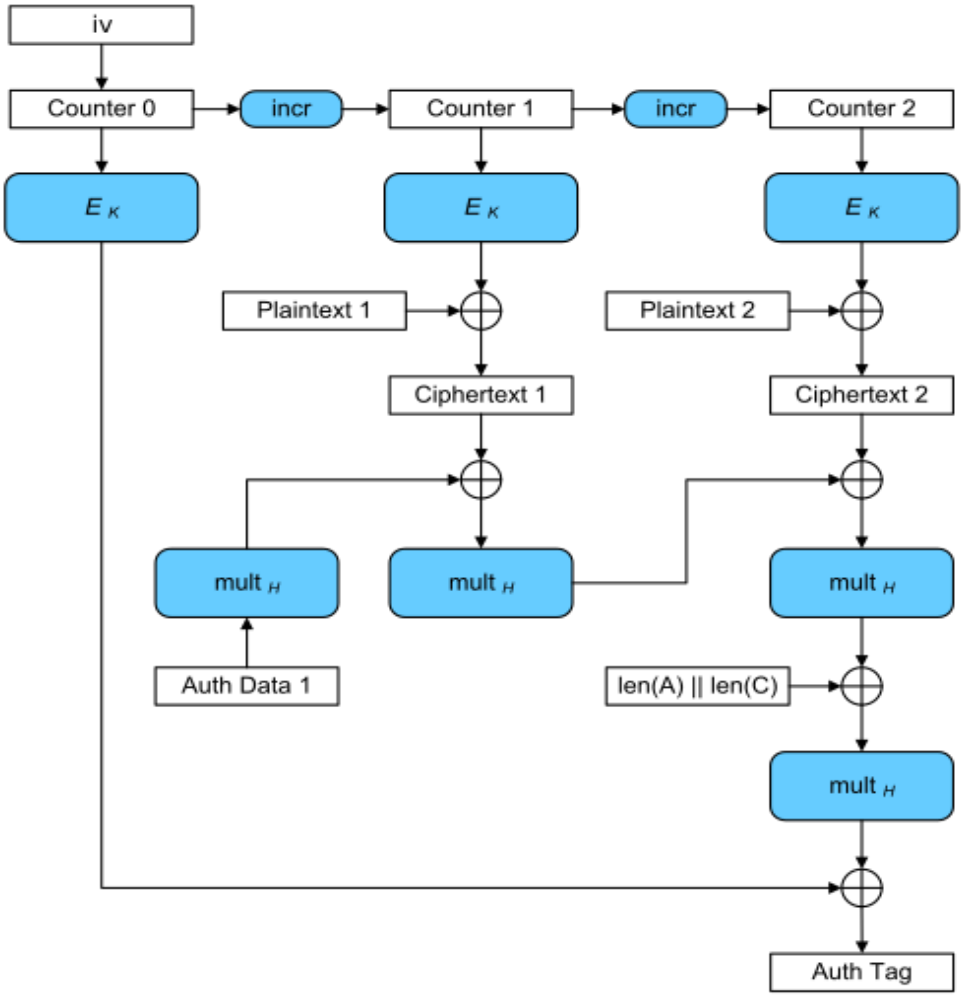


Рисунок 1.14 – Алгоритм работы GCM режима

Достоинства:

- может принимать векторы инициализации произвольной длины, что значительно упрощает требование об уникальности вектора;
- шифрование с возможностью распараллеливания;
- дешифрование с возможностью распараллеливания;
- произвольный доступ для чтения.

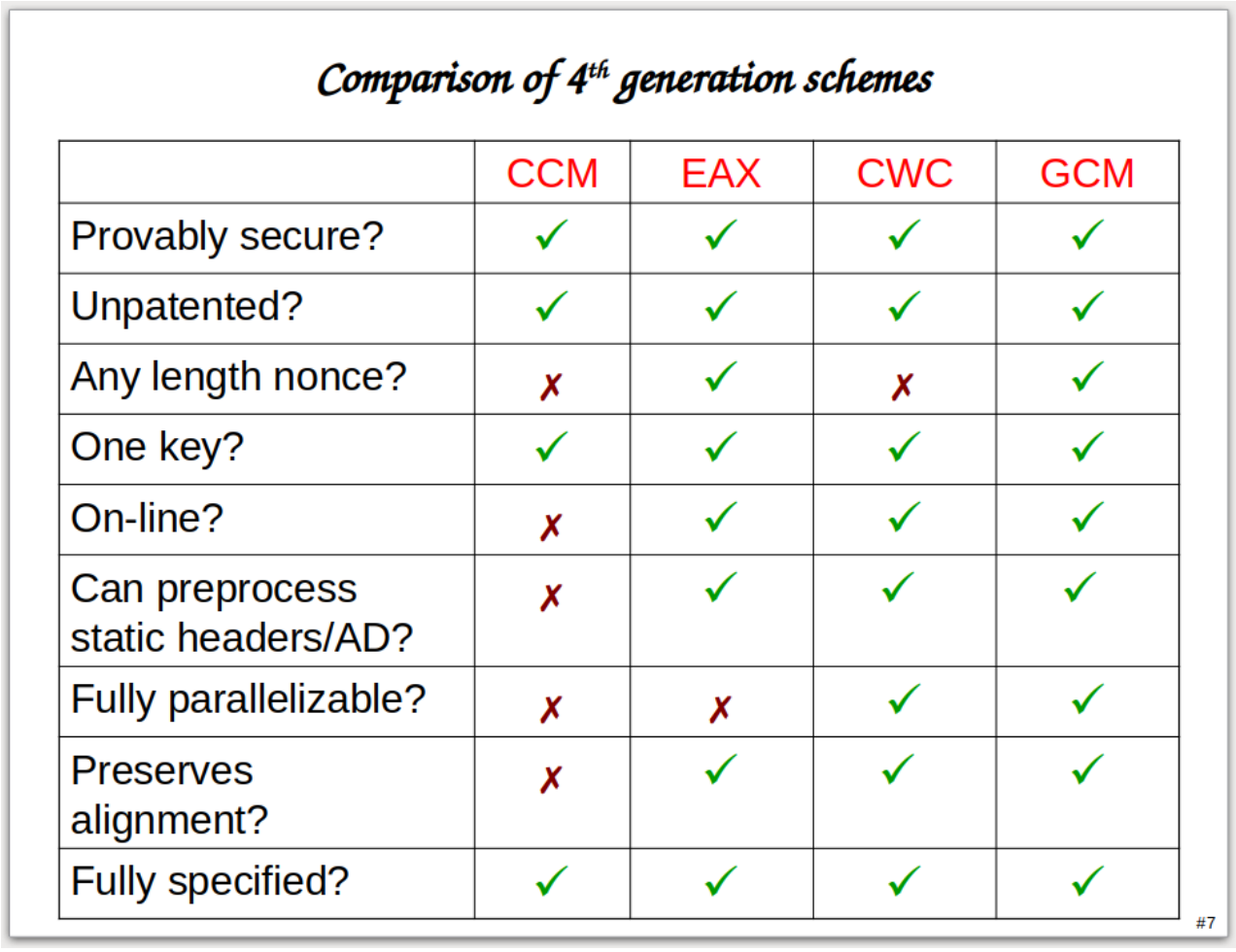
Недостатки:

- необходимо обеспечить уникальность вектора инициализации.

1.2.4. Режим CWC (Carter-Wegman в режиме CTR).

Режим работы блочного шифра, который обеспечивает как шифрование, так и встроенную целостность сообщений. Режим был передан в NIST для стандартизации, однако вместо этого был сделан выбор в пользу GCM.

В 2004 году доктор Дэвид Вагнер представил доклад в IACR [6], в котором обсуждались и сравнивались режимы аутентифицированного шифрования 4-го поколения (см. рисунок 1.15).



Comparison of 4th generation schemes

	CCM	EAX	CWC	GCM
Provably secure?	✓	✓	✓	✓
Unpatented?	✓	✓	✓	✓
Any length nonce?	✗	✓	✗	✓
One key?	✓	✓	✓	✓
On-line?	✗	✓	✓	✓
Can preprocess static headers/AD?	✗	✓	✓	✓
Fully parallelizable?	✗	✗	✓	✓
Preserves alignment?	✗	✓	✓	✓
Fully specified?	✓	✓	✓	✓

#7

Рисунок 1.15 – Сравнение режимов аутентифицированного шифрования

2. ГОСТ 28147-89 «Системы обработки информации. Защита криптографическая. Алгоритм криптографического преобразования» – государственный стандарт СССР и межгосударственный стандарт СНГ, описывающий алгоритм симметричного блочного шифрования и режимы его работы. Стандарт был отменен с 31 мая 2019 года в связи с переходом на ГОСТ 34.12-2018 (алгоритмы «Кузнечик» и «Магма»).

Достоинства:

- бесперспективность атаки полным перебором;
- эффективность реализации и, соответственно, высокое быстродействие на современных компьютерах;
- наличие защиты от навязывания ложных данных.

Недостатки:

- неполнота стандарта в части генерации ключей и таблиц замен;
- нельзя определить криптостойкость алгоритма, не зная заранее таблицы замен;
- реализации алгоритма от различных производителей могут быть несовместимы;
- возможность преднамеренного предоставления слабых таблиц;
- потенциальная возможность понизить криптостойкость шифр, используя «слабые» таблицы замены;
- заменен на более новые алгоритмы.

3. Шифр «Кузнечик» – симметричный алгоритм блочного шифрования, опубликованный в 2015 году. Использует блок размером 128 бит и ключа длиной 256 бит. Данный шифр утверждён (наряду с блочным шифром «Магма») в качестве стандарта в ГОСТ Р 34.12-2015 «Информационная технология. Криптографическая защита информации. Блочные шифры».

Достоинства:

- ожидается, что шифр будет устойчив ко всем видам атак на блочные шифры.

Недостатки:

- необходимость программирования шифра «с нуля».

4. Twofish – симметричный алгоритм блочного шифрования с размером блока 128 бит и длиной ключа до 256 бит. Являлся одним из пяти финалистов второго этапа конкурса AES. Алгоритм работы изображен на рисунке 1.16.

Достоинства:

- простота алгоритма;
- отсутствие слабых ключей;
- эффективная программная и аппаратная реализация;
- гибкость;
- алгоритм выстоял атаку по принципу «разделяй и властвуй».

Недостатки:

• сложность структуры алгоритма и, соответственно, анализа на предмет слабых ключей или скрытых связей;

• достаточно медленный по сравнению с AES.

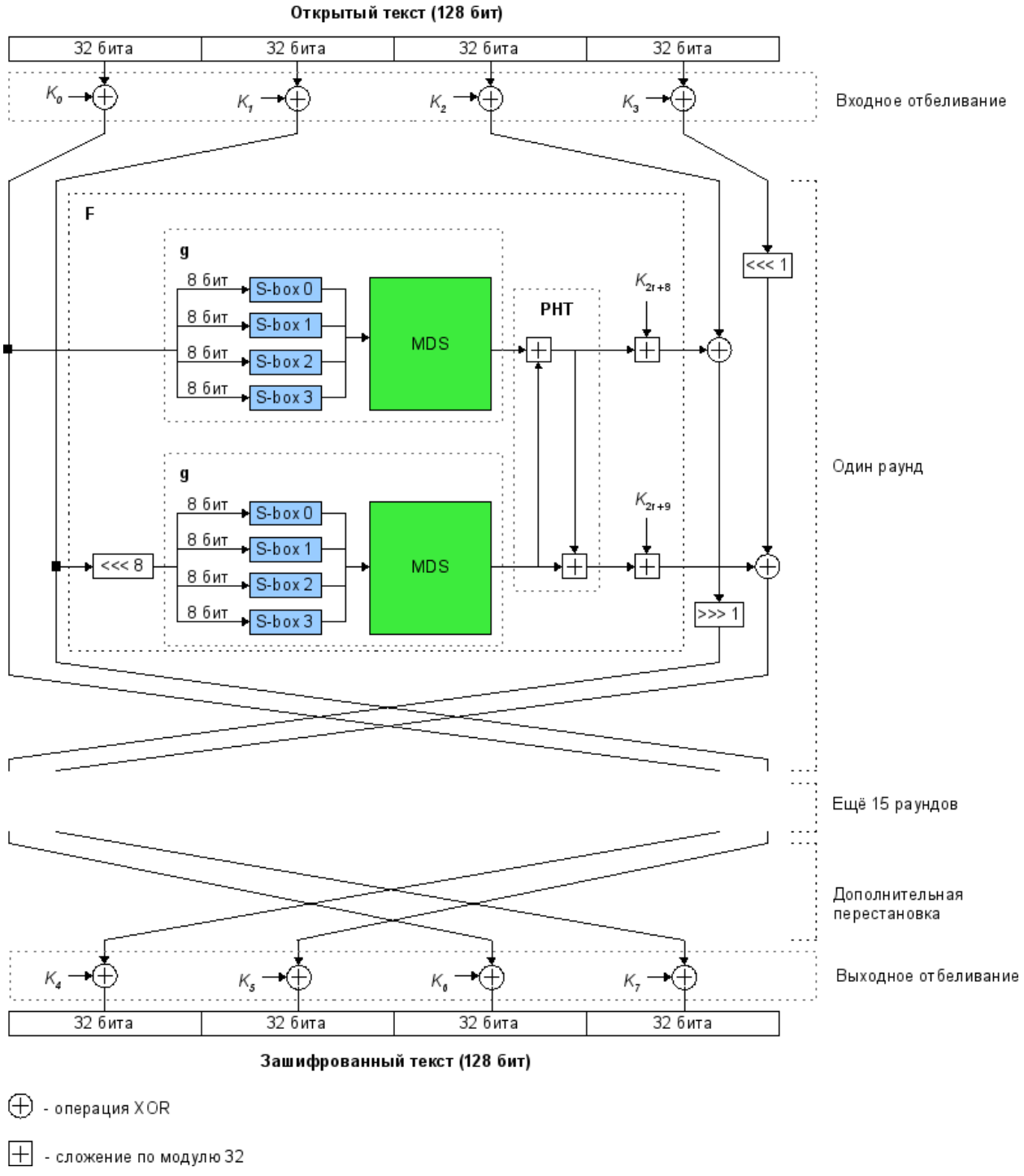


Рисунок 1.16 – Алгоритм работы Twofish

Поточные шифры строятся на основе генератора псевдослучайных битов и обрабатывают каждый бит или байт исходного текста с использованием логической операции XOR набора битов и текста.

1. RC4, также известен как ARC4 или ARCFOUR (alleged RC4) – потоковый шифр, широко применяющийся в различных системах защиты информации в компьютерных сетях. На вход генератора записывается ключ, а на выходе читаются псевдослучайные биты. Генерируемые биты имеют равномерное распределение. Общий алгоритм работы изображен на рисунке 1.17.

Рисунок 1.17 – Общий алгоритм работы RC4

Достоинства:

- высокая скорость работы;
- переменный размер ключа.

Недостатки:

- уязвим, если используются неслучайные или несвязанные ключи;
- один ключевой поток используется дважды.

2. SEAL (англ. Software-optimized Encryption Algorithm, программно-оптимизированный алгоритм шифрования) – симметричный поточный алгоритм шифрования данных, оптимизированный для программной реализации. Для кодирования и декодирования используется 160-битный ключ. Схема алгоритма работы изображена на рисунке 1.18.

Достоинства:

- высокая эффективность, особенно для 32-битных процессоров;
- алгоритм считается очень надежным.

Недостатки:

- защищён патентом.

В общем случае, в сравнении с ассиметричными алгоритмами шифрования, симметричные шифры имеют следующие преимущества:

- скорость;

- простота реализации;
- меньшая требуемая длина ключа для сопоставимой стойкости;
- изученность.

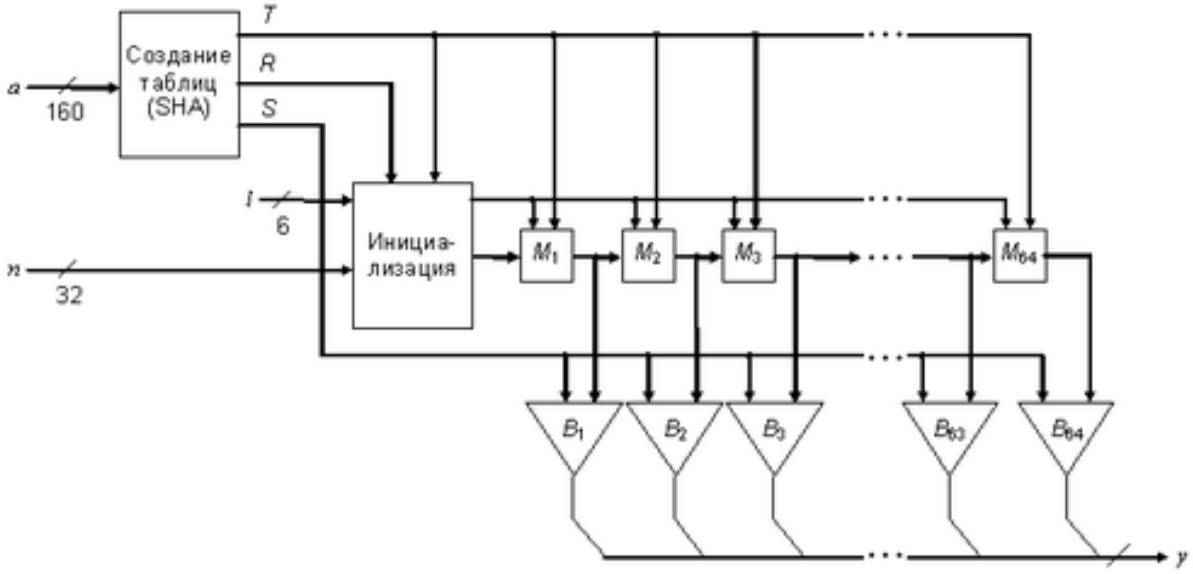


Рисунок 1.18 – Схема алгоритма SEAL

К недостаткам всех симметричных алгоритмов, по отношению к ассиметричным, можно отнести:

- сложность управления ключами в большой сети;
- сложность обмена ключами, т.к. нужно обеспечить надежную передачу ключа другой стороне;
- невозможность использовать симметричные шифры для создания цифровых подписей и сертификатов.

Ассиметричные алгоритмы – алгоритмы, использующие два ключа: открытый и закрытый. Открытый ключ предназначен для передачи по незащищенному каналу другой стороне, которая использует этот ключ для шифрования передаваемой информации и проверки электронной подписи. Закрытый ключ используют для расшифровки сообщений и генерации электронной подписи. Типичными представителями ассиметричных шифров являются:

1. RSA (аббревиатура от фамилий Rivest, Shamir и Adleman) – криптографический алгоритм с открытым ключом, основывающийся на вычислительной сложности задачи факторизации больших полупростых чисел. Исторически первая система, используемая как для цифровой подписи, так и для шифрования данных. На рисунке 1.19 изображен алгоритм работы шифра.

Достоинства:

- хорошо изучен;

• включен во многие стандарты.

Недостатки:

• низкая скорость шифрования;

• на данный момент требуется генерировать достаточно длинные ключи для обеспечения надежности шифрования информации.

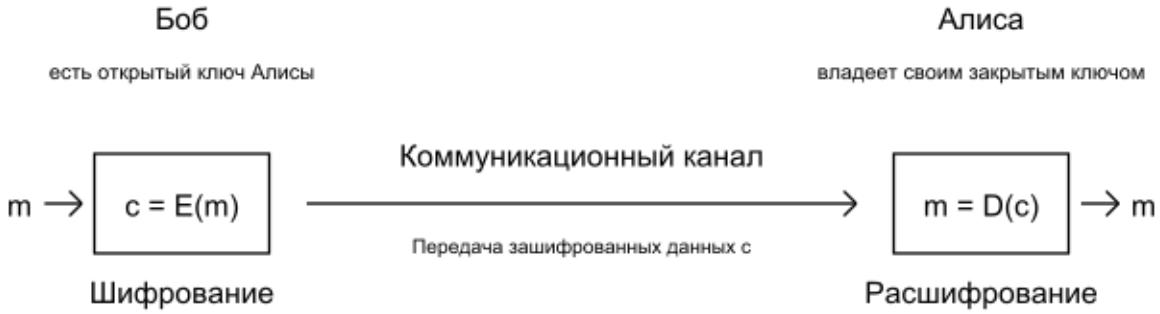


Рисунок 1.19 – Схема работы RSA алгоритма

2. Elgamal (Схема Эль-Гамаля) – криптосистема с открытым ключом, основанная на трудности вычисления дискретных логарифмов. Криптосистема включает в себя алгоритм шифрования и алгоритм цифровой подписи. Схема работы алгоритма представлена на рисунке 1.20.

Достоинства:

• вероятностный характер шифрования.

Недостатки:

• зашифрованный текст по размеру вдвое больше незашифрованного.

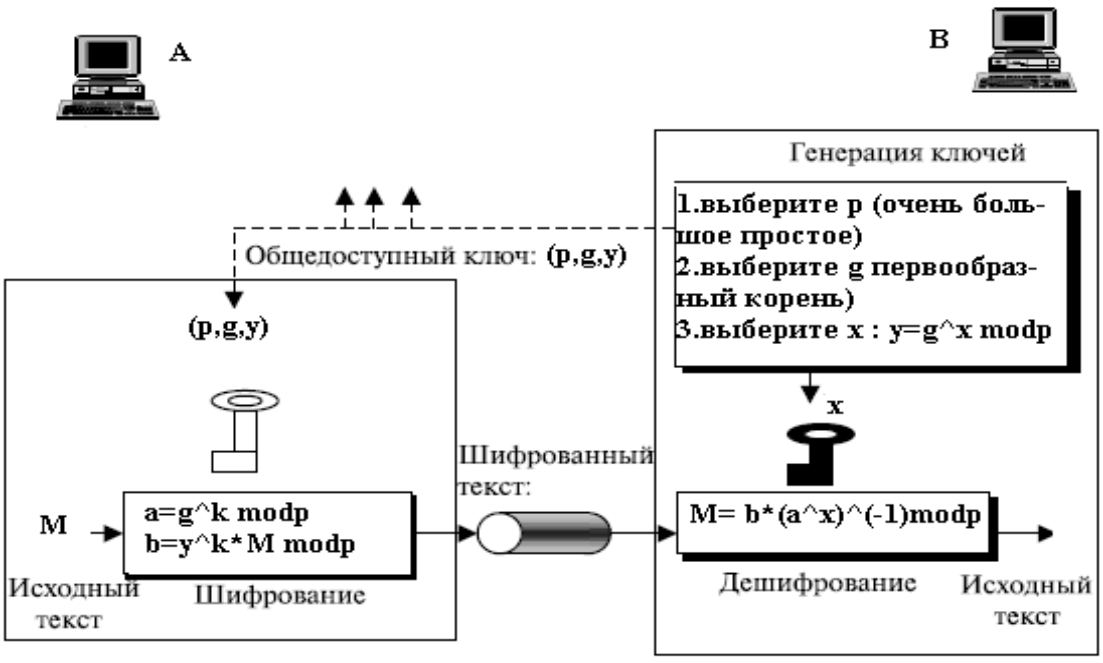


Рисунок 1.20 – Схема работы Elgamal

1.4 Выбор средств реализации и описание функций разрабатываемой системы

Мессенджер Juniorgram реализован на языке C++ в объектно-ориентированной парадигме. Разработка ведется с использование среды разработки Microsoft Visual Studio на сервисе GitHub. При разработке системы шифрования обязательным требованием является использование библиотеки CryptoPP.

C++ – компилируемый, статически типизированный язык программирования общего назначения. C++ широко используется для разработки программного обеспечения, являясь одним из самых популярных языков программирования. Область его применения включает создание операционных систем, разнообразных прикладных программ, драйверов устройств, приложений для встраиваемых систем, высокопроизводительных серверов, а также компьютерных игр. Существует множество реализаций языка C++ для различных платформ.

Microsoft Visual Studio – интегрированная среду разработки программного обеспечения. Данный продукт позволяет разрабатывать консольные приложения, игры и приложения с графическим интерфейсом.

GitHub – крупнейший веб-сервис для хостинга IT-проектов и их совместной разработки. Веб-сервис основан на системе контроля версий Git Git – распределённая система управления версиями. Сервис бесплатен для проектов с открытым исходным кодом и небольших частных проектов.

CryptoPP – это бесплатная библиотека C++ с открытым исходным кодом криптографических алгоритмов и схем. Библиотека полностью поддерживает 32-разрядные и 64-разрядные архитектуры для многих главных операционных систем и платформ, таких как Android, Apple, Linux и Windows и другие. Она предлагает готовые реализации различных криптографических алгоритмов, включая алгоритмы шифрования (симметричные и асимметричные), схемы цифровой подписи и генерации ключей.

1.4.1 Алгоритм шифрования

В качестве алгоритма шифрования был выбран AES. Этот алгоритм хорошо проанализирован и широко распространен на данный момент. Появления поддержки ускорения AES на центральных процессорах Intel в 2010 году, а затем и AMD в 2011 году, что делает алгоритм быстрее, чем он был до поддержки ускорения процессорами (см. рис. 1.21, 1.22, 1.23, 1.24).

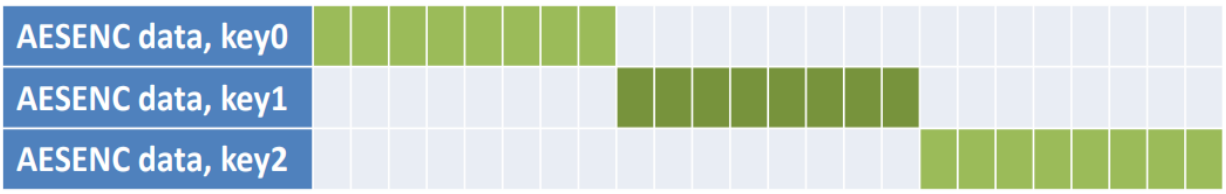


Рисунок 1.21 – Шифрование на процессоре без поддержки ускорения AES

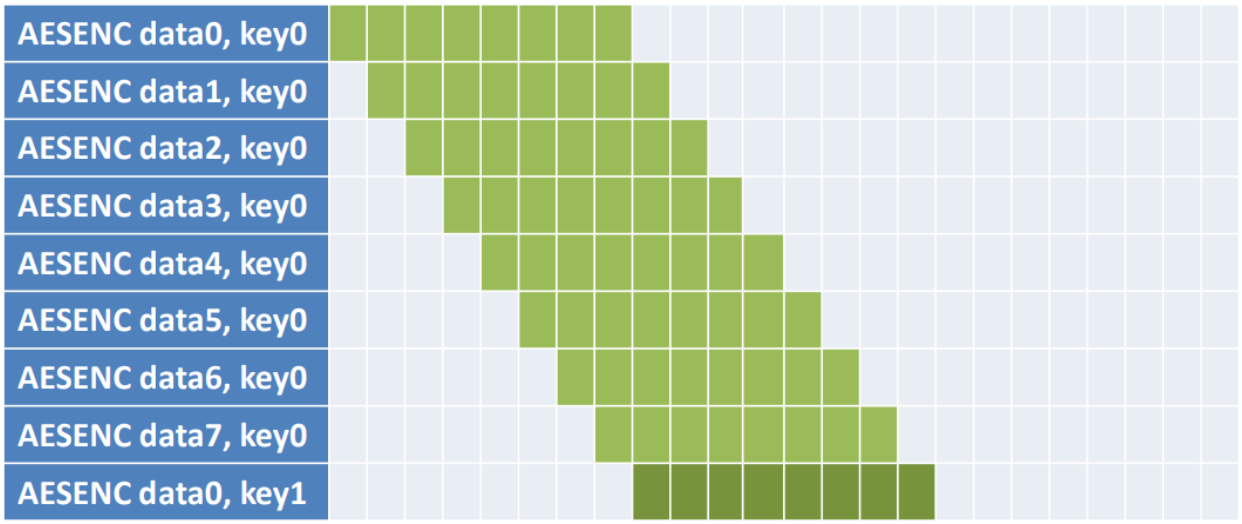


Рисунок 1.22 – Шифрование на процессоре с поддержкой ускорения AES

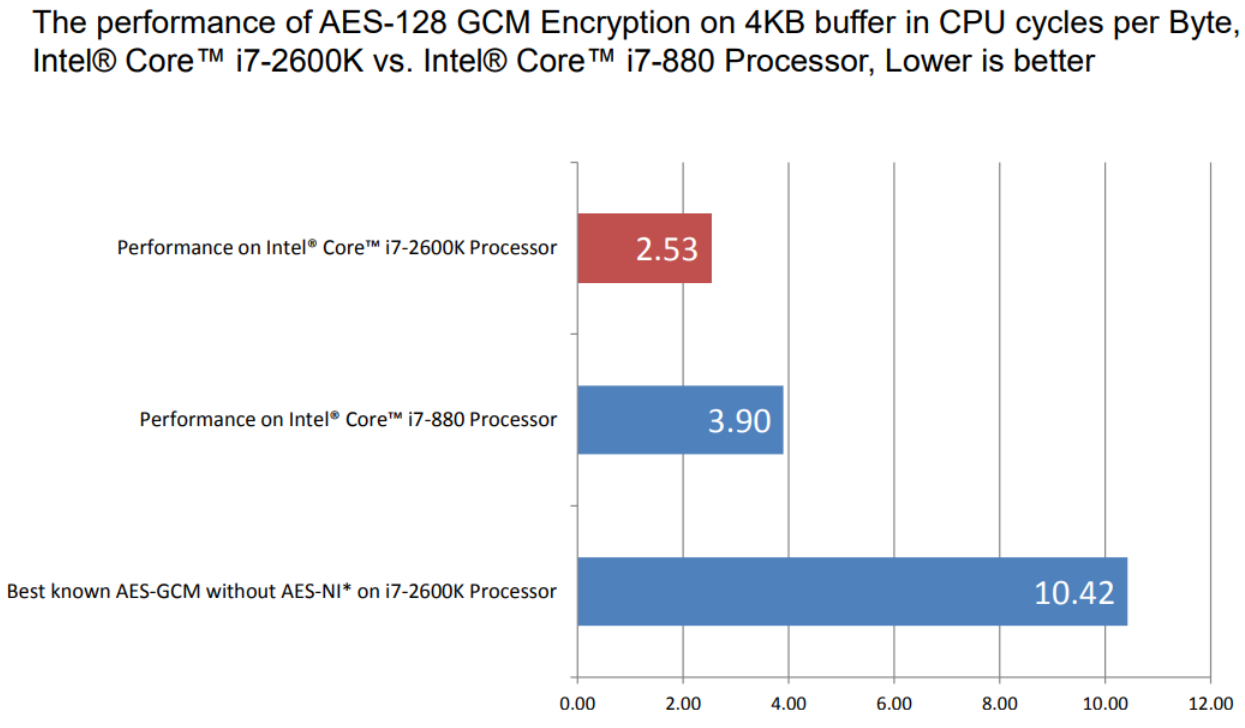


Рисунок 1.23 – Производительность AES GCM на процессорах с поддержкой ускорения AES и без него

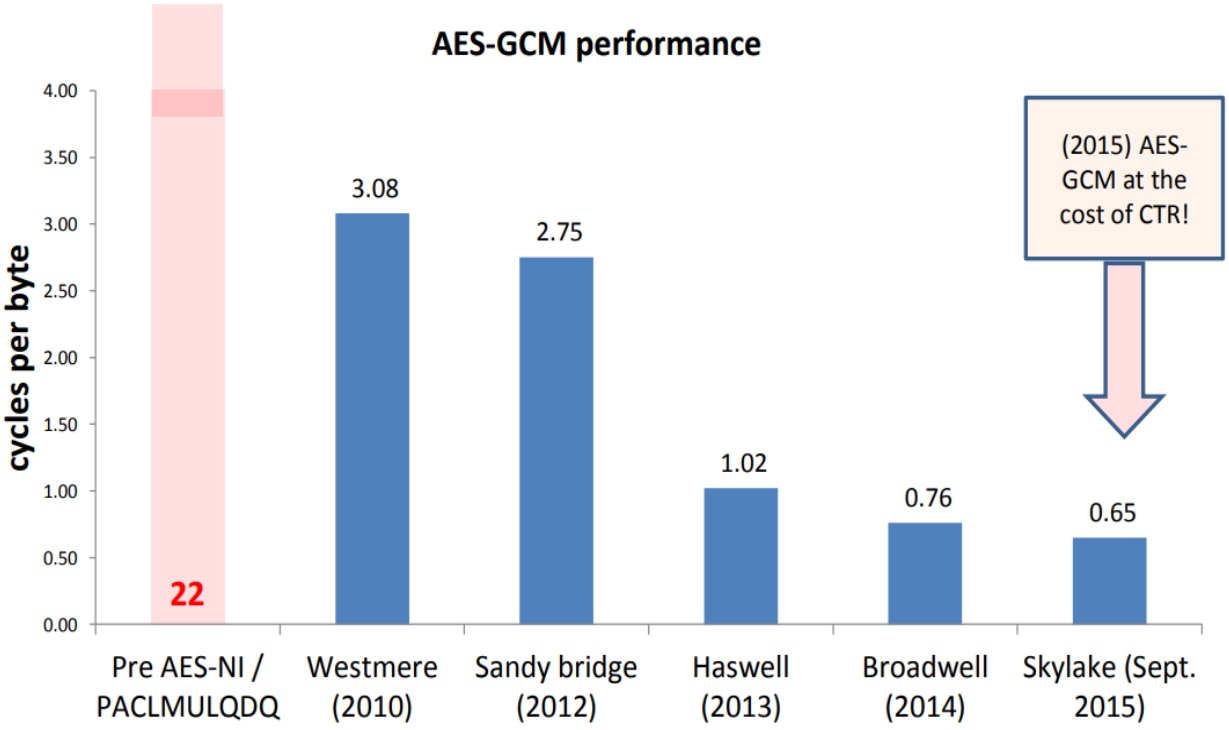


Рисунок 1.24 – Сравнение AES GCM алгоритма на процессорах разного поколения с поддержкой ускорения AES и до появления поддержки

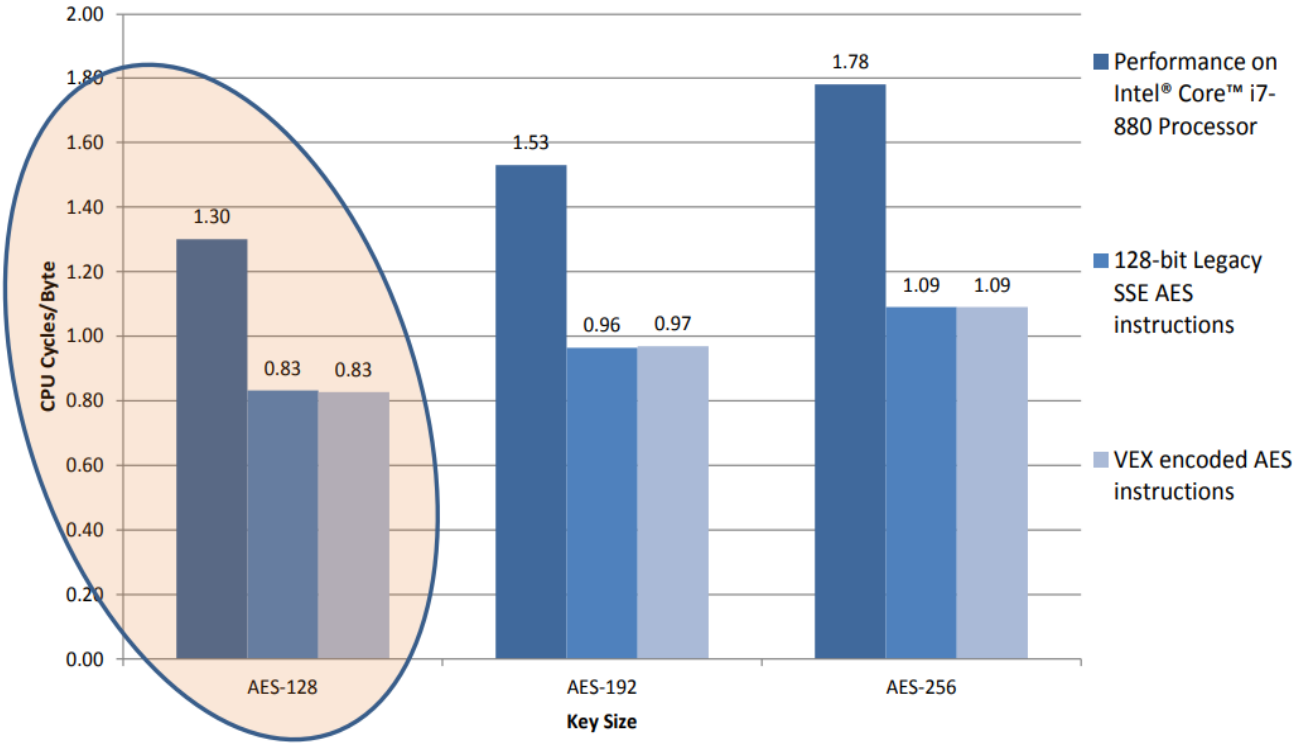


Рисунок 1.25 – Производительность режима CTR

Для AES был выбран режим GCM, т.к. он позволяет параллельное выполнение как шифрования, так и дешифрование. GCM основан на CTR режиме, который сам является высокопроизводительным (см. рисунок 1.25), а в совокупности с обеспечением

целостности и аутентификации зашифрованных данных, обеспечивая дополнительную защиту от компрометации или порчи данных, является хорошим выбором для защиты информации с отличной эффективность, по сравнению с другими комбинациями алгоритмов шифрования и хэш-функций для обеспечения целостности данных (см. рис. 1.26, 1.27).

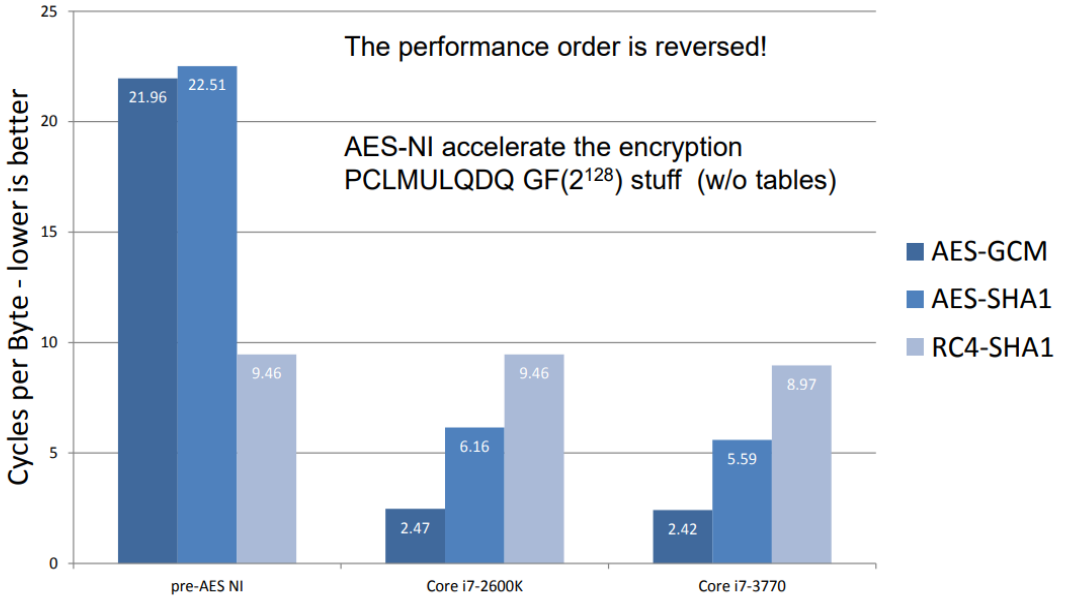


Рисунок 1.26 – Сравнение AES GCM с другими вариациями алгоритмов шифрования и целостности данных

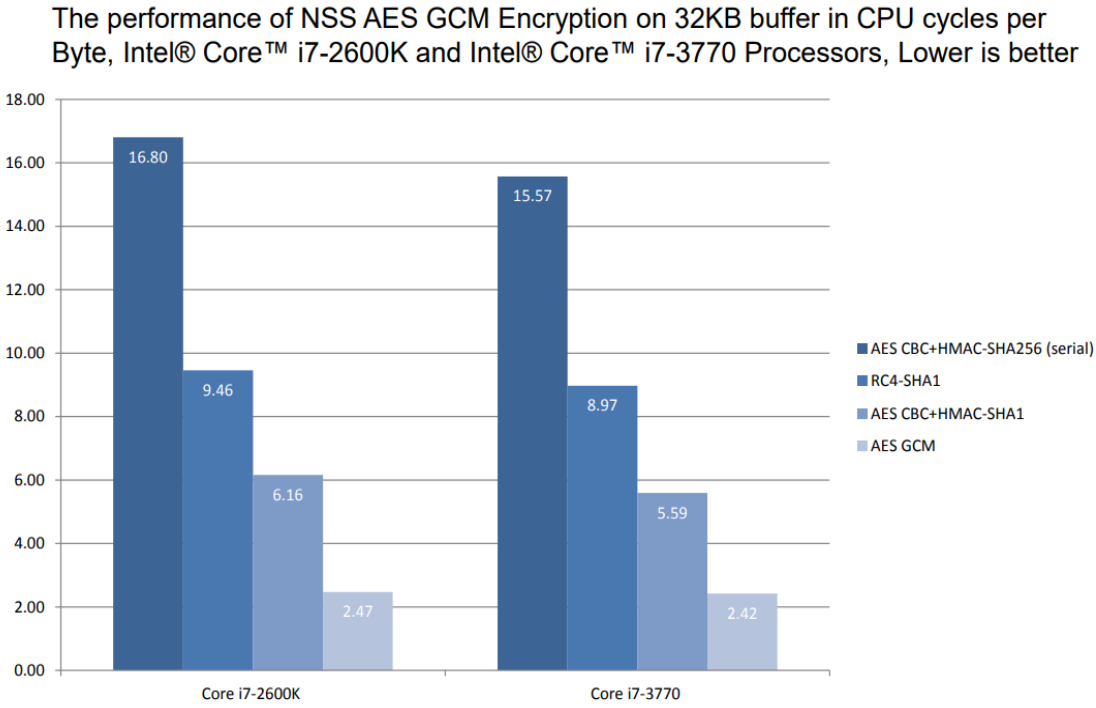


Рисунок 1.27 – Сравнение AES GCM с другими вариациями алгоритмов шифрования и целостности данных

AES GCM использует один и тот же ключ для шифрования и дешифрования, поэтому необходимо выбрать способ генерации или передачи сессионного ключ от одной стороны к другой.

1.4.2 Алгоритм генерации ключа

Для получения сессионного ключа, который будет использован как ключ шифрования данных, выбран протокол Диффи-Хеллмана на эллиптических кривых (Elliptic Curve Diffie-Hellman, ECDH). Этот протокол является классическим решением для генерации ключа, а использование эллиптических кривых основано на том, что на сегодняшний день неизвестны субэкспоненциальные алгоритмы дискретного логарифмирования. Был выбран вариант эфемерно-эфемерный ECDH для обеспечения большей случайности при генерации сессионного ключа.

Недостатком эфемерно-эфемерной реализации протокола является его уязвимость в атаке «Человек посередине». Суть атаки заключается в том, что злоумышленник устанавливает связь с двумя сторонами, которые считают, что общаются непосредственно друг с другом, и тайно ретранслирует их сообщения, искажая информацию при необходимости. В данной атаке злоумышленник устанавливает отдельные каналы связи с каждой стороной, и генерация ключа происходит с его участием, что дает ему возможность расшифровывать защищенные сообщения.

1.4.3 Предотвращение атаки «Человек посередине»

Mihai Ordean and Mircea Giurgiu в статье « Towards Securing Client-Server Connections against Man-in-the-Middle Attacks» [10] предлагают использовать строку аутентификации для защиты от атаки «Человек посередине». Суть защиты заключается в том, что клиент вычисляет хэш из конкатенации публичного ключа сервера, своего пароля и идентификатора соединения, который генерируется случайно каждое новое соединение. При этом данный хэш шифруется публичным ключом. Таким образом, для предотвращения атаки используется инфраструктура открытых ключей для реализации некоторых функциональных возможностей протокола обмена ключами с проверкой подлинности пароля. Необходимость шифровать верификационный хэш открытым ключом подразумевает использование асимметричного алгоритма шифрования.

1.4.4 Ассиметричный алгоритм шифрования

Для асимметричного шифрования был выбран RSA в силу его хорошей изученности и всеобщего признания. С помощью этого алгоритма с открытым ключом можно создавать электронные цифровые подписи. Цифровая подпись будет использоваться как подтверждение того, что клиент работает с настоящим сервером, а не злоумышленником.

1.4.5 Подтверждение ключа

Использование симметричного алгоритма шифрования с аутентификацией в совокупности с использованием ECDH для генерации общего ключа требует, чтобы обе стороны подтвердили, что владеют одинаковым ключом. В противном случае, при расшифровке сообщения ключом, отличным от ключа, который был использован при шифровании, произойдет сбой аутентификации, что не позволит сторонам обмениваться информацией, используя эти ключи. Согласно NIST SP 800-152 [2], одним из способов подтвердить идентичность ключей обеих сторон является использование тестового сообщения с заранее известным содержимым. Такой способ подтверждения ключа и был выбран в силу его простоты и эффективности.

1.4.6 Длина ключей

Для приложения Juniorgram было решено использовать уровень безопасности в 128 бит, согласно рекомендациям NIST [9]. Это означает, что ключи для ранее выбранных элементов криптографической системы имеют следующие размеры:

- AES GCM – 128 бит;
- ECDH – 256 бит;
- RSA – 3072 бита.

В соответствии с выбранным уровнем безопасности было решено использовать хэш-функцию SHA-256.

Приведение размера ключа, полученного в результате работы алгоритма Диффи-Хеллмана, к размеру ключа, необходимому для алгоритма AES в режиме GCM, будет

происходить путем разделения 256-битного ключа на две равные части и применения над ними операции исключающего ИЛИ.

По прогнозам NIST, данные длины ключей и выбранная хэш-функция обеспечат достаточную защиту в периоде с 2019 года по 2030 год и позже. Выбор таких размеров ключей обеспечивает отсутствие слабого звена во всей криптографической системе, что положительно сказывается на защите приложения.

2 Проектирование системы

Разрабатываемая в рамках дипломного проекта криптографическая система для приложения Juniorgram будет представлять собой совокупность модулей, используемые клиентской и серверной сторонами. При разработке шифрования произойдет как создание новых модулей, так и модификация уже существующих.

Криптографическая защита будет состоять из следующих модулей:

1. Модуль симметричного шифрования.
2. Модуль асимметричного шифрования.
3. Модуль генерации ключа шифрования.
4. Модуль подтверждения ключа.
5. Модуль верификации соединения.
6. Модуль авторизации.
7. Хранилище для симметричных ключей.
8. Генератор псевдослучайных чисел (далее – ГПСЧ).
9. Модуль хэширования.

2.1 Проектирование модуля симметричного шифрования

Данный модуль предназначен для шифрования всех данных, передаваемых клиентом и сервером. При шифровании и дешифровании используется ключ, полученный в результате работы модуля для его генерации. Для обеспечения гибкости приложения в выборе алгоритма шифрования, модуль будет спроектирован и реализован, опираясь на паттерн проектирования «Стратегия».

Паттерн – типовое решения для часто встречающейся проблемы в проектировании программного обеспечения.

Паттерн «Стратегия» – поведенческий паттерн проектирования, который определяет семейство схожих алгоритмов и помещает каждый из них в собственный класс, после чего алгоритмы можно взаимозаменять прямо во время исполнения программ [11].

Основываясь на данном паттерне, была спроектирована следующая иерархия классов (см. рисунок 2.1), в которой имеются классы *ICryptography* и *AES_GCM*.

Класс *ICryptography* обеспечивает возможность замены алгоритма шифрования во время работы программы.

Класс *AES_GCM* будет отвечать за шифрование с использованием соответствующего алгоритма. Данный класс будет обеспечивать конфиденциальность,

целостность и аутентификацию данных. Подтверждение авторства сообщения будет основано на использовании псевдонима отправителя.

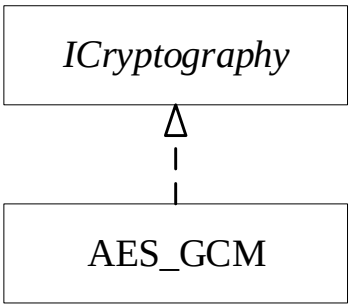


Рисунок 2.1 – Иерархия классов в модуле симметричного шифрования

2.2 Проектирование модуля асимметричного шифрования

Данный модуль предназначен для реализации выбранного ранее асимметричного алгоритма, т.е. RSA. Руководствуясь принципами SOLID, в частности принципом единственной ответственности, который гласит, что каждый класс должен отвечать на одну определенную функцию внутри модуля будут классы, каждый из которых будет решать определенную задачу. Исходя из того, что ранее в приложении Juniorgram отсутствовало асимметричное шифрование, появилась необходимость ее реализации и интеграции в приложение. Данный модуль предназначен, в первую очередь, как часть защиты от атаки «Человек посередине», согласно статье « Towards Securing Client-Server Connections against Man-in-the-Middle Attacks» [10], однако, с небольшой модификацией клиентской стороны, может полноценно использоваться как алгоритм шифрования данных.

Данный модуль будет реализован с использованием трех классов:

1. *RSAKeyManager* – класс, главной задачей которого будет загрузка RSA-ключей из файла с жесткого диска. После загрузки ключей в оперативную память будет производится проверка этих ключей на их достоверность. В данной реализации класс будет интегрирован в серверную часть.

2. *RSA* – класс, реализующий шифрование, руководствуясь соответствующим алгоритмом. Данный класс будут использовать и клиент, и сервер.

3. *RSAKeyGenerator* – класс, ответственный за генерацию приватного и публичного RSA-ключей. Будет функционировать при отсутствии пары RSA ключей сервера на жестком диске либо, если ключи были скомпрометированы или повреждены.

2.3 Проектирование модуля генерации ключа

Данный модуль реализует способ получения одинакового ключа шифрования для клиента и сервера без его непосредственной передачи. Модуль будет спроектирован на основе вышеуказанного паттерна «Стратегия», поэтому была получена следующая иерархия классов (см. рисунок 2.2), которой будет иметься два класса: *IKeyAgreement* и *ECDH*.

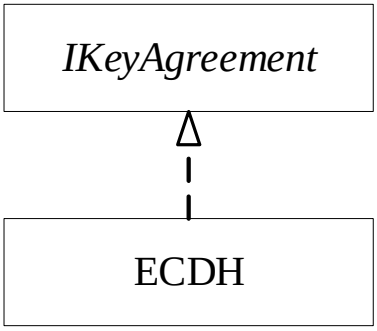


Рисунок 2.2 – Иерархия классов в модуле генерации ключа

Класс *IKeyAgreement* по цели существования аналогичен классу *ICryptography*, т.е. обеспечивает заменимость алгоритма генерации общего ключа во время работы приложения. Класс *ECDH* реализует выбранный алгоритм формирования ключа шифрования, т.е. реализует алгоритм Диффи-Хеллмана на эллиптических кривых. В данной реализации для генерации ключа будет использована кривая NIST P-256. После удачной генерации общего ключа, временные ключи, на основе которых и был сгенерирован ключ, уничтожаются.

В текущей реализации предполагается 2 попытки для генерации. Если же все они закончились неудачей, канал связи закрывается.

2.4 Проектирование модуля подтверждения ключа

Данный модуль реализует способ подтверждения ключа шифрования, полученного в результате работы модуля генерации ключа. Согласно выбранному способу проверки ключа на совпадение у клиента и сервера, и, используя средства языка C++, было решено реализовать данную проверку одним шаблонным классом *KeyConfirmation*. Данный класс будет содержать некоторое сообщение, одинаковое для обеих сторон клиент-серверной архитектуры. Проверка ключа будет проводится в три этапа:

1. Клиент отправляет зашифрованное сообщение с содержимым, известным также и серверу.

2. Сервер расшифровывает полученное сообщение.

3. Сервер сверяет содержимое расшифрованного сообщения с контрольным. В случае неудачи, процесс генерации общего ключа запускается снова. Использование шаблонного класса в данном модуле дает свободу в выборе структуры данных, которая может использоваться для проверки правильности вычисленного ключа шифрования.

2.5 Проектирование модуля верификации соединения

Проектирование данного модуля необходимо для компенсации слабости к атаке «Человек посередине» алгоритма Диффи-Хеллмана, который используется для генерации ключа шифрования. Структура модуля будет основана на паттерне «Стратегия», что позволит динамически выбирать способ верификации. Данный модуль будет генерировать строку аутентификации на основе полученных данных о соединении клиента с сервером. Исходя из вышесказанного, будет получена архитектура, состоящая из двух классов: *IConnectionVerifier* и *HashVerifier* (см. рисунок 2.3).

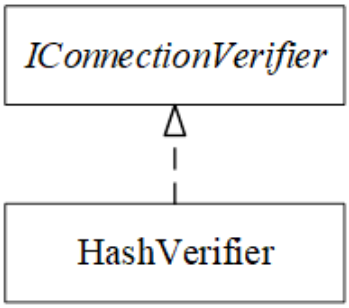


Рисунок 2.3 – Иерархия классов в модуле верификации соединения

2.6 Проектирование модуля авторизации

Ранее существовавший в приложении модуль авторизации недостаточно надежен для обеспечения полноценной защиты приложения, поэтому он будет модифицирован, согласно статье « Towards Securing Client-Server Connections against Man-in-the-Middle Attacks» [10]. Таким образом, для авторизации пользователя необходимо предоставить ему информацию о текущем соединении серверу, которую он сможет получить по

запросу. На основе этой информации модуль верификации соединения сформирует аутентифицирующую строку, которую и будет сверять сервер при авторизации клиента.

2.7 Проектирование хранилища симметричных ключей

Появление шифрования в приложении Juniorgram обязывает предусмотреть место для хранения симметричного ключа. Для этого было решено создать класс *SessionKeyHolder*, который будет ответственен за хранение ключей в течении одного сеанса. Данный класс будет разработан, опираясь на паттерн «Одиночка». Данный паттерн обеспечивает существование единственного объекта класса в программе, а, значит, единственного места хранения симметричных ключей.

2.8 Проектирование генератора псевдослучайных чисел

Основной задачей ГПСЧ является генерация вектора инициализации, который будет использован для шифрования симметричным алгоритмом AES GCM. Данный вектор должен быть уникальным для шифрования, что обеспечивается ГПСЧ, имеющимся в библиотеке CryptoPP. Данный ГПСЧ самостоятельно инициализируется с помощью операционной системы и генерирует криптографически стойкие случайные числа, что удовлетворяет требованию об уникальности вектора инициализации. Данный генератор чисел будет реализован в классе *ByteBlockGenerator*.

2.9 Проектирование модуля хэширования

Данный модуль, в первую очередь, предназначен для вычисления строки аутентификации. Используемая криптографическая библиотека реализует различные хэш-функции, в том числе и выбранную раннее для вычисления строки аутентификации, SHA-256.

3 Реализация и испытание

Криптографическая система разрабатывалась на языке высокого уровня C++ в интегрированной среде разработки Microsoft Visual Studio, с использованием системы контроля версий Git, криптографической библиотеки CryptoPP, системы сборки проектов CMake на платформе GitHub. Все разрабатываемые классы находятся в отдельных файлах с расширениями .hpp и .cpp.

В ходе проектирования было решено разбить криптографическую защиту приложения Juniorgram на самостоятельные модули, и, используя документацию библиотеки CryptoPP [3], была выполнена реализация и изменение ранее указанных модулей криптографической защиты.

3.1 Реализация модуля симметричного шифрования

Данный модуль состоит из классов *ICryptography* и *AES_GCM*, каждый из которых находится в собственном файле.

В классе *ICryptography* были реализованы два метода.

```
encrypt(yas::shared_buffer& buffer, const SecByteBlock& key, const SecByteBlock& iv, const std::string authData).
```

Метод выполняет шифрование данных, содержащихся в buffer, используя ключ key, вектор инициализации iv и данные для аутентификации authData.

```
decrypt(yas::shared_buffer& buffer, const SecByteBlock& key, const SecByteBlock& iv, const std::string authData).
```

Метод выполняет дешифрование данных, содержащихся в buffer, используя ключ key, вектор инициализации iv и данный для аутентификации authData.

Методы данного класса не реализуют никакой логики, а служат для указания возможных действий для алгоритмов симметричного шифрования.

Класс *AES_GCM* содержит методы, аналогичные классу *ICryptography*, в которых реализовано шифрование алгоритмом AES GCM. Данный класс использует коды возврата, которые указывают на успешность или неудачу процесса шифрования. Если процесс шифрования или расшифрования закончился неудачей, т.е. была нарушена

целостность сообщения или его авторство, данное сообщение игнорируется приложением.

3.2 Реализация модуля асимметричного шифрования

Результатом проектирования модуля ассиметричного шифрования стала необходимость его разделения на три класса, каждый из которых будет ответственен за определенные на него функции и хранится отдельно друг от друга.

Класс *RSAKeyManager* выполняет функции хранения пары RSA-ключей, их загрузки с жесткого диска и сохранение на него. Таким образом, данный класс имеет методы, которые перечислены далее.

```
RSAKeyManager(const std::string& filename).
```

Данный публичный метод используется сервером для определения имени файла, с которым в дальнейшим будет взаимодействовать класс. По умолчанию приватный ключ хранится в файле «juniorgram.rsa», а публичный – в файле «juniorgram.rsa.pub».

```
loadKeyPair().
```

Данный публичный метод вызывается при запуске сервера с целью загрузки RSA-ключей с жесткого диска. При вызове он вызывает метод loadPrivateKey().

```
loadPrivateKey().
```

Приватный метод обращается к жестокому диску сервера с целью загрузки приватного ключа. Загрузка происходит с использованием кодирования BER. После неудачной загрузки приватного ключа по причине его отсутствия на диске или недействительности, запускается процесс генерации пары RSA-ключей, который выполняется классом *RSAKeyGenerator*, после чего класс сохраняет эту пару в своем приватном поле. В противном случае происходит вызов функции loadPublicKey().

```
loadPublicKey().
```

Приватный метод выполняет загрузку публичного RSA-ключа сервера с диска с использованием кодирования ASN.1 BER. Если при загрузке ключа произошла ошибка, генерируется новый ключ на основе загруженного приватного.

`saveKeyPair()`.

Публичный метод, который вызывается при завершении работы сервера. Вызывает приватный метод savePrivateKey(), который должен сообщить об успехе, чтобы был вызван метод savePublicKey().

`savePrivateKey()`.

Приватный метод, используемый для сохранения приватного RSA-ключа на диск с использованием кодирования ASN.1 DER. Если метод сообщит об неудачном сохранении, публичный ключ не будет сохранен.

`savePublicKey()`.

Приватный метод, вызываемый в случае успешного сохранения приватного RSA-ключа. Данный метод сохраняет публичный ключ сервера на жесткий диск, используя кодировку ASN.1 DER.

`getPublicServerKeyStr()`.

Метод, предназначенный для получения публичного ключа сервера для его последующей передачи клиенту.

`getPrivateKey()`.

Метод, вызываемый при расшифровке сообщения алгоритмом RSA.

Класс *RSAKeyGenerator* отвечает за генерацию RSA-ключей в случаях их неудачной загрузки классом *RSAKeyManager*. Для данного класса определены два метода.

`generateRSAKeyPair(RSASecurityStrength _secStrength)`.

Метод, вызываемый классом *RSAKeyManager* в случае неудачной загрузки или недействительности приватного ключа. В методе указывается размер приватного ключа _secStrength, значение которого соответствует рекомендациям NIST [9]. Для текущей реализации значения этого параметра равно _128, что задает размер приватного RSA-ключа равным 3072 бита. Метод возвращает пару RSA-ключей, которая потом хранится в *RSAKeyManager*.

`generateRSAPublicKey(const RSA::PrivateKey& privateKey)`.

Метод, генерирующий публичный ключ сервера на основе данного приватного ключа privateKey. Метод вызывается в случае неудачной загрузки публичного RSA-ключа с диска.

Класс *RSA* реализует асимметричное шифрование, используя приватный ключ сервера для шифрования и публичный ключ для дешифрования. Библиотека CryptoPP предоставляет возможность конфигурации объекта, реализующего алгоритм, поэтому был выбран вариант алгоритма, определенного в PKCS #1 v2.2 [7], в котором используются случайное заполнение OAEP и хэш-функцию SHA-256. Данный класс имеет два метода.

```
encrypt(const std::string& dataForEncrypt, const CryptoPP::RSA::PublicKey& publicServerKey).
```

Данный публичный метод использует публичный ключ сервера publicServerKey для шифрования данных dataForEncrypt. Функция возвращает зашифрованное сообщение в случае успеха и пустую строку в случае ошибки.

```
decrypt(const std::string& dataForDecrypt, const CryptoPP::RSA::PrivateKey& privateServerKey).
```

Данный публичный метод используется для дешифрования данных dataForDecrypt, используя приватный ключ сервера privateServerKey. Возвращает расшифрованное сообщение.

3.3 Реализация модуля генерации ключа

Согласно результатам проектирования данного модуля и выбранному алгоритму генерации общего ключа, данный модуль имеет два класса. Текущая реализация дает 2 попытки для успешной генерации ключа шифрования, иначе сервер закрывает соединение с клиентом. Каждый класс находится в своем собственном файле.

Первый класс – *IKeyAgreement* – предназначен для указания внешнему коду о возможных функциях данного модуля. Класс хранит временные приватный и публичный ключи, на основе которых будет сгенерирован ключ шифрования. В классе имеются методы, указанные далее.

```
generateKeys().
```

Публичный метод, используемый для генерации временных ключей.

`calculateSharedKey(const SecByteBlock& publicOthersideKey).`

Публичный метод, вычисляющий сессионный ключ, используя публичный ключ другой стороны publicOthersideKey, который затем будет использоваться в симметричном шифровании.

Вышеуказанные методы не реализуют никакой логики.

`getPublicKey().`

Доступный внешнему коду метод, возвращающий публичный ключ, который затем будет передан другой стороне.

Второй класс *ECDH* наследуется от первого и реализует алгоритм Диффи-Хеллмана на эллиптических кривых. Данный класс определяет логику для публичных методов

`generateKeys()` и
`calculateSharedKey(const SecByteBlock& publicOthersideKey)`.

3.4 Реализация модуля подтверждения ключа

После безошибочного завершения работы модуля генерации ключа шифрования хорошей практикой является проверка того, что клиент и сервер имеют одинаковый ключ. При отсутствии данной проверки и случае, когда клиент и сервер имеют разные ключи симметричного шифрования, несоответствие ключей будет обнаружено только при дешифровании сообщения одной из сторон. К тому же, такая ситуация не позволяет клиенту и серверу обмениваться данными и не дает пользователю использовать приложение по назначению.

Учитывая результаты проектирования модуля и выбранный способ подтверждения, в данном модуле будет находится один класс *KeyConfirmation.* Текущая реализация будет сравнивать заранее определенное сообщение «testMessage123», которое будет храниться внутри класса, с полученным от клиента и расшифрованным на стороне сервера, который затем сообщит, имеет ли он такой же ключ, каким владеет клиент.

Модуль представлен один классом *KeyConfirmation*, содержащим следующие методы:

`KeyConfirmation(const T& verificationUnit_).`

Публичный метод, определяющий единицу *verificationUnit*, которая будет использоваться для проверки ключа шифрования. Эта единица должна быть согласована обеими сторонами.

`getVerificationUnit().`

Публичный метод, возвращающий сообщение, которое в дальнейшем будет зашифровано и передано серверу.

`compareWithVerificationUnit(const T& objToCompare).`

Публичный метод, сравнивающий полученное и расшифрованное сообщение с определенным ранее обеими сторонами.

3.5 Реализация модуля верификации соединения

Для предотвращения атаки «Человек посередине», согласно выбранному способу защиты, необходимо передать информацию о соединении клиенту, чтобы он мог сгенерировать строку аутентификации. Для получения данных о соединении клиент отправляет запрос, а сервер отправляет сообщение с необходимым содержимым: случайно генерируемым номером соединения и публичным ключом сервера.

Для реализации данного модуля было решено использовать два класса, один из которых реализует интерфейс для взаимодействия с клиентской и серверной частями, а другой выполняет генерацию строки аутентификации в соответствии с выбранном алгоритмом.

Класс *IConnectionVerifier* не реализует никакой логики для верификации соединения, а указывает другим частям приложения о доступных функциях модуля. Данный класс содержит единственный метод, который является публичным для внешнего кода.

`calculateVerifyingHash(const std::string& pwdHash, const ConnectionInfo& connInfo).`

Класс *HashVerifier* реализует ранее определенный в дипломной работе способ защиты от атаки «Человек посередине», путем добавления логики методу

calculateVerifyingHash для функционирования которого необходимо иметь хэш пароля пользователя pwdHash и данные о соединении с сервером connInfo.

`calculateVerifyingHash(const std::string& pwdHash, const ConnectionInfo& connInfo),`

После выполнения функции полученная строка используется в модуле авторизации.

3.6 Реализация модуля авторизации

В приложении уже существует модуль авторизации по логину и паролю, к которому применена хэш-функция SHA-256. Однако для защиты криптографической системы от атаки «Человек посередине» необходим более сложный способ авторизации. Для этого было решено реализовать модуль верификации соединения, создающий аутентифицирующую строку.

Данный модуль содержит публичный метод, алгоритм работы которого следующий:

1. Сервер получает от клиента данные для авторизации.
2. Вызывается функция loginUser, в которую передается логин пользователя, вычисленная строка аутентификации и метод верификации соединения.
3. Сервер получает из базы данных хэш пароля пользователя.
4. Он генерирует свою строку аутентификации.
5. Происходит сравнение полученной от клиента строки с сгенерированной сервером строкой. Если эти строки совпадают, авторизация прошла успешно, иначе – в авторизации будет отказано.

`loginUser(const LoginInfo& loginInfo, const ConnectionInfo& connInfo, std::shared_ptr<IConnectionVerifier> verifier).`

3.7 Реализация хранилища для симметричных ключей

Реализация шифрования в приложении Juniorgram требует места, в котором будет храниться полученный ключ шифрования. В ходе проектирования было решено

использовать класс-хранилище, в котором будут хранится симметричные ключи шифрования.

Класс *SessionKeyHolder* хранит ключи шифрования в виде «ключ-значение», где ключом является идентификатор пользователя, а значением – ключ шифрования. Данное хранилище идентично и для клиента, и для сервера, при этом у клиента сервер всегда имеет идентификатор 1. Данный класс имеет методы, которые указаны далее.

```
setKey(SecByteBlock&& newKey, const uint64_t& userId).
```

Метод, используемый для добавления ключа шифрования в хранилище. Для этого в метод передают симметричный ключ newKey и идентификатор пользователя userId, которому этот ключ будет соответствовать.

```
getKey(const uint64_t& userId).
```

Метод, используемый для получения симметричного ключа шифрования пользователя с указанием его идентификатора userId.

```
removeKey(const uint64_t& userId).
```

Метод, удаляющий ключ шифрования пользователя с указанным идентификатором userId. Удаление происходит при отключении клиента от сервера.

3.8 Реализация генератора псевдослучайных чисел

Необходимость наличия ГПСЧ обусловлена выбранным алгоритмом симметричного ключа шифрования, поэтому необходимо дополнительно реализовать механизм, создающий уникальную последовательность битов.

Для данного модуля было решено использовать ГПСЧ, предоставляемый библиотекой CryptoPP. Согласно результатам проектирования, генератор случайных чисел будет реализован в классе *ByteBlockGenerator*, который будет иметь метод для генерации блока байтов размером, указанным в параметре blockSize.

```
generateBlock(size_t blockSize).
```

3.9 Реализация модуля хэширования

Для реализации хэширования было решено использовать функцию SHA-256. Данная функция будет достаточно надежной для разрабатываемой криптографической защиты в приложении Juniorgram. Было решено не использовать отдельный класс для реализации данного модуля, т.к. в нем нет необходимости. Таким образом, в данном модуле будет находится метод, вычисляющий хэш по алгоритму SHA-256, поэтому в модуле имеется только один метод, принимающий строку plainText, хэш которой нужно вычислить.

```
SHA_256(const std::string& plainText).
```

3.10 Тестирование системы

Для проверки корректности работы разработанной криптографической системы использовалось модульное тестирование. Для каждого нового модуля в приложении были написаны тесты с использованием фреймворка Catch2.

Фреймворк – готовое решение, использующееся для определенный целей. В данном случае – для упрощения проведения тестирования модулей.

Тесты для всех разработанных модулей находятся в файле CryptoTests.cpp.

Данный фреймворк определяет структуру модульных тестов, используя тестовые примеры и секции. Тестовый пример обозначается ключевым словом *TEST_CASE*, секция – *SECTION*. Все тесты будут проводится с использованием далее указанных команд.

```
REQUIRE(выражение).
```

Команда, ожидающая, что *выражение* после выполнения примет значение ПРАВДА. Если *выражение* принимает значение ЛОЖЬ, данный тест будет указан как проваленный.

```
REQUIRE_NOTHROW(функция).
```

Команда, используемая для проверки наличия исключений, которые выбрасывает *функция* во время своего выполнения. В текущей реализации все функции новых модулей

не используют выбрасывание исключений для сигнализации об ошибке в процессе обработки. Почти все методы классов разработанных модулей сообщают о результате работы, используя коды УСПЕХ и НЕУДАЧА. В зависимости от возвращаемого значения, вне криптографического модуля выполняется нужный алгоритм.

Исходя из разработанных криптографических модулей, файл для тестирования будет структурно разделен следующим образом.

```
TEST_CASE("Hash functions test").
```

Данный блок предназначен для тестирования модуля хэширования.

```
SECTION("SHA_256 method without salt test").
```

Данная секция предназначена для тестирования разработанной хэш-функции. В блоке производится проверка правильности работы функции SHA_256.

```
TEST_CASE("RSA components test").
```

Блок содержит тесты для классов модуля асимметричного шифрования.

```
SECTION("RSAKeyGenerator test").
```

Секция содержит тесты для класса *RSAKeyGenerator*. Здесь производится проверка на корректную генерацию RSA-ключей.

```
SECTION("RSAKeyManager test").
```

Секция содержит тесты для класса *RSAKeyManager*. Тесты этого блока проверяют корректность функций загрузки и сохранения асимметричных ключей, методов для получения приватного и публичного ключей сервера и указания имени файла, из которого ключи должны быть загружены и сохранены после завершения работы сервера.

```
SECTION("RSA encryption test").
```

Данная секция выполняет проверку корректности асимметричного шифрования, реализованного в классе *RSA*, на определенном сообщении.

```
TEST_CASE("ByteBlockGenerator test").
```

Блок служит для проверки элементов модуля, содержащего ГПСЧ.

```
SECTION("Check on randomness").
```

Данный блок реализует проверку ГПСЧ на уникальность генерируемых им значений.

```
TEST_CASE("ConnectionVerifier test").
```

Данный блок содержит тесты, проверяющие правильность вычисления строки аутентификации функции модуля верификации соединения.

```
SECTION ("HashVerifier test").
```

В данном блоке производится проверка корректной работы верификатора соединения, выбранного в дипломном проекте.

```
TEST_CASE("KeyConfirmation test").
```

В блоке выполняется проверка методов класса *KeyConfirmation*, ответственного за совпадение содержимого двух сообщений, из модуля подтверждения ключа.

```
TEST_CASE("Symmetric encryption test").
```

Блок, в котором реализована проверка модуля симметричного шифрования. В нем выполняется подготовка классов для дальнейшего тестирования в секциях этого блока.

```
SECTION ("AES GCM test").
```

В данной секции происходит подготовка класса *AESGCM* и данных, которые будут использованы при тестировании этого класса.

```
SECTION("Sended message has not any problems").
```

В секции происходит контроль за шифрованием и дешифрованием класса симметричного шифрования *AES_GCM* в ситуация, не имеющих ошибок.

```
SECTION("Sended message has integration or verification problems").
```

В секции происходит контроль за поведением класса *AES_GCM* в ошибочных ситуациях.

```
TEST_CASE("KeyAgreement test").
```

Данный блок предназначен для тестирования модуля генерации ключа.

```
SECTION ("ECDH test").
```

В этом блоке происходит тестирование генерации временных ключей и вычисления ключа шифрования в классе *ECDH,* реализующий алгоритм Диффи-Хеллмана на эллиптических кривых.

```
TEST_CASE("SessionKeyHolder test").
```

Данный блок содержит тесты для хранилища симметричных ключей.

```
SECTION("Correct situations").
```

Секция содержит тесты, проверяющие работоспособность хранилища ключей в безошибочных ситуациях.

```
SECTION("Incorrect situations").
```

Секция содержит тесты, контролирующие поведения класса *SessionKeyHolder* в ситуациях с ошибками. Производится тестирование ситуаций, когда невозможно установить ключ шифрования для определенного пользователя, ключ шифрования отсутствует для указанного клиента, удаление ключа несуществующего пользователя.

Вывод результатов тестирования всех криптографических модулей представлен на плакате «Результаты работы системы».

3.11 Стойкость криптосистемы к атаке «Человек посередине»

Разработанная криптографическая защита спроектирована для противостояния атаке «Человек посередине».

При отсутствии защиты, после успешного попадания в канал передачи данных атака злоумышленника может принимать две формы: либо читать данные, передаваемые

между клиентом и сервером и сохранять полезные данные для дальнейшего использования, либо перехватывать сообщения и ретранслировать их между двумя сторонами. Во время этой ретрансляции могут быть выполнены изменения передаваемых данных в режиме реального времени. Две формы атаки описаны далее.

3.11.1 Атака активного канала связи

На рисунке 3.1 описана атака «Человек посередине», которая активно изменяет и ретранслирует данные с клиента на сервер. В этом сценарии злоумышленник успешно представляет себя клиенту в качестве сервера, отправляя свой собственный открытый ключ, позволяющий расшифровать данные, передаваемые клиентом. По отношению к серверу злоумышленник выдает себя за обычного клиента и надеется передать данные, полученные от клиента, после того как они будут обработаны в соответствии с его потребностями.

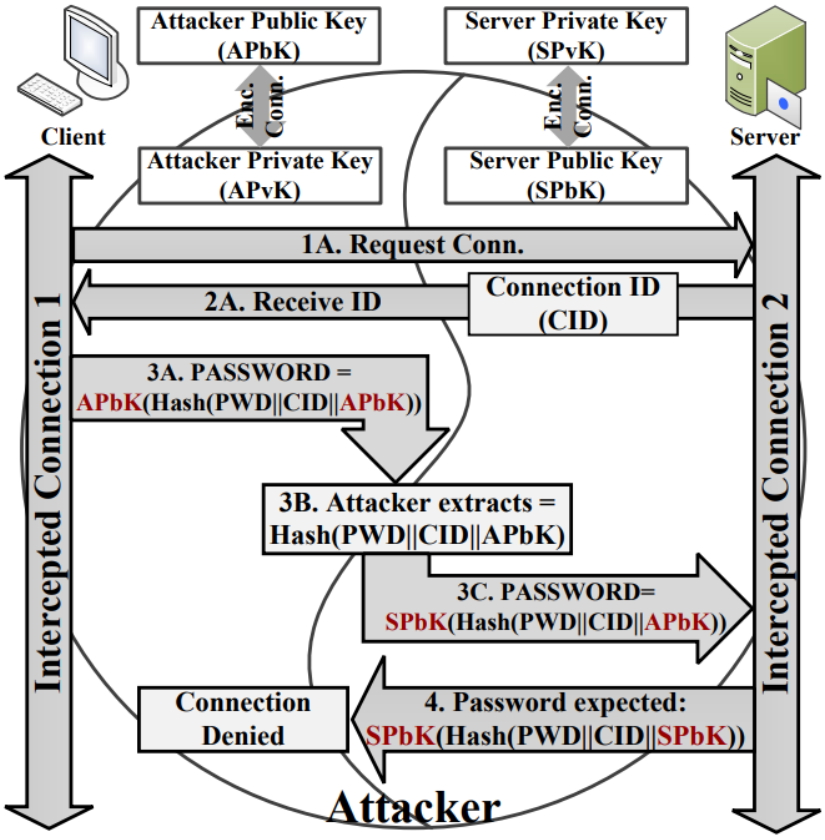


Рисунок 3.1 – Атака «Человек посередине» при активном соединении

Типичный обмен данными между клиентом и сервером в предлагаемом сценарии атаки описывается следующим образом:

1. Клиент запрашивает подключение к серверу, не зная, что сервер на самом деле является злоумышленником (шаг 1A). Злоумышленник передает запрос от своего имени реальному серверу (шаг 1B).

2. Сервер отвечает уникальным идентификатором соединения, который злоумышленник передает клиенту в неизмененном виде (шаги 2A, 2B). Целью злоумышленника является получение строки аутентификации, поэтому желательным подходом для злоумышленника является передача неизмененного идентификатора соединения.

3. Клиент создает строку аутентификации на основе полученного идентификатора подключения, пароля пользователя и открытый ключ для активного соединения (т.е. соединения с злоумышленником). Чтобы предотвратить возможную подмену компонент для вычисления строки аутентификации, при которой злоумышленник может обманом заставить пользователя внедрить открытый ключ законного сервера вместо открытого ключа подключения, хэш также шифруется с помощью открытого ключа подключения. Затем это сообщение, используемое при аутентификации, отправляется обратно злоумышленнику, выдающему себя за сервер.

4. Злоумышленник на этом этапе может расшифровать полученное сообщение и получить обычный хэш, но повторное шифрование этого хэша с помощью открытого ключа законного сервера привело бы к отказу в авторизации на реальном сервере (шаги 3A, 3B, 3C и 4) из-за открытого ключа, встроенного в хэш.

Злоумышленник сталкивается с проблемой воссоздания хэша, зная идентификатор соединения и открытый ключ, и не зная пароля пользователя, который и был целью атаки

3.11.2 Атака повторного воспроизведения

Атаку повторного воспроизведения (см. рисунок 3.2) можно рассматривать как частный случай атаки «Человек посередине» в реальном времени, которая происходит со значительной задержкой между полученными данными и данными, отправленными злоумышленником.

Эта атака состоит из фазы сбора и фазы воспроизведения. На этапе сбора данных злоумышленник пытается либо перехватить часть данных, которыми обмениваются клиент и сервер, либо воссоздать часть сервера и базы данных, чтобы обманом заставить пользователя предоставить свои учетные данные (т.е. фишинговая атака). На этапе воспроизведения злоумышленник использует собранные данные для аутентификации.

Эта атака очень успешна, даже если собранные данные зашифрованы или замаскированы, поскольку законный сервер не имеет возможности узнать, является ли подключающийся объект злоумышленником или законным клиентом.

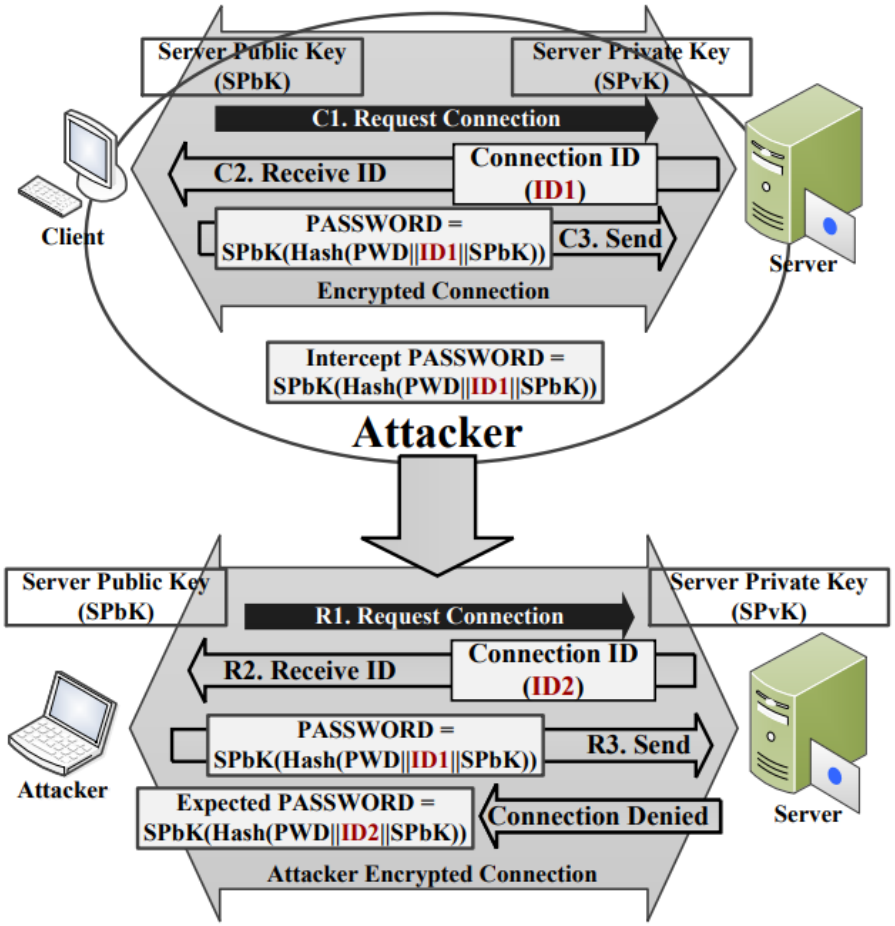


Рисунок 3.2 – Атака повторного воспроизведения

Данная атака выполняется в два этапа:

1. Злоумышленник пассивно перехватывает данные, которыми обмениваются по защищенному соединению сервер и клиент (C1, C2 и C3). Как только строка аутентификации будет перехвачена, злоумышленник сможет провести повторную атаку.

2. Когда злоумышленник пытается пройти аутентификацию позже с полученной строкой аутентификации он получит другой идентификатор подключения (R1, R2, R3).

Задача злоумышленника состоит в том, чтобы расшифровать строку и повторно вычислить хэш для нового идентификатора подключения, не зная секретного ключа сервера и пароля пользователя. Единственный возможный подход - повторять попытку подключения до тех пор, пока полученный идентификатор соединения не совпадет с идентификатором в хэше, но этого подхода можно легко избежать при случайной генерации идентификатора соединения в достаточно большом диапазоне значений.

4 Технико-экономическое обоснование

Любая разработка программного обеспечения, приложения или их модулей требует финансовых и человеческих ресурсов.

Задачей дипломного проекта является разработка шифрования, а также реализация всех необходимых модулей для корректной работы модуля симметричного шифрования, используя среду разработки Microsoft Visual Studio и язык C++.

Разработка элементов криптографической защиты приложения предусматривает проведение практически всех стадий проектирования и относится ко второй группе сложности.

Последовательность расчетов:

1. Расчёт объёма функций программных модулей.
2. Расчёт полной себестоимости криптографической защиты.
3. Расчёт цены и прибыли по программному продукту.

4.1 Расчёт объёма функций программного модуля

Общий объём ПО определяется по формуле (4.1), исходя из объёма функций, реализуемых программой.

$$V_0 = \sum_{i=0}^{n} V_i, \qquad (4.1)$$

где V_0 – общий объём ПО;

V_i – объём функций ПО;

n – общее число функций.

В том случае, когда на стадии технико-экономического обоснования проекта невозможно рассчитать точный объём функций, то данный объём может быть получен на основании прогнозируемой оценки имеющихся фактических данных по аналогичным проектам, выполненным ранее, или применением нормативов по каталогу функций.

По каталогу функций на основании функций разрабатываемых криптографических модулей определяется общий объём. Также на основе зависимостей от организационных и технологических условий, был скорректирован объём на основе экспертных оценок.

Уточнённый объём ПО определяется по формуле (4.2).

$$V_y = \sum_{i=0}^{n} V_{yi}, \quad (4.2)$$

где V_y – уточнённый объём ПО;

V_{yi} – уточнённый объём отдельной функции в строках исходного кода.

Перечень и объём функций модулей криптографии приведён в таблице 4.1.

Таблица 4.1 – Перечень и объём функций криптографических модулей

Номер функции	Наименование (содержание) функции	Объём функции строк исходного кода	
		По каталогу V_y	Уточнённый V_{yi}
303	Обработка файлов	1100	95
401	Генерация рабочих программ	3360	901
405	Система настройки ПО	370	1022
506	Обработка ошибочных и сбойных ситуаций	1720	49
602	Вспомогательные и сервисные программы	580	251

С учётом информации, указанной в таблице 4.1, уточнённый объём ПО составил 2318 строк кода вместо предполагаемого количества 7130.

4.2 Расчёт полной себестоимости программного модуля

Стоимостная оценка программного средства у разработчика предполагает составление сметы затрат, которая включает следующие статьи расходов:

1. Отчисления на социальные нужды ($Р_{соц}$).
2. Материалы и комплектующие изделия ($Р_м$).
3. Спецоборудование ($Р_с$).
4. Машинное время ($Р_{мв}$).
5. Расходы на научные командировки ($Р_{нк}$).
6. Прочие прямые расходы ($Р_{пр}$).
7. Накладные расходы ($Р_{нр}$).
8. Затраты на освоение и сопровождение программного средства ($Р_о$ и $Р_{со}$).

Полная себестоимость ($С_п$) разработки программного продукта рассчитывается как сумма расходов по всем статьям с учётом рыночной стоимости аналогичных продуктов.

Основной статьёй расходов на создание программного продукта является заработная плата проекта (основная и дополнительная) разработчиков (исполнителей) ($ЗП_{осн} + ЗП_{доп}$), в число которых принято включать инженеров-программистов, руководителей проекта, системных архитекторов, дизайнеров, разработчиков баз данных, Web-мастеров и других специалистов, необходимых для решения специальных задач в команде.

Расчёт заработной платы разработчиков программного продукта начинается с определения:

1. Продолжительности времени разработки ($Ф_{рв}$), которое устанавливается студентом экспертным путём с учётом сложности, новизны программного обеспечения и фактически затраченного времени. В данном дипломном проекте $Ф_{рв}$ – 2 месяца.

2. Количества разработчиков программного обеспечения.

В данном дипломном проекте будет разработчик: инженер-программист. Заработная плата разработчиков определятся как сумма основной и дополнительной заработной платы всех исполнителей.

Основная заработная плата $ЗП_{осн}$ каждого исполнителя определяется по формуле (4.3).

$$ЗП_{осн} = Т_{бтс} \cdot \frac{Т_к}{Ф_{эфф.р.в.}} \cdot Ф_{рв} \cdot К_{пр}, \quad (4.3)$$

где $Т_{бтс}$ – размер базовой ставки, на текущий момент равный 228 бел.руб.;

$Т_к$ – тарифный коэффициент согласно разряду исполнителя;

$Ф_{эфф.р.в.}$ – среднее количество рабочих дней;

$Ф_{р.в.}$ – фонд рабочего времени исполнителя (продолжительность разработки программного модуля), дни;

$К_{пр}$ – коэффициент премии, $К_{пр}$ = 1,5.

Тарифный коэффициент согласно 11 разряду инженера-программиста $Т_к$ = 2,65. Продолжительность разработки программного продукта – 60 дней, количество рабочих дней в месяце на текущий год составляет 21 день. Рассчитаем основную заработную плату инженера-программиста, согласно формуле 6.3:

$ЗП_{осн} = 228 \cdot \frac{2,65}{42} \cdot 60 \cdot 1,5 = 1295$ (белорусских рублей).

Дополнительная заработная плата $ЗП_{доп}$ каждого исполнителя рассчитывается от основной заработной платы по формуле (4.4).

$$ЗП_{доп} = ЗП_{осн} \cdot \frac{Н_{доп.зп}}{100\%}, \quad (4.4)$$

где $Н_{доп.зп}$ – надбавка дополнительной заработной платы, равная 20%.

Рассчитаем дополнительную заработную плату инженера-программиста, согласно формуле 6.3:

$ЗП_{доп} = 1295 \cdot \frac{20\%}{100\%} = 259$ (белорусских рублей).

Результаты вычислений внесём в таблицу 4.2.

Таблица 4.2 – Расчет заработной платы

Категория работников	Разряд	Тарифные коэффициент ($Т_к$)	$Ф_{эфф.р.в.}$, (дн.)	Коэффициент премии ($К_{пр}$)	Заработная плата, бел.руб.		
					Основная	Дополнительная	Всего
Инженер-программист	11	2,65	42	1,5	1295	259	1554
Итого	-	-	-	-	1295	259	1554

Таким образом, как видно из таблицы, заработная плата инженера- программиста составляет 1554 бел. руб.

Отчисления на социальные нужды $Р_{соц}$ определяются по формуле (4.5) в соответствии с действующим законодательством по нормативу.

$$Р_{соц} = (ЗП_{осн} + ЗП_{доп}) \cdot \frac{35\%}{100\%}. \quad (4.5)$$

Рассчитаем отчисления на социальные нужды:

$Р_{соц} = (1295 + 259) \cdot \frac{35\%}{100\%} = 543{,}9$ (белорусских рубля).

Расходы на материалы и комплектующие $Р_м$ отражают расходы на магнитные носители, бумагу, красящие ленты и другие материалы, необходимые для разработки программного продукта. Норма расхода материалов в суммарном выражении определяется в процентах к основной заработной плате (4.6).

$$Р_м = ЗП_{осн} \cdot \frac{Н_{мз}}{100\%}, \quad (4.6)$$

где $Н_{мз}$ – норма расхода материалов от основной заработной платы, в процентах.

Рассчитаем расходы на материалы и комплектующий, приняв $Н_{мз}$ равным 3%:

$Р_м = 1295 \cdot \frac{3\%}{100\%} = 38{,}9$ (белорусских рубля).

Расходы на спецоборудование $Р_с$ включают затраты на приобретение технических и программных средств специального назначения, необходимых для разработки методического пособия, включая расходы на проектирование, изготовление, отладку и другое.

В данном дипломном проекте для разработки криптографической защиты приложения приобретение какого-либо спецоборудования не предусматривалось. Так как спецоборудование не было приобретено, расходы равны нулю.

Расходы на машинное время $Р_{мв}$ включают оплату машинного времени, необходимого для разработки и отладки программного продукта. Они определяются в машино-часах по нормативам на 100 строк исходного кода машинного времени. $Р_{мв}$ определяется по формуле (4.7).

$$Р_{мв} = Ц_{мвi} \cdot \frac{V_0}{100} \cdot Н_{мв}, \quad (4.7)$$

где $Ц_{мвi}$ – цена одного машинного часа (1,6 бел. руб.);

V_0 – уточнённый общий объём машинного кода;

$Н_{мв}$ – норматив расхода машинного времени на отладку 100 строк кода в машино-часах. Принимается в размере 0,8.

Рассчитаем расходы на машинное время:

$$Р_{мв} = 1{,}6 \cdot \frac{2318}{100} \cdot 0{,}8 = 29{,}7 \text{ (белорусских рубля)}.$$

Расходы по статье «Научные командировки» $Р_{нк}$ берутся либо по смете научных командировок, разрабатываемой на предприятии, либо в процентах от основной заработной платы исполнителей (10-15%). Так как в данном проекте научные командировки не предусмотрены, данная статься не рассчитывается.

Прочие затраты $Р_{пр}$ включают затраты на приобретение специальной научно-технической информации и специальной литературы. Определяются по нормативу в процентах к основной заработной плате исполнителей. Так как специальная научно-техническая информация и специальная литература не приобреталась, то данная статья не рассчитывается.

Затраты на накладные расходы $Р_{нр}$ связаны с содержанием вспомогательных хозяйств, а также с расходами на общехозяйственные нужды. Определяется по нормативу в процентах к основной заработной плате по формуле (4.8).

$$Р_{нр} = \frac{Н_{нр}}{100\%} \cdot ЗП_{осн}, \quad (4.8)$$

где $Н_{нр}$– норматив накладных расходов.

В данном дипломном проекте норматив накладных расходов равен 40%, поэтому затраты на накладные расходы равны:

$Р_{нр} = \frac{40\%}{100\%} \cdot 1295 = 518$ (белорусских рублей).

Сумма вышеперечисленных расходов по статьям СЗ на программный продукт служит исходной базой для расчёта затрат на освоение и сопровождение программного продукта. Они рассчитываются по формуле (4.9).

$$СЗ = ЗП_{осн} + ЗП_{доп} + Р_{соц} + Р_{м} + Р_{с} + Р_{мв} + Р_{нк} + Р_{пр} + Р_{нр}, \quad (4.9)$$

тогда

$СЗ = 1295 + 259 + 543{,}9 + 38{,}9 + 29{,}7 + 518 = 2684{,}5$ (белорусских рубля).

Организация-разработчик участвует в освоении программного продукта и несёт соответствующие затраты, на которые составляется смета, оплачиваемая заказчиком по договору. Затраты на освоение $Р_{о}$ определяются по установленному нормативу от суммы затрат по формуле (4.10).

$$Р_{о} = СЗ \cdot \frac{Н_{о}}{100\%}, \quad (4.10)$$

где СЗ – сумма вышеперечисленных расходов по статьям на разработку программного продукта;

$Н_{о}$ – установленный норматив затрат на освоение. Для данного дипломного проекта принимается равной 5%.

Рассчитаем затраты на освоение продукта:

$Р_{о} = 2684{,}5 \cdot \frac{5\%}{100\%} = 134{,}3$ (белорусских рубля).

Организация-разработчик осуществляет сопровождение программного продукта и несёт расходы, которые оплачиваются заказчиком в соответствии с договором и сметой на сопровождение. Расходы на сопровождение $Р_{со}$ рассчитываются по формуле (4.11).

$$Р_{со} = СЗ \cdot \frac{Н_{со}}{100\%}, \quad (4.11)$$

где $Н_{со}$ – установленный норматив затрат на сопровождение программного продукта. Для данного дипломного проекта принимается равным 5%.

Рассчитаем расходы на сопровождение:

$Р_{со} = 2684{,}5 \cdot \frac{5\%}{100\%} = 134{,}3$ (белорусских рубля).

Полная себестоимость (СП) разработки программного продукта рассчитывается как сумма расходов по всем статьям. Она определяется по формуле (4.12).

$$СП = СЗ + Р_{о} + Р_{со}. \quad (4.12)$$

Рассчитаем полную себестоимость продукта:

$СП = 2684{,}5 + 134{,}3 + 134{,}3 = 2953{,}1$ (белорусских рубля).

Результаты вычислений занесём в таблицу 4.3.

Таблица 4.3 – Себестоимость программного продукта

Наименование статей затрат	Норматив, %	Сумма затрат, бел.руб.
Заработная плата, всего	-	1554
Основная заработная плата	-	1295
Дополнительная заработная плата	-	259
Отчисления на социальные нужды	35%	543,9
Спецоборудование	Не применялось	-
Материалы и комплектующие изделия	3%	38,9
Машинное время	-	29,7
Научные командировки	Не планировались	-
Прочие затраты	Не применялись	-
Накладные расходы	40%	518
Сумма затрат	-	2684,5
Затраты на освоение	5%	134,3
Затраты на сопровождение	5%	134,3
Полная себестоимость	-	2953,1

В результате всех расчётов полная себестоимость программного продукта составила 2953,1 бел.руб.

4.3 Расчет цены и прибыли по программному продукту

Для определения цены программного продукта необходимо рассчитать плановую прибыль П, которая рассчитывается по формуле (4.13).

$$П = СП \cdot \frac{R}{100\%}, \quad (4.13)$$

где СП – полная себестоимость программного модуля, бел. руб;

R – уровень рентабельности программного модуля.

В данном дипломном проекте уровень рентабельности равен 20%. Рассчитаем прибыль от реализации:

$П = 2953{,}1 \cdot \frac{20\%}{100\%} = 590{,}7$ (белорусских рубля).

После расчета прибыли от реализации по формуле (4.14) определяется прогнозируемая цена программного продукта без налогов $Ц_п$.

$$Ц_п = СП + П \quad (4.14)$$

Рассчитаем цену программного продукта без налогов:

$Ц_п = 2953{,}1 + 590{,}7 = 3543{,}8$ (белорусских рубля).

Отпускная цена $Ц_о$ (цена реализации) программного продукта включает налог на добавленную стоимость и рассчитывается по формуле (4.15).

$$Ц_о = СП + П + НДС_{пп}, \quad (4.15)$$

где $НДС_{пп}$ – налог на добавленную стоимость для программного продукта.

Для данного программного продукта $НДС_{пп}$ рассчитывается по формуле (4.16).

$$НДС_{пп} = Ц_п \cdot \frac{НДС}{100\%}, \quad (4.16)$$

где НДС – налог на добавленную стоимость. В настоящее время он составляет 20%.

Рассчитаем $НДС_{пп}$ и отпускную цену:

$НДС_{пп} = 3543{,}8 \cdot \frac{20\%}{100\%} = 708{,}8$ (белорусских рубля).

$Ц_о = 2953{,}1 + 590{,}7 + 708{,}8 = 4252{,}6$ (белорусских рубля).

Прибыль от реализации программного продукта за вычетом налога на прибыль является чистой прибылью ПЧ. Чистая прибыль остаётся организации-разработчику и представляет собой экономический эффект от создания нового программного продукта. Она рассчитывается по формуле (4.17).

$$ПЧ = П \cdot (1 - \frac{Н_п}{100\%}), \quad (4.17)$$

где $Н_п$ – ставка налога на прибыль. В настоящее время он равен 20%.

Рассчитаем чистую прибыль:

$ПЧ = 590{,}7 \cdot \left(1 - \frac{20\%}{100\%}\right) = 472{,}6$ (белорусских рубля).

Результаты расчётов цены и прибыли по программному продукту сведены в таблицу 4.4.

Таблица 4.4. – Расчёт отпускной цены и чистой прибыли программного модуля

Наименование статей затрат	Норматив, %	Сумма затрат, бел. руб.
Полная себестоимость	-	2953,1
Прибыль	20	590,7
Цена без НДС	-	3543,8
НДС	20	708,8
Отпускная цена	-	4252,6
Налог на прибыль	20	118,1
Чистая прибыль	-	472,6

В ходе произведенных расчетов определены основные экономические показатели: полная себестоимость – 2953,1 бел.руб.; прогнозируемая цена – 3543,8 бел. руб.; чистая прибыль – 472,6 бел.руб.

Разработанный программный продукт имеет большое количество конкурентов с более высокими ценами за реализацию приложений. Например, социальная сеть Вконтакте оценивается стоимостью в 318000 бел. руб. [13], компания Purrweb разрабатывает мобильные приложения стоимостью от 50000 бел.руб [12], компания Merehead оценивает разработку мессенджеров с криптографической защитой в 73000 бел.руб и более [8]. Несмотря на сравнение цены полноценного приложения с разработанным модулем, можно отметить, что стоимость модуля, разработанного в ходе

дипломного проектирования, является ниже среднего значения. Таким образом, рассчитанная отпускная цена на криптографический модуль, разрабатываемой в рамках данного дипломного проекта, является конкурентоспособной. При расчете цены учтены отчисления в фонд социальной защиты, а также налоги, необходимые к уплате. В конечном итоге получаем окончательную цену продукта, равную 4252,6 белорусских рубля.

Заключение

В рамках данного дипломного проекта была реализована криптографическая система, которая отвечает за авторизацию пользователей и шифрование передаваемых данных по незащищенному каналу данных. Данная система состоит из модулей симметричного и асимметричного шифрования, генерации и подтверждения ключа, верификации соединения, генератора псевдослучайных чисел и хранилища для ключей симметричного шифрования. Данные модули независимы, что обеспечивает гибкость такой системы в выборе компонент защиты.

Для контроля качества разработанной системы была реализована проверка модульными тестами, используя фреймворк Cacth2. Результаты тестов показали, что разработанные модули функционируют правильно и стойки в нестандартных ситуациях.

В результате технико-экономического обоснования было определено, что стоимость разработки составила 2946,3 белорусских рубля.

Благодаря внедренному шифрованию, пользователи приложения Juniorgram могут быть уверены в безопасности своих личных сообщений, что повысит привлекательность данного мессенджера для новых пользователей.

Список сокращений

AES – Advanced Encryption Standard
ASN.1 – Abstract Syntax Notation One
BER – Basic Encoding Rules
CBC – cipher block chaining
CBC-MAC – cipher block chaining message authentication code
CCM – counter with CBC-MAC
CFB – cipher feedback
CTR – counter
CWC – Carter–Wegman with CTR mode
DER – Distinguished Encoding Rules
EAX – encrypt-then-authenticate-then-translate
ECB – electronic codebook
ECDH – Elliptic curve Diffie–Hellman
GCM – Galois/counter
IACR – International Association for Cryptologic Research
IP – Internet Protocol
NIST – Национальный институт стандартов и технологий
OAEP – Optimal asymmetric encryption padding
OFB – output feedback
RSA – Rivest, Shamir, Adleman
SEAL – Software-optimized encryption algorithm
SHA – Secure Hash Algorithm
SOLID – single responsibility, open–closed, Liskov substitution, interface segregation, dependency inversion
TCP – Transmission Control Protocol
ГПЧС – генератор псевдослучайных чисел
ПО – программное обеспечение
СНГ – Содружество Независимых Государств
СССР – Союз Советских Социалистических Республик

Список использованных источников

1. Advanced Encryption Standard (AES) / National Institute of Standards and Technology ; ed.: J. Kelsey. – Gaithersburg : National Institute of Standards and Technology, 2023. – 38 p.

2. Barker, E. A Profile for U.S. Federal Cryptographic Key Management Systems / E. Barker, D. Branstad, M. Smid – SP 800-152. – Gaithersburg : National Institute of Standards and Technology, 2015 – 137 p.

3. Crypto++ Library [Электронный ресурс]. – Режим доступа: https://cryptopp.com/wiki/Main_Page. – Дата доступа: 21.05.2023.

4. DES Modes of Operation / National Bureau of Standards ; ed.: Computer Systems Laboratory. – Gaithersburg : National Bureau of Standards, 1980. – 26 p.

5. Dworkin, M. Recommendation for Block Cipher Modes of Operation: Methods and Techniques / M. Dworkin – SP 800-38A. – Gaithersburg : National Institute of Standards and Technology, 2001 – 59 p.

6. EAX. A two-pass authenticated encryption mode: papers from Fast Software Encryption, Delhi, 5–7 Feb. 2004 / D. Wagner, M. Bellare, P. Rogaway ; ed.: R. Bimal, W. Meier. – Delhi : International Association for Cryptologic Research, 2004. – 31 p.

7. PKCS #1: RSA Cryptography Specifications Version 2.2 / K. Moriarty [et al.]. – Internet Engineering Task Force, 2016. – 78 p.

8. Purrweb [Электронный ресурс]. – Режим доступа: https://www.purrweb.com/ru/uslugi/veb-razrabotka/. – Дата доступа: 21.05.2023.

9. Recommendation for Key Management for Cryptographic Applications: in 3 vol. / ed.: Shirley M. Radack. – Gaithersburg : National Institute of Standards and Technology, 2020 – Vol. 1 – 158 p.

10. Towards Securing Client-Server Connections against : papers from 10th International Symposium on Electronics and Telecommunications, Timisoara, 15–16 Nov. 2012 / ed.: Institute of Electrical and Electronics Engineers. – Timisoara : Institute of Electrical and Electronics Engineers, 2012. – 4 p.

11. Паттерн «Стратегия» [Электронный ресурс]. – Режим доступа: https://refactoring.guru/ru/design-patterns/strategy. – Дата доступа: 21.05.2023.

12. Сколько стоит разработка мессенджера? [Электронный ресурс]. – Режим доступа: https://merehead.com/ru/blog/messenger-app-development/. – Дата доступа: 21.05.2023.

13. Сколько стоит разработка приложения [Электронный ресурс]. – Режим доступа: https://www.purrweb.com/ru/blog/skolko-stoit-razrabotka-prilozheniya/. – Дата доступа: 21.05.2023.

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Интеллектуальные информационные технологии»

ШИФРОВАНИЕ ДАННЫХ, ПЕРЕДАВАЕМЫХ ПО КОМПЬЮТЕРНЫМ СЕТЯМ, НА ПРИМЕРЕ ДАННЫХ ПРИЛОЖЕНИЯ JUNIORGRAM

ТЕКСТ ПРОГРАММЫ

(на оптическом носителе CD-R)

ДП.АС55.190232-05 12 00

Листов 4
Объем 7,78 МБайт

Руководитель Ю. И. Давидюк

Выполнил В. И. Лепешев

Консультант:

по ЕСПД Ю. И. Давидюк

2023

Аннотация

В дипломной работе была реализована криптографическая система для приложения Juniorgram. Данная система состоит из следующих модулей:

- Модуль симметричного шифрования.
- Модуль асимметричного шифрования.
- Модуль генерации ключа шифрования.
- Модуль подтверждения ключа.
- Модуль верификации соединения.
- Хранилище для симметричных ключей.
- Генератор псевдослучайных чисел.

Все эти модули были внедрены в приложение и протестированы и каждый снабжен комментариями в коде.

Содержание

1. Juniorgram\Base\Crypto\Crypto.Static – содержит модули криптографической системы.
2. Juniorgram\Base\Crypto\Crypto.Test – содержит модульные тесты, разработанные для криптографической системы.
3. Juniorgram\Client\Client.Core\Client.Core.Static –клиентская часть приложения.
4. Juniorgram\Server\Server.Core\Server.Core.Static – серверная часть приложения.
5. Juniorgram\Network\Public\Include\Network – содержит обработчики сообщений, для шифрования в том числе.
6. Juniorgram\Docs\Cryptography.md – документация на разработанную криптографическую систему.