МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ**

**«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ ТЕХНОЛОГИЧЕСКИЙ УНИВЕРСИТЕТ «МИСИС»**

ИНСТИТУТ КОМПЬЮТЕРНЫХ НАУК

*КАФЕДРА ИНЖЕНЕРНОЙ КИБЕРНЕТИКИ*

*НАПРАВЛЕНИЕ 01.03.04 ПРИКЛАДНАЯ МАТЕМАТИКА*

**ВЫПУСКНАЯ**

**КВАЛИФИКАЦИОННАЯ**

**РАБОТА БАКАЛАВРА**

1. **на тему:** **Криптосистема proxy re-encryption на базе блокчейн для безопасной передачи конфиденциальных данных в финансовом секторе.**

*Студент\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_М.А. Корнилов*

*Руководитель* *работы*\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_*И.А. Тарханов*

*Нормоконтроль проведен\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_М.В. Хмарная*

*Проверка на заимствования проведена \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_М.В. Хмарная*

***Работа рассмотрена кафедрой и допущена к защите в ГЭК***

*Заведующий кафедрой* *\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ А.Р. Ефимов*

*Директор института\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ С.В. Солодов*

Москва июнь 2024

**СОДЕРЖАНИЕ**

[ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ 4](#_Toc164615092)

[ВВЕДЕНИЕ 5](#_Toc164615093)

[1 АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ 6](#_Toc164615094)

[1.1 Существующие решения для безопасной передачи конфиденциальных данных 6](#_Toc164615095)

[1.1.1 Основные принципы криптографии 6](#_Toc164615096)

[1.1.2 Существующие методы защиты данных 7](#_Toc164615097)

[1.1.3 Анализ преимуществ и недостатков существующих методов 9](#_Toc164615098)

[1.2 Описание схемы работы Proxy re-encryption 11](#_Toc164615099)

[1.2.1 Принцип работы схемы 11](#_Toc164615100)

[1.2.2 Существующие PRE схемы 12](#_Toc164615101)

[1.2.3 Анализ безопасности схемы 12](#_Toc164615102)

[1.2.4 Эффективность криптографической защиты 14](#_Toc164615103)

[2 СПЕЦИАЛЬНАЯ ЧАСТЬ 17](#_Toc164615104)

[2.1 Содержательная постановка задачи 17](#_Toc164615105)

[2.2 Математическая постановка задачи 18](#_Toc164615106)

[2.3 Алгоритм перешифрования 23](#_Toc164615107)

[3 ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ТЕСТИРОВАНИЕ 26](#_Toc164615108)

[3.1 Описание подхода к разработке и тестированию программного обеспечения 26](#_Toc164615109)

[3.2 Используемые программные средства 27](#_Toc164615110)

[3.3 Описание функциональности разработанного ПО 28](#_Toc164615111)

[3.4 Тестирование и оценка эффективности 30](#_Toc164615112)

[3.4.1 Описание методологии тестирования схемы Proxy re-encryption 30](#_Toc164615113)

[3.5 Оценка криптостойкости системы 33](#_Toc164615114)

[3.6 Анализ полученных результатов 36](#_Toc164615115)

[ЗАКЛЮЧЕНИЕ 37](#_Toc164615116)

[СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ 39](#_Toc164615117)

[ПРИЛОЖЕНИЕ A 41](#_Toc164615118)

# ПЕРЕЧЕНЬ СОКРАЩЕНИЙ И ОБОЗНАЧЕНИЙ

В настоящей работе применяют следующие сокращения и

обозначения:

CA – certificate authority

CB-PRE – certificate based proxy re-encryption

CCA – chosen-ciphertext attack

CPA – chosen-plaintext attack

EC – elliptic-curve

ECQV – elliptic curve Qu Vanstone

IOT – internet of things

PRE – proxy re-encryption

SCA – side-channel attack

SSH – secure shell

TLS – transport layer security

# ВВЕДЕНИЕ

В современном мире передача конфиденциальной информации является критически важной задачей из-за возрастающего из года в год числа кибератак. Это особенно важно в финансовом секторе, где необходимо соблюдать строгие нормативные требования к защите информации без риска компроментации данных в процессе их передачи. В связи с этим существует потребность в разработке надежных и безопасных методов передачи данных. Схема proxy re-encryption (PRE) является одним из таких методов, который позволяет передавать зашифрованные данные от отправителя к получателю через надежный посредник, называемый прокси-сервером.

Данная курсовая работа посвящена разработке алгоритмического и программного обеспечения для создания криптосистемы proxy re-encryption на базе блокчейна для безопасной передачи конфиденциальных данных в финансовом секторе.

Работа будет состоять из следующих разделов: введение в схему Proxy Re-Encryption, описание CB-PRE схемы и всех ее компонентов, анализ применимости для передачи конфиденциальных данных, реализация схемы на основе выбранного языка программирования, тестирование разработанной системы и обсуждение результатов.

Целью данной курсовой работы является изучение схемы Proxy Re-Encryption в сфере финансовых технологий и разработка полнофункциональной системы CB-PRE для передачи конфиденциальных данных с использованием выбранного языка программирования. Ожидается, что разработанная система будет эффективно работать на практике и обеспечивать высокий уровень безопасности для передаваемых данных.

# АНАЛИТИЧЕСКИЙ ОБЗОР ЛИТЕРАТУРЫ

## Существующие решения для безопасной передачи конфиденциальных данных

### Основные принципы криптографии

Криптография – это наука о защите информации, которая изучает методы обеспечения конфиденциальности, целостности и аутентичности данных. Ее основная задача - обеспечить безопасную передачу информации между двумя или более участниками без риска ее прослушивания или изменения.

Основными принципами криптографии являются:

* Симметричное и асимметричное шифрование – это методы шифрования данных, которые обеспечивают защиту информации при ее передаче по сети. В симметричном шифровании для шифрования и расшифрования используется один и тот же ключ. В асимметричном шифровании используются два ключа: открытый и закрытый[1].
* Хэширование – это процесс преобразования данных произвольной длины в уникальную фиксированную строку определенной длины. Хэширование используется для проверки целостности данных [1].
* Цифровые подписи – это методы обеспечения аутентичности данных. Цифровая подпись позволяет установить, что сообщение было создано определенным отправителем, и что сообщение не было изменено в процессе передачи.
* Протоколы аутентификации – это методы обеспечения подлинности пользователя. Протоколы аутентификации позволяют установить, что пользователь является тем, за кого себя выдает.
* Протоколы обмена ключами – это методы обеспечения конфиденциальности данных. Протоколы обмена ключами позволяют двум участникам обменяться секретным ключом для дальнейшего использования в процессе шифрования.

В контексте разработки схемы Proxy re-encryption для безопасной передачи конфиденциальных данных основными принципами криптографии, которые будут использоваться, являются симметричное и асимметричное шифрование, протоколы обмена ключами и цифровые подписи.

### Существующие методы защиты данных

Существует множество методов и технологий защиты данных, начиная от простых методов шифрования и заканчивая сложными системами управления доступом и многофакторной аутентификации. В данной работе мы рассмотрим несколько основных методов, которые широко используются для защиты конфиденциальных данных.

1. Шифрование данных: один из наиболее распространенных методов защиты данных, который заключается в преобразовании информации таким образом, чтобы только тот, у кого есть ключ, мог ее расшифровать. Существует множество алгоритмов шифрования, таких как AES, RSA, DES и другие [1].
2. Сетевые протоколы безопасности: используются для защиты данных, передаваемых по сети, от несанкционированного доступа. Примеры таких протоколов включают SSL/TLS, SSH, IPSec и другие [2].

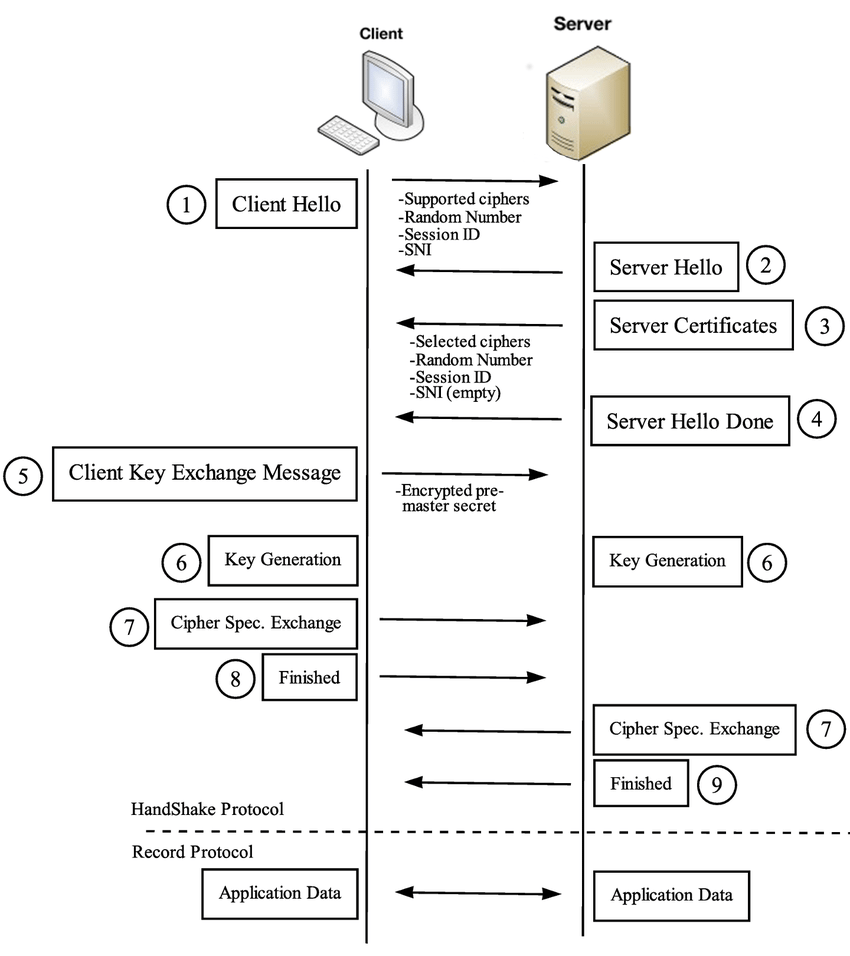


Рисунок 1 – Схема tls соединения

1. Управление доступом: технологии, позволяющие определять, кто и как может получить доступ к конфиденциальным данным. Обычно включают авторизацию, аутентификацию и управление правами доступа.
2. Программные средства защиты информации: программные продукты, предназначенные для защиты данных, обнаружения и предотвращения вторжений, мониторинга безопасности и других задач. Примеры таких продуктов включают антивирусы, фаерволы, IDS/IPS и другие.
3. Схемы Proxy Re-Encryption: специальный тип криптографических протоколов, который позволяет безопасно передавать зашифрованные данные между несколькими участниками, используя промежуточный сервер для перешифрования данных.
4. Токенизация данных: Токенизация представляет собой процесс замены конфиденциальных данных уникальными токенами, которые не раскрывают исходные данные. Токены используются для обработки данных, при этом исходные данные сохраняются в безопасности. Этот метод позволяет уменьшить риски утечки конфиденциальной информации и повысить безопасность хранения данных.
5. Криптографические хеш-функции: Хеш-функции преобразуют данные в уникальную последовательность фиксированного размера. Они широко используются для обеспечения целостности данных и проверки их целостности. Криптографические хеш-функции обеспечивают невозможность обратного преобразования хеша в исходные данные, что делает их надежными для защиты данных.
6. Средства мониторинга и обнаружения инцидентов безопасности: Эти инструменты предназначены для постоянного мониторинга и обнаружения потенциальных инцидентов безопасности. Они позволяют своевременно выявлять аномалии, несанкционированный доступ или подозрительное поведение в системе, что помогает предотвратить и реагировать на угрозы безопасности.
7. Резервное копирование и восстановление данных: Регулярное создание резервных копий данных и разработка стратегии восстановления в случае потери или повреждения данных являются важными аспектами защиты данных. Этот метод обеспечивает возможность быстрого восстановления данных и минимизации потерь в случае сбоев или атак.
8. Обучение сотрудников в области безопасности: Обучение сотрудников в области безопасности данных является важным компонентом общей стратегии защиты данных. Сотрудники должны быть осведомлены о лучших практиках безопасности, правилах использования паролей, распознавании фишинговых атак и других видов угроз безопасности.
9. Физическая безопасность: Обеспечение физической безопасности серверных помещений, хранилищ данных и других физических ресурсов является важным аспектом защиты данных. Меры физической безопасности могут включать контроль доступа, видеонаблюдение, а также защиту от пожара и повреждений.

### Анализ преимуществ и недостатков существующих методов

Для выбора наиболее подходящего метода защиты данных для конкретного случая необходимо проанализировать преимущества и недостатки каждого метода.

Симметричное шифрование, например, является быстрым и эффективным методом, однако у него есть недостатки. К примеру, в этом методе используется общий секретный ключ, который должен быть передан между отправителем и получателем, что может привести к его компрометации. Кроме того, управление доступом к данным в таком методе сложно реализовать.

Асимметричное шифрование, с другой стороны, имеет преимущества в том, что нет необходимости передавать общий секретный ключ, так как используются два различных ключа - публичный и приватный. Однако он требует большего времени и вычислительной мощности для работы, а также имеет проблемы с управлением доступом к данным.

Прокси-шифрование является методом, который позволяет обеспечить безопасную передачу данных между отправителем и получателем, не требуя от них знания ключа шифрования. Кроме того, он обеспечивает гибкое управление доступом к данным. Однако, такой метод также имеет свои недостатки, такие как дополнительная производительность, которая необходима для выполнения решифрования.

Таким образом, каждый метод имеет свои преимущества и недостатки, и выбор конкретного метода зависит от требований к безопасности, производительности и управлению доступом к данным в конкретной ситуации.

## Описание схемы работы Proxy re-encryption

### Принцип работы схемы

Схема Proxy re-encryption (прокси-перешифрование) является эффективным методом для безопасной передачи конфиденциальных данных между отправителем и получателем с использованием промежуточного прокси-сервера. Она обеспечивает конфиденциальность и защиту данных, позволяя изменять ключ шифрования без необходимости расшифровывать и повторно шифровать данные.

Принцип работы схемы Proxy re-encryption состоит из следующих этапов:

Настройка (Setup): В начале работы прокси-сервера проводится этап настройки, в ходе которого генерируются необходимые параметры, такие как приватные и публичные ключи, а также другие конфигурационные данные. Этот этап выполняется один раз при запуске прокси-сервера [3].

Сертификация пользователей (CertifiedUserKeyGen): Перед использованием схемы, каждый пользователь должен пройти процесс сертификации, в результате которого ему выдаются сертификаты. Эти сертификаты подтверждают идентификацию и аутентичность пользователей, а также содержат необходимую информацию для работы схемы [3].

Шифрование (Encrypt): Отправитель использует публичный ключ получателя для шифрования данных перед отправкой. При этом прокси-сервер не имеет доступа к исходным данным, так как они зашифрованы публичным ключом получателя [3].

Перешифрование (ReEncrypt): Прокси-сервер выполняет операцию перешифрования данных с использованием своего приватного ключа и ключей получателя. В результате этой операции данные становятся доступными для расшифровки только для получателя, при этом прокси-сервер не расшифровывает исходные данные [3].

Расшифровка (Decrypt): Получатель, имея свой приватный ключ, расшифровывает полученные данные. При этом получателю не требуется наличие приватного ключа прокси-сервера для успешной расшифровки [3].

Схема Proxy re-encryption позволяет достичь высокого уровня безопасности при передаче конфиденциальных данных, так как она обеспечивает конфиденциальность и контроль доступа, при этом минимизируя потребность в передаче приватных ключей между отправителем и получателем.

### Существующие PRE схемы

1. Односторонняя: позволяет делегировать доступ к шифрованным данным от одного пользователя к другому, но не обратно [4].
2. Двусторонняя: поддерживает возможность делегирования доступа между двумя пользователями в обе стороны.
3. Многоступенчатая: подразумевает возможность передачи доступа через цепочку пользователей.
4. Симметричная vs. Асимметричная: В симметричном PRE используется один ключ для шифрования и расшифровки, в асимметричном — разные ключи для каждого пользователя [4].
5. Условная: позволяет решифрование только если выполнены определённые условия.

В данной работе в целях повышения безопасности, гибкости управления доступа к данным и удобства передачи данных между участниками сети была выбрана двухсторонняя ассиметричная PRE схема.

### Анализ безопасности схемы

Анализ безопасности схемы Proxy re-encryption включает в себя оценку ее сильных сторон и потенциальных уязвимостей. Ниже представлен анализ безопасности схемы:

1. Конфиденциальность: Схема Proxy re-encryption обеспечивает высокий уровень конфиденциальности данных. Шифрование данных перед их передачей от отправителя к прокси-серверу и от прокси-сервера к получателю гарантирует, что только авторизованные пользователи смогут получить доступ к расшифрованным данным. Криптографические протоколы, такие как ReKeyGen и ReEncrypt, обеспечивают безопасное преобразование и перешифрование данных, минимизируя риски утечки информации.
2. Целостность: Схема Proxy re-encryption обеспечивает целостность данных, поскольку шифрование и расшифрование данных выполняются с использованием ключей и алгоритмов, которые гарантируют, что данные не изменяются неправомерно или без уведомления сторон.
3. Аутентификация: Схема Proxy re-encryption предоставляет возможности для аутентификации отправителя и получателя данных. Сертификационный процесс, описанный в алгоритме CertifiedUserKeyGen, позволяет установить доверие к публичным ключам пользователей, что гарантирует, что данные будут переданы только тем, кто имеет соответствующие авторизационные данные.
4. Устойчивость к атакам: Схема Proxy re-encryption разработана с учетом различных типов атак, таких как перехват, подмена и подслушивание данных. Использование криптографических алгоритмов, включая эллиптические кривые[5], повышает уровень безопасности и усложняет возможность проведения успешных атак.
5. Надежность ключей: Безопасность схемы Proxy re-encryption зависит от надежности и безопасности используемых ключей. Важно обеспечить защиту приватных ключей пользователей и прокси-серверов, чтобы предотвратить их компрометацию. Это может включать использование сильных алгоритмов генерации ключей, хранение ключей в безопасной среде и управление доступом к ключам с помощью соответствующих политик и процедур.
6. Мониторинг и обнаружение атак: Важным аспектом обеспечения безопасности схемы Proxy re-encryption является мониторинг и обнаружение атак. Регулярный мониторинг сетевого трафика, анализ логов и использование систем обнаружения вторжений (IDS) и систем защиты от вторжений (IPS) помогут обнаружить аномальную активность и потенциальные попытки нарушения безопасности системы.
7. Регулярные проверки на уязвимости: Для поддержания высокого уровня безопасности необходимо проводить регулярные проверки на уязвимости схемы Proxy re-encryption. Это может включать проведение пенетрационного тестирования, аудита кода, анализа уязвимостей и других методов оценки безопасности. Выявление и устранение потенциальных уязвимостей поможет повысить стойкость системы к возможным атакам [7].

Однако, как и любая криптографическая схема, схема Proxy re-encryption имеет свои потенциальные уязвимости. Например, уязвимости могут возникнуть в результате компрометации приватных ключей пользователей или прокси-серверов, а также из-за ошибок в реализации алгоритмов. Поэтому важно применять хорошие практики безопасности, такие как защита приватных ключей и регулярные проверки на уязвимости системы.

Таким образом, схема Proxy re-encryption обладает высоким уровнем безопасности, но требует правильной настройки и применения рекомендаций по обеспечению безопасности, чтобы минимизировать потенциальные риски и уязвимости.

### Эффективность криптографической защиты

Оценка эффективности защиты информации с помощью криптографии включает в себя выявление всех возможных нападений, которым может подвергнуться криптографическая система в конкретных условиях её использования, и установление её способности сопротивляться таким атакам. Под прочностью системы мы будем подразумевать её криптографическую надежность, то есть её способность выдерживать нападения криптоаналитиков.

Разработка методов определения криптографической надежности представляет собой сложную задачу, требующую комплекс инструментальных средств, которые позволят оценивать действия злоумышленника в попытках взлома системы с использованием разнообразных техник криптоанализа. Детальное описание программного комплекса, предназначенного для анализа устойчивости асимметричных криптосистем с применением математических методов, выходит за рамки данной статьи, но его можно найти в других источниках.

Для точного определения эффективности криптосистемы целесообразно проверять её на устойчивость не ко всем без исключения атакам, а лишь к тем, которые наиболее вероятны и представляют собой наибольшую опасность. Набор таких атак определяется в зависимости от типа криптосистемы и специфики её применения.

Процесс оценки эффективности криптографической защиты можно представить в виде схемы в соответствии с Рисунком 2 [6].

Этап 1: Какая криптосистема является объектом атаки?

Этап 2: Кто будет атаковать эту криптосистему?

Этап 3: Какие методы криптоанализа с наибольшей вероятностью будут использованы при осуществлении попыток взлома криптосистемы?

Этап 4: Способна ли криптосистема противостоять таким атакам?

Этап 5: Является ли использование исследуемой криптосистемы экономически выгодном в данном контексте

Этапы 1-3 представляют моделирование угроз для информационных ресурсов, которые защищает исследуемая криптосистема.

Этап 4 – это анализ устойчивости системы в атаках определенного типа, которые были определены на предыдущих этапах. Цель заключается в оценке риска нарушения безопасности.

Этап 5 – использование различных подходов оценки экономической эффективности использования выбранной криптосистемы.

A diagram of a diagram

Description automatically generated

Рисунок 2 – Процесс оценки эффективности системы

# СПЕЦИАЛЬНАЯ ЧАСТЬ

## Содержательная постановка задачи

Цель работы: Разработка и анализ оптимизированной схемы прокси-решифрования с использованием смарт-контрактов, обеспечивающих повышение эффективности и безопасности доступа к зашифрованным данным.

Предметная область: Информационная безопасность, криптография.

В качестве входных данных выступают:

1. Клиент, сервер(получатель), проски сервер.
2. Конфиденциальная информация, которую необходимо передать по сети.

В качестве результата должна быть разработана и проверена схема прокси-решифрования, интегрированная с смарт-контрактами, обеспечивающая безопасную передачу и доступ к конфиденциальным данным.

Для достижения поставленной цели необходимо выполнить следующие действия:

1. создание алгоритма прокси решифрования на python.
2. разработка смарт-контракта на solidity.
3. объединение прокси-решифрования и смарт-контракта в единную систему.
4. генерация публичных и приватных ключей участников сети на основе эллиптических кривых и их регистрация на блокчейне.
5. проверка работоспособности системы.
6. тестирование системы на уязвимости.

## Математическая постановка задачи

Чтобы определить комплекс угроз для криптосистемы, разработаем математическую модель, которая оценивает безопасность защиты информации. Модель базируется на таких исходных данных:

* возможно, что один хакер будет использовать множество разнообразных методов атаки, также как один и тот же тип атаки может быть осуществлён различными хакерами.
* разнообразные атаки могут быть направлены против одной криптосистемы, и одинаковые атаки могут нарушить безопасность разных криптосистем.
* атакующий, скорее всего, предпочтёт тот метод атаки, который при фиксированном уровне ресурсов даст ему наилучший итог или который окажется наиболее экономичным среди различных опций, ведущих к схожему результату.

Пусть – множество параметрических моделей атак, где – множество значений j-го параметра модели атаки. Тип атаки определяется в соответствии с критериями классификации [8]:

- по доступу к открытому и зашифрованному тексту;

- по контролю над процессом шифрования;

- по объему необходимых ресурсов;

- по степени применимости к различным шифрам;

- по используемым средствам;

- по последствиям атаки;

- по возможности распарллеливания.

Каждая модель преставляет собой вектор , где

Мощность множества моделей атак:

Пусть – множество параметрических моделей злоумышленников, где ) – множество значений j-го параметра модели злоумышленника. Тип злоумышленника определяется в соответствии с критериями классификации [8]:

- по технической оснащенности;

- по конечной цели;

- по доступу к шифрующим средствам;

- по уровню подготовки;

- по первичной информации о средстве шифрования;

- по возможности кооперации.

Каждая модель преставляет собой вектор , где

Мощность множества моделей злоумышленников:

Пусть – множество параметрических моделей криптосистем, где ) – множество значений j-го параметра модели криптосистемы. Тип криптосистемы определяется в соответствии с критериями классификации [8]:

- по доступности информации о криптоалгоритме;

- по числу ключей;

- по стойкости криптоалгоритма;

- по используемым средствам шифрования;

- по наличию сертификата.

Каждая модель представляет собой вектор , где

Мощность множества моделей криптосистем:

Риск от атаки вычисляется на основе двух факторов – вероятности происшествия и тяжести возможных последствий:

Функцию для оценки уровня риска обозначим через: . Она связана с атакой , когда она применена злоумышленником  для взлома криптосистемы .

Пусть – функция влияния, которая оценивает степень ущерба от применения атаки к криптосистеме .

Пусть – вероятность того, что злоумышленник предпримет атаку .

Таким образом, функция риска будет выглядеть так:

Для определения функции рассмотрим семейство функций:

, , где , - множество неотрицательных действительных чисел. Функция задает уровень взаимного влияния параметра криптосистемы и параметра атаки :

* , если атака c параметром  не применима к криптосистеме со значением параметра ;
* , если значение параметра криптосистемы снижает вероятность успешного применения атаки со значением параметра ;
* , если значение параметра криптосистемы не влияет на применимость атаки с параметром ;
* , если значение параметра криптосистемы указывает на то, что атака с параметром применима для взлома;

Уровень взаимного влияния параметром криптосистемы и атаки определяется на основе экспертных оценок.

Пусть – нормированная функция:

Значит ущерб от применения атаки к криптосистеме вычисляется по следующей формуле:

Где атака и криптосистема заданы параметрами: и (.

Для определения функции рассмотрим семейство функций , , . Функция задает уровень взаимного влияния параметра злоумышленника и параметра атаки :

* *,* если злоумышленник со значением параметра  гарантированно не будет использовать атаку со значением параметра ;
* *,* если значение параметра злоумышленника  снижает вероятность использования атаки со значением параметра ;
* *,* если значение параметра злоумышленника не влияет на вероятность использования атаки со значением параметра ;
* *,* если злоумышленник со значением параметра с большой вероятностью будет использовать атаку со значением параметра .

Уровень взаимного влияния параметров злоумышленника и атаки также определяется экспертами.

Пусть – нормированная функция:

Тогда вероятность того, что злоумышленник предпримет атаку :

Где атака и злоумышленник заданы параметрами и соответственно.

Таким образом общая формула для определения уровня риска имеет вид:

Криптосистема подвержена атаке в условиях, когда ей угрожает злоумышленник , если , где  – заданное пороговое значение. Допустимый уровень риска настраиваемый параметр, который задается с учетом критериев:

* Критичность защищаемых данных.
* Ресурсы, доступные специалисту для осуществления проверки защищенности системы.

В общем контексте криптосистема может включать в себя несколько подсистем, таких как генератор ключей и симметричный шифратор, каждая из которых подвержена своему собственному спектру возможных атак. Криптосистема может быть также объектом нападения различных злоумышленников. Совокупность всех возможных атак на криптосистему, состоящую из подмножества атак из универсального множества атак  (где ), в условиях, когда злоумышленникам доступно множество  (где ), можно определить формулой , где при заданном уровне риска. Для оценки надежности криптосистемы необходимо использовать инструментальные средства, чтобы оценить её способность сопротивляться атакам, входящим в множество .

В описанной математической модели сделаны следующие допущения:

* Не учитывается зависимость параметров атаки от сочетания параметров криптосистемы
* Не учитывается возможность совместных действий со стороны взломщиков различных типов

Учёт этих изменений в модели сделает её более сложной. Остаётся открытым вопрос, сказываются ли эти изменения на точности предсказаний модели по поводу угроз безопасности, этот аспект требует дополнительного анализа.

## Алгоритм перешифрования

* SetUp(l): При заданном определенном параметре безопасности l будут выполнены следующие шаги для получения общедоступных параметров params и главного секретного ключа msk.

1. Сначала центр сертификации выбирает l-разрядное простое число q. Далее генерируется EC порядка q и определяется соответствующая точка генератора P. Обозначим через G группу точек EC.
2. Выбирается случайное значение и вычисляется .
3. Определяются четыре различные хэш-функции:
4. Общедоступные параметры params теперь: , секретный ключ .

* CertifiedUserKeyGen(): Этот алгоритм основан на механизме сертификации эллиптической кривой Кванстоуна (ECQV) и состоит из следующих трех этапов:

1. генерирует случайное значение и вычисляется , далее кортеж отправляется в CA.
2. По прибытию CA проверяется идентификатор . Далее он выбирает случайное значение и вычисляет и сертификат получен. Вспомогательная информация для вычисления приватного ключа для задействованного клиента вычисляется по формуле:

Кортеж отправляется обратно.

1. Клиент сначала получает свой закрытый ключ . Его публичный ключ эквивалентен . Если , то он принимает пару ключей .

* Encrypt(: Метадата генерируется для сообщения M, .

Дальше производятся следующие вычисления:

На выходе алгоритма получаем

* ReKey: получено из C. Публичный ключ вычисляется как . Это приводит к вычислению ключа решифрования как:

В результате получен ключ решифрования .

* ReEncrypt: Повторное шифрование изменяет зашифрованные текст от в зашифрованные текст :

Если подставить , то получаем, что:

В результате работы функции получаем:

* Decrypt1: Для расшифровки оригинального сообщения и проверки его подлинности требуются следующие вычисления:

Проверка:

* Decrypt2На этом этапе получатель извлекает сообщение из с помощью следующих операций:

Проверка:

# ПРОГРАММНОЕ ОБЕСПЕЧЕНИЕ И ТЕСТИРОВАНИЕ

## Описание подхода к разработке и тестированию программного обеспечения

1. Выбор инструментов разработки:

* Для разработки и деплоя смарт контрактов, которые будут использоваться в схеме Proxy re-encryption, будет использована онлайн-среда Remix Ethereum. Этот инструмент обладает мощными возможностями разработки, отладки и развертывания смарт контрактов на блокчейне Ethereum.
* Для создания локальной сети блокчейна Ethereum, в которой будут тестироваться и разрабатываться смарт контракты, будет использоваться инструмент Ethereum Ganache, который предоставляет среду для развертывания локального блокчейна Ethereum, что позволяет тестировать и отлаживать смарт контракты в изолированной среде без необходимости подключения к основной блокчейн-сети.

1. Разработка смарт контрактов:

* Cреда Remix Ethereum хорошо подходит для создания и разработки смарт контракты на языке Solidity. В ней необходимо реализовать алгоритмы и протоколы, описанные в предыдущих разделах работы, внутри смарт контрактов.
* Remix Ethereum предоставляет мощные инструменты для проверки синтаксиса, отладки и тестирования смарт контрактов.

1. Деплой смарт контрактов:

* После разработки и отладки смарт контрактов в среде Remix Ethereum, мы можем использовать Ganache для развертывания смарт контрактов на локальной сети блокчейна Ethereum.
* Ganache предоставляет нам локальный блокчейн с эмулированными аккаунтами и средствами для развертывания контрактов. Мы можем использовать его для проверки работоспособности наших смарт контрактов в изолированной среде.

1. Тестирование и отладка:

* После деплоя смарт контрактов на локальной сети блокчейна с помощью Ganache, необходимо провести тестирование и отладку наших контрактов.
* Remix Ethereum для отправки транзакций и вызова функций наших смарт контрактов, а также для проверки возвращаемых результатов.

Таким образом, описанный подход позволит разработать и тестировать смарт контракты для схемы Proxy re-encryption с помощью Remix Ethereum и локального блокчейна Ethereum с использованием Ganache.

## Используемые программные средства

Для реализации использованы языки программирования Python 3 и Solidity.

Среда программирования – PyCharm IDE, Remix IDE. Для поднятия локального блокчейна используется Ganache.

Были использованы следующие библиотеки, которые не входят в стандартные библиотеки python, и фреймворки:

* Charm-crypto.
* Web3.
* Cryptography.

Charm-crypto – это фреймворк для быстрого создания прототипов передовых криптосистем. Основанный на языке Python, он был разработан с нуля, чтобы минимизировать время разработки и сложность кода, одновременно обеспечивая повторное использование компонентов.

В Charm используется гибридная конструкция: ресурсоемкие математические операции реализованы в собственных модулях C, а сами криптосистемы написаны на читаемом языке высокого уровня. Charm дополнительно предоставляет ряд новых компонентов, облегчающих быструю разработку новых схем и протоколов.

Web3 – это библиотека Python для взаимодействия с Ethereum.

Его обычно можно встретить в децентрализованных приложениях (dapps), чтобы помочь с отправкой транзакций, взаимодействием со смарт-контрактами, чтением данных блоков и множеством других вариантов использования.

Исходный API был основан на Javascript API Web3.js, но с тех пор был усовершенствован в соответствии с потребностями и удобством разработчиков Python.

Cryptography – это библиотека, которая предоставляет разработчикам Python криптографические рецепты и примитивы. Цель состоит в том, чтобы это была «стандартная криптографическая библиотека».

Включает в себя как рецепты высокого уровня, так и интерфейсы низкого уровня для общих криптографических алгоритмов, таких как симметричные шифры, дайджесты сообщений и функции получения ключей.

Для построения полноценной схеме было поднято 4 виртуальные машины на платформе Yandex Cloud, каждая со своим белым ip адресом.

## Описание функциональности разработанного ПО

Проект состоит из 7 файлов расширения (.py):

1. ca.py: центр сертификации, который генерирует секретный ключ, ключи участников сети и публичные параметры.
2. sender.py: участник сети, которые отправляет данные.
3. receiver.py: получатель зашифрованных данных.
4. proxy.py: прокси сервер, которые выполняет функции решифрования данных и их передачи получателю.
5. smart\_contract.py: описание ABI смарт-контракта, поддерживаемых им методов, функции для взаимодействия с ним.
6. constants.py: общие константы (адрес смарт-контракта, эндпоинты прокси и так далее).
7. util.py: вспомогательные функции для работы с данными.

Так же в проекте есть 1 файл с расширение (.sol) – исходных код смарт-контракта.

## Тестирование и оценка эффективности

### Описание методологии тестирования схемы proxy re-encryption

Юнит-тестирование:

* Юнит-тестирование смарт контрактов проводится с использованием фреймворка Truffle. Мы создаем тестовые сценарии для проверки отдельных функций и компонентов смарт контрактов. Целью этого тестирования является уверенность в корректной работе отдельных частей смарт контрактов, таких как функции шифрования, расшифровки, генерации ключей, а также обработки данных [9].

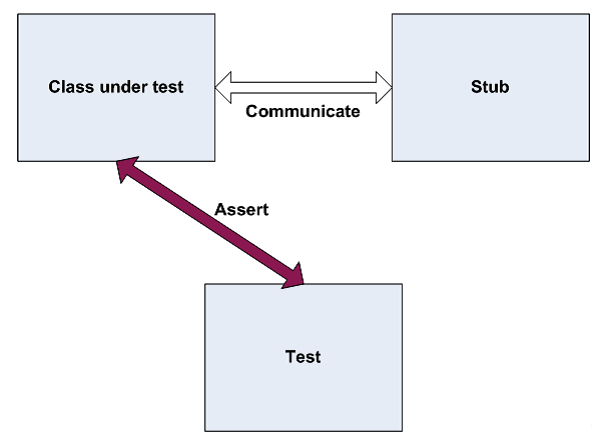


Рисунок 3 – Схема юнит тестирования

Интеграционное тестирование:

* Интеграционное тестирование направлено на проверку взаимодействия между различными компонентами системы и правильности работы смарт контрактов в реальных сценариях использования. Мы создаем тестовые сценарии, которые охватывают отправку и получение зашифрованных данных, выполнение операций Proxy re-encryption и проверку правильности результатов. Целью тестирования является убедиться, что смарт контракты взаимодействуют корректно и обеспечивают безопасную передачу конфиденциальных данных.

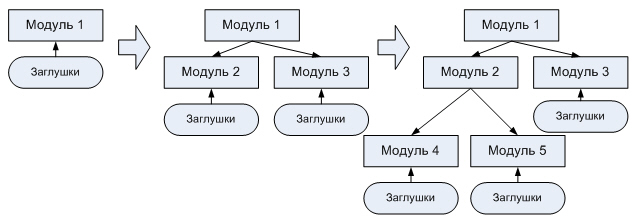


Рисунок 4 – Схема интеграционного тестирования

Функциональное тестирование:

* Функциональное тестирование направлено на проверку функциональности схемы Proxy re-encryption в целом. Мы разрабатываем тестовые сценарии, которые охватывают основные функции и возможности системы. Например, мы проверяем правильность шифрования и расшифровки данных с использованием различных ключей, проверяем правильность перешифрования данных и работу алгоритмов Proxy re-encryption. Целью тестирования является убедиться, что система работает согласно спецификации и выполняет свои функции корректно.

Нагрузочное тестирование:

* Нагрузочное тестирование проводится для оценки производительности и масштабируемости системы в условиях высокой нагрузки. Мы используем инструменты, такие как Truffle или другие фреймворки для нагрузочного тестирования блокчейн-систем. Целью этого тестирования является проверка работоспособности системы при высокой нагрузке и определение ее производительности [8].

Помимо перечисленных видов тестирования, важным этапом в разработке и обеспечении безопасности схемы Proxy re-encryption является аудит смарт контрактов. Аудит выполняется независимыми специалистами с целью выявления потенциальных уязвимостей и ошибок в коде смарт контрактов. Аудит помогает обеспечить безопасность системы и предотвратить возможные атаки или нарушения конфиденциальности данных.

Все эти виды тестирования и аудит смарт контрактов играют важную роль в обеспечении качества и надежности программного обеспечения для схемы Proxy re-encryption. Они помогают выявить потенциальные проблемы и ошибки, а также обеспечить корректную работу системы в различных сценариях использования.

ДОП ИНФОРМАЦИЯ ПО ТЕСТИРОВАНИЮ

## Оценка криптостойкости системы

В соответствии с разработанной математической моделью (п. 2.2) и критериями классификации [8]: проведем идентификацию криптосистемы:

* по доступности информации – криптосистема ограниченного использования;
* по числу ключей – многоключевая;
* по стойкости криптоалгоритма – доказуемо стойкая;
* по используемым средствам шифрования – программная;
* по наличию сертификата – сертифицированная;

Следующим этапом в процессе оценки эффективности (рисунок 2) является определение потенциальных взломщиков, на основании основных угроз из статьи [10] смоделированы два основных профиля атакующих:

Взломщик 1:

* по технической оснащенности – сеть ЭВМ;
* по конечной цели – полный взлом алгоритма;
* по доступу к шифрующим средствам – “внешний” нарушитель
* по уровню подготовки – программирование;
* по первичной информации о средстве шифрования – криптограф;
* по возможности кооперации – коллектив;

Взломщик 2:

* по технической оснащенности – персональный компьютер;
* по конечной цели – обнаружение слабости в алгоритме;
* по доступу к шифрующим средствам – “внутренний” нарушитель;
* по уровню подготовки – математический аппарат;
* по первичной информации о средстве шифрования – пользователь;
* по возможности кооперации – одиночка;

В соответствии со следующим этапом определение множества атак основывалось на статьях [4] и [10], наиболее подходящие:

Атаки с выбором шифротекста (Chosen Cyphertext Attack – CCA) – это вид атак, при котором у атакующего есть определенное количество криптограмм и возможность получить соответствующие им открытые текста [12].

Классификация:

* по доступу к открытому и зашифрованному тексту – только шифротекста;
* по контролю над процессом шифрования – активная, с имитацией и подменой сообщения;
* по объему необходимых ресурсов – память и время;
* по степени применимости к различным шифрам – универсальная;
* по используемым средствам – математические методы;
* по последствиям атаки – нарушение конфиденциальности;
* по возможности распараллеливания – распределенная;

Атаки с выбором открытого текста (Chosen Plaintext Attack – CPA) – это вид атак, при котором у атакующего есть определенное количество открытых текстов с соответствующими ими криптограммами [12].

Классификация:

* по доступу к открытому и зашифрованному тексту – доступ к открытому тексту;
* по контролю над процессом шифрования – пассивная;
* по объему необходимых ресурсов –данные;
* по степени применимости к различным шифрам – универсальная;
* по используемым средствам – математические методы;
* по последствиям атаки – нарушение конфиденциальности;
* по возможности распараллеливания – не распределенная;

Атаки на побочные каналы (Side channel attacks – SCA) — это вид криптографических атак, использующих информацию, полученную по этим каналам. Под таковой понимается информация, которая может быть получена с устройства шифрования и не является при этом ни открытым текстом, ни шифртекстом. Как правило, предполагается, что криптографические вычисления реализуются в виде идеальных «черных ящиков» в том смысле, что текущее состояние вычислительного процесса закрыто от враждебного наблюдателя. Единственной информацией, доступной криптоаналитику, является общая структура алгоритма шифрования, шифртекст и, зачастую, соответствующий ему открытый текст [11]. НУЖНО ПЕРЕПИСАТЬ

Классификация:

* по доступу к открытому и зашифрованному тексту – информация из побочных каналов;
* по контролю над процессом шифрования – активная;
* по объему необходимых ресурсов –память, время и данные;
* по степени применимости к различным шифрам – универсальная;
* по используемым средствам – перехватчики физических параметров процесса шифрования;
* по последствиям атаки – нарушение конфиденциальности;
* по возможности распараллеливания – распределенная;

Для подсчёта функции взаимного влияния, которая определена в формуле (10) необходимо задать конкретные значения влияния параметров атаки на параметры криптосистемы, для того чтобы не перебирать все возможные пары рассмотрим такие , что их взаимное влияние будет минимально, т.к. в формуле (10) находится именно минимум.

Для CCA:



Для CPA:



Для SCA:



## Анализ полученных результатов

**НАДО СДЕЛАТЬ**

# ЗАКЛЮЧЕНИЕ

В рамках данной курсовой работы была рассмотрена тема " Исследование применения схемы Proxy re-encryption для безопасной передачи конфиденциальных данных в финансовом секторе". В процессе исследования были осуществлены следующие шаги:

Проведен обзор существующих решений для безопасной передачи конфиденциальных данных. Были изучены различные методы и протоколы, используемые для защиты данных в процессе их передачи, включая схему Proxy re-encryption.

Описаны основные принципы криптографии, необходимые для понимания работы схемы Proxy re-encryption. Рассмотрены алгоритмы шифрования, электронной подписи и асимметричного шифрования на основе эллиптических кривых.

Проведен анализ преимуществ и недостатков существующих методов защиты данных. Выявлены их особенности, сильные и слабые стороны, а также области применения.

Описана схема Proxy re-encryption и принцип ее работы. Рассмотрены ключевые компоненты схемы, включая генерацию ключей, операции шифрования и расшифровки, а также алгоритмы Proxy re-encryption.

Разработан подход к разработке алгоритмического и программного обеспечения для схемы Proxy re-encryption. Выбраны соответствующие инструменты, такие как онлайн-среда Remix Ethereum и локальная сеть блокчейна Ethereum Ganache, для разработки и тестирования смарт контрактов.

Описана методология тестирования, включающая юнит-тестирование, интеграционное тестирование, функциональное тестирование и нагрузочное тестирование. Обозначены цели каждого вида тестирования и их вклад в обеспечение качества и безопасности схемы Proxy re-encryption.

Необходимо отметить, что практическая часть работы, включающая разработку алгоритмического и программного обеспечения, а также проведение тестирования, осталась за пределами данной работы.

В дальнейшем планируется реализация и тестирование схемы Proxy re-encryption в реальной среде, чтобы подтвердить ее эффективность и безопасность. Это позволит дополнительно проверить и оценить применимость схемы в различных сценариях использования и ее соответствие требованиям безопасности.

В целом, данная работа является важным шагом в исследовании и разработке схемы Proxy re-encryption, которая представляет собой перспективное решение для безопасной передачи конфиденциальных данных. Исследование и разработка в данной области имеют большую практическую значимость и могут применяться в различных сферах, где требуется надежная защита конфиденциальной информации.

В конечном итоге, схема Proxy re-encryption может стать важным инструментом для обеспечения безопасности и конфиденциальности данных в современном информационном обществе.

НУЖНО ДОБАВИТЬ РЕЗУЛЬТАТЫ ПОДСЧЕТОВ ПО МАТ МОДЕЛИ

# СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Гатченко Н.А., Исаев А.С., Яковлев А.Д. «Криптографическая защита информации» – СПб: НИУ ИТМО, 2012. – 142 с.
2. Бузов Г.A. Защита от утечки информации по техническим каналам: Учебн. пособие / Бузов Г.A., Калинин C.B., Кондратьев A.B.- M.: Горячая линия - Телеком, 2005. - 416 c.
3. Manzoor, A., Liyanage, M., Braeken, A., Kanhere, S. S., & Ylianttila, M. Blockchain based Proxy Re-Encryption Scheme for Secure IoT Data Sharing, – 2019
4. Nunez D., Agudo I., Lopez J. Proxy Re-Encryption: Analysis of Constructions and its Application to Secure Access Delegation, – 2017.
5. Кнэпп Э. Эллиптические кривые. – М.: Факториал, 2004. – 488 c.
6. Savelieva A. Formal methods and tools for evaluating crypto- graphic systems security // St. Petersburg, ISP RAS, In Proceedings of the Second Spring Young Researchers Colloquium on Software Engineering (SYRCoSE’2008), 2008. Часть 1. С. 33—36.
7. Bashir I. Mastering Blockchain: A Technical Reference Guide to What's Under the Hood of Blockchain, from Cryptography to DeFi and NFTs. – Packt Publishing, Limited, 2023.
8. Авдошин С. М., Савельева А. А. Проблемы оценки криптозащищенности информационных систем // “Новые информационные технологии”. Тез. докл. XVI Международной студенческой школы-семинара. М.: МИЭМ, 2008 С. 15-29
9. Мясников С. О., Намиот Д. Е. Инструменты нагрузочного тестирования // Прикладная информатика. – 2018. – Т. 13. – №. 1. – С. 92-102.
10. Carter P. 6 of the biggest threats banks faced in 2023 [Электронный ресурс] // American Banker. – Режим доступа: <https://www.americanbanker.com/list/6-of-the-biggest-threats-banks-faced-in-2023>
11. Жуков А.Е. Криптоанализ по побочным каналам (side channel attacks) // Защита информации. Инсайд. – 2010 № 5 (35). – C. 28-33.
12. Петров А.А. Компьютерная безопасность. Криптографические методы защиты. – М.: ДМК Пресс, 2022. – 448 с.

# ПРИЛОЖЕНИЕ A

ca.py

**import** http.server  
**import** ssl  
**import** json  
**import** logging  
  
**from** util **import** convert\_object\_to\_hex\_str, setup\_pre  
  
  
logging.basicConfig(level=logging.INFO)  
  
pre, group = setup\_pre()  
(master\_secret\_key, params) = pre.setup()  
  
  
**class** Handler(http.server.SimpleHTTPRequestHandler):  
 **def** do\_POST(self):  
 **if** self.path == **'/key'**:  
 content\_length = int(self.headers[**'Content-Length'**])  
 post\_data = self.rfile.read(content\_length)  
  
 **try**:  
 data = json.loads(post\_data.decode(**'utf-8'**))  
  
 sender\_id = data.get(**'sender\_id'**)  
 logging.info(**"Got key request from: %s"**, sender\_id)  
  
 id\_secret\_key = pre.keyGen(master\_secret\_key, sender\_id)  
  
 \_, \_, params\_hex\_str, id\_secret\_key\_hex\_str = convert\_object\_to\_hex\_str(  
 group,  
 id\_secret\_key=id\_secret\_key,  
 params=params  
 )  
  
 response\_message = json.dumps({  
 **'id\_secret\_key\_hex\_str'**: id\_secret\_key\_hex\_str,  
 **'params\_hex\_str'**: params\_hex\_str,  
 })  
  
 self.send\_response(200)  
 self.send\_header(**'Content-type'**, **'text/html'**)  
 self.end\_headers()  
 self.wfile.write(response\_message.encode(**'utf-8'**))  
  
 **except** json.JSONDecodeError:  
 self.send\_response(400)  
 self.send\_header(**'Content-type'**, **'text/html'**)  
 self.end\_headers()  
 self.wfile.write(b'Invalid JSON received')  
  
 **return  
  
 return** super().do\_GET()  
  
  
**def** main():  
 server\_address = (**'0.0.0.0'**, 1026) *#* ***TODO change to 443*** httpd = http.server.HTTPServer(server\_address, Handler)  
 *# httpd.socket = ssl.wrap\_socket(* ***TODO: uncomment*** *# httpd.socket,  
 # server\_side=True,  
 # certfile='/etc/nginx/ssl/cert.pem',  
 # keyfile='/etc/nginx/ssl/key.pem',  
 # ssl\_version=ssl.PROTOCOL\_TLS  
 # )* httpd.serve\_forever()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

sender.py:

**import** socket  
**import** argparse  
**import** json  
**import** logging  
  
**from** util **import** get\_key\_params, setup\_pre, convert\_object\_to\_hex\_str, send\_large\_message  
**from** constants **import** CA\_URL  
  
logging.basicConfig(level=logging.INFO)  
pre, group = setup\_pre()  
ID = **"client-a"**ID2 = **"client-b"  
  
  
def** main(host\_ip, host\_port):  
 id\_secret\_key, params = get\_key\_params(CA\_URL, ID, group)  
  
 client\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
  
 client\_socket.connect((host\_ip, host\_port))  
  
 message = **"<confidential message>"** encrypted\_message = pre.encrypt(params, ID, message)  
 rekey = pre.rkGen(params, id\_secret\_key, ID, ID2)  
 print(**f"PARAMS: {**params**}"**)  
 print(**f"MESSAGE: {**encrypted\_message**}"**)  
 print(**f"REKEY: {**rekey**}"**)  
 encrypted\_message\_hex\_str, rekey\_hex\_str, \_, \_ = convert\_object\_to\_hex\_str(  
 group,  
 message=encrypted\_message,  
 rekey=rekey  
 )  
  
 print(rekey\_hex\_str)  
 print(encrypted\_message\_hex\_str)  
  
 message\_data = json.dumps({  
 **'sender\_id'**: ID,  
 **'encrypted\_message\_hex\_str'**: encrypted\_message\_hex\_str,  
 **'rekey\_hex\_str'**: rekey\_hex\_str,  
 })  
  
 send\_large\_message(client\_socket, message\_data)  
  
 response = client\_socket.recv(1024)  
 logging.info(**"Get response from client: %s"**, response.decode(**'utf-8'**))  
  
 client\_socket.close()  
  
  
**def** parse\_args():  
 parser = argparse.ArgumentParser()  
 parser.add\_argument(**"--host-ip"**, type=str, default=**"0.0.0.0"**)  
 parser.add\_argument(**"--host-port"**, type=int, default=1024)  
 parser.add\_argument(**"--client-name"**, type=str, default=**"client-a"**)  
  
 **return** parser.parse\_args()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 args = parse\_args()  
 main(args.host\_ip, args.host\_port)

proxy.py

**import** requests  
**import** socket  
**import** argparse  
**import** logging  
  
  
**from** util **import** receive\_large\_message  
  
  
logging.basicConfig(level=logging.INFO)  
  
  
**def** main(args):  
 server\_host = **'0.0.0.0'** server\_port = 1024  
  
 server\_socket = socket.socket(socket.AF\_INET, socket.SOCK\_STREAM)  
 server\_socket.bind((server\_host, server\_port))  
 server\_socket.listen(5)  
  
 print(**f"Server listen {**server\_host**}:{**server\_port**}"**)  
  
 **while True**:  
 client\_socket, client\_address = server\_socket.accept()  
  
 message\_data = receive\_large\_message(client\_socket)  
 *# if not data:  
 # break* print(**f"Got encrypted message from client: {**message\_data**}"**) *#* ***TODO change to sender id*** url = args.client\_url  
 response = requests.post(url, data=message\_data) *#* ***TODO: verify=args.client\_cert*** client\_socket.send(response.text.encode(**'utf-8'**))  
 client\_socket.close()  
  
  
**def** parse\_args():  
 parser = argparse.ArgumentParser()  
 *#* ***TODO: change to client-b.test-zone.ru*** parser.add\_argument(**"--client-url"**, type=str, default=**"http://127.0.0.1:1028/secure"**)  
 parser.add\_argument(**"--client-cert"**, type=str, metavar=**"PATH"**,  
 default=**"/usr/local/share/ca-certificates/client-b.crt"**)  
  
 **return** parser.parse\_args()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 args = parse\_args()  
 main(args)

receiver.py

**import** http.server  
**import** json  
**import** logging  
  
**from** util **import** get\_key\_params, setup\_pre, convert\_hex\_str\_to\_object, receive\_large\_message  
**from** constants **import** CA\_URL  
  
  
logging.basicConfig(level=logging.INFO)  
pre, group = setup\_pre()  
ID = **"client-b"***# REKEY: {'N': "<class 'integer.Element'>", 'R': "<class 'pairing.Element'>"}  
# {  
# MESSAGE 'S': "<class 'pairing.Element'>", 'C': {  
# 'A': "<class 'pairing.Element'>",  
# 'B': "<class 'pairing.Element'>",  
# 'C': "<class 'integer.Element'>"  
# }  
# }***class** Handler(http.server.SimpleHTTPRequestHandler):  
 **def** do\_POST(self):  
 **if** self.path == **'/secure'**:  
 content\_length = int(self.headers[**'Content-Length'**]) *# Получаем размер данных* post\_data = self.rfile.read(content\_length) *# Читаем данные* **try**:  
 *#* ***TODO add logic to check key lifecycle*** id\_secret\_key, params = get\_key\_params(CA\_URL, ID, group)  
  
 logging.info(**"Got data: %r"**, post\_data)  
 data = json.loads(post\_data.decode(**'utf-8'**))  
  
 sender\_id = data.get(**'sender\_id'**)  
 encrypted\_message\_hex\_str = data.get(**'encrypted\_message\_hex\_str'**)  
 rekey\_hex\_str = data.get(**'rekey\_hex\_str'**)  
 encrypted\_message, rekey, \_, \_ = convert\_hex\_str\_to\_object(  
 group,  
 message\_hex\_str=encrypted\_message\_hex\_str,  
 rekey\_hex\_str=rekey\_hex\_str  
 )  
  
 logging.info(**f"PARAMS: {**params**}"**)  
 logging.info(**f"MESSAGE: {**encrypted\_message**}"**)  
 logging.info(**f"REKEY: {**rekey**}"**)  
  
 *#* ***TODO: move this logic to proxy*** ciphertext2 = pre.reEncrypt(params, **"client-a"**, rekey, encrypted\_message)  
  
 *#* ***TODO: add correct id params*** decrypted\_message = pre.decryptSecondLevel(params, id\_secret\_key, **"client-a"**, ID, ciphertext2)  
 logging.info(**"Decrypted message: %s, from client: %s"**, decrypted\_message, sender\_id)  
  
 self.send\_response(200)  
 self.send\_header(**'Content-type'**, **'text/html'**)  
 self.end\_headers()  
 response\_message = **"Data received"**.encode()  
 self.wfile.write(response\_message)  
  
 **except** json.JSONDecodeError **as** e:  
 logging.info(**"Got error %s"**, e)  
 self.send\_response(400)  
 self.send\_header(**'Content-type'**, **'text/html'**)  
 self.end\_headers()  
 self.wfile.write(b'Invalid JSON received')  
  
 **return  
  
 return** super().do\_GET()  
  
  
**def** main():  
 server\_address = (**'0.0.0.0'**, 1028) *#* ***TODO change to 443*** httpd = http.server.HTTPServer(server\_address, Handler)  
 *# httpd.socket = ssl.wrap\_socket(* ***TODO: uncomment*** *# httpd.socket,  
 # server\_side=True,  
 # certfile='/etc/nginx/ssl/cert.pem',  
 # keyfile='/etc/nginx/ssl/key.pem',  
 # ssl\_version=ssl.PROTOCOL\_TLS  
 # )* httpd.serve\_forever()  
  
  
**if** \_\_name\_\_ == **"\_\_main\_\_"**:  
 main()

smart\_contract.py

**from** web3 **import** Web3  
**from** cryptography.hazmat.primitives.asymmetric.ec **import** EllipticCurvePublicKey  
**from** cryptography.hazmat.primitives **import** serialization  
**from** cryptography.hazmat.backends **import** default\_backend  
**import** logging  
  
CONTRACT\_ADDRESS = **"0x7Dc0fa574F7cb11FE3E86CcD1e39AC4b7e17d658"** *#* ***TODO DELETE*def** convert\_to\_string(pub\_key: EllipticCurvePublicKey) -> str:  
 pub\_key = pub\_key.public\_bytes(  
 encoding=serialization.Encoding.PEM,  
 format=serialization.PublicFormat.SubjectPublicKeyInfo  
 )  
  
 **return** pub\_key.decode(**'utf-8'**)  
  
  
**def** convert\_to\_ec(pub\_key: str):  
 public\_key = serialization.load\_pem\_public\_key(pub\_key.encode(**'utf-8'**), backend=default\_backend())  
  
 **return** public\_key  
  
  
**class** SmartContract:  
 abi = [  
 {**'inputs'**: [],  
 **'stateMutability'**: **'nonpayable'**,  
 **'type'**: **'constructor'** },  
 {  
 **'inputs'**: [  
 {**'internalType'**: **'string'**, **'name'**: **'clientName'**, **'type'**: **'string'**},  
 {**'internalType'**: **'string'**, **'name'**: **'pubKey'**, **'type'**: **'string'**}  
 ],  
 **'name'**: **'addPublicKey'**,  
 **'outputs'**: [],  
 **'stateMutability'**: **'nonpayable'**,  
 **'type'**: **'function'** },  
 {  
 **'inputs'**: [  
 {**'internalType'**: **'string'**, **'name'**: **'clientName'**, **'type'**: **'string'**}  
 ],  
 **'name'**: **'getPublicKey'**,  
 **'outputs'**: [{**'internalType'**: **'string'**, **'name'**: **''**, **'type'**: **'string'**}],  
 **'stateMutability'**: **'view'**,  
 **'type'**: **'function'** },  
 {  
 **'inputs'**: [],  
 **'name'**: **'owner'**,  
 **'outputs'**: [  
 {**'internalType'**: **'address'**, **'name'**: **''**, **'type'**: **'address'**}  
 ],  
 **'stateMutability'**: **'view'**, **'type'**: **'function'** }  
 ]  
  
 **def** \_\_init\_\_(self, blockchain\_addr, contract\_addr):  
 self.web3 = Web3(Web3.HTTPProvider(blockchain\_addr))  
  
 contract\_address = CONTRACT\_ADDRESS *#* ***TODO get from args*** self.contract = self.web3.eth.contract(address=contract\_address, abi=SmartContract.abi)  
  
 **def** addPublicKey(self, sender\_address, clientName, pubKey):  
 sender\_address = **'0x3eB4d8dF7D9B9E3Df03A294467df0666429FbEA7'** pubKey = convert\_to\_string(pubKey)  
  
 logging.info(**"Add public key for %s"**, clientName)  
 tx\_hash = self.contract.functions.addPublicKey(clientName, pubKey).transact({  
 **'from'**: sender\_address,  
 **'gas'**: 2000000,  
 })  
  
 tx\_receipt = self.web3.eth.wait\_for\_transaction\_receipt(tx\_hash)  
  
 **def** getPublicKey(self, clientName):  
 logging.info(**"Get public key for client: %s"**, clientName)  
 pub\_key = self.contract.functions.getPublicKey(clientName).call()  
  
 **return** pub\_key  
  
  
contract = SmartContract(**"http://127.0.0.1:8545"**, **""**) *#* ***TODO fix args***

util.py

**import** os  
**import** json  
**import** zlib  
  
**import** requests  
  
**from** datetime **import** datetime, timedelta  
**from** charm.core.engine.util **import** objectToBytes, bytesToObject  
**from** charm.toolbox.pairinggroup **import** PairingGroup  
**from** charm.schemes.pre\_mg07 **import** PreGA  
**from** charm.toolbox.conversion **import** Conversion  
  
  
**def** serialize\_data(group, data):  
 **if** isinstance(data, dict):  
 serialized\_data = {}  
 **for** key, value **in** data.items():  
 serialized\_data[key] = serialize\_data(group, value)  
 **return** serialized\_data  
 **else**:  
 **try**:  
 **return** objectToBytes(data, group).hex()  
 **except** TypeError:  
 **return** Conversion.IP2OS(data).hex()  
  
  
**def** deserialize\_data(group, serialized\_data):  
 **if** isinstance(serialized\_data, dict):  
 data = {}  
 **for** key, value **in** serialized\_data.items():  
 data[key] = deserialize\_data(group, value) *# Рекурсивная обработка для словарей* **return** data  
 **else**:  
 **try**:  
 **return** bytesToObject(bytes.fromhex(serialized\_data), group)  
 **except** zlib.error:  
 **return** Conversion.OS2IP(bytes.fromhex(serialized\_data), element=**True**)  
  
  
**def** convert\_object\_to\_hex\_str(group, message=**None**, rekey=**None**, params=**None**, id\_secret\_key=**None**):  
 id\_secret\_key\_hex\_str = **None  
 if** id\_secret\_key:  
 id\_secret\_key\_bytes = objectToBytes(id\_secret\_key, group)  
 id\_secret\_key\_hex\_str = id\_secret\_key\_bytes.hex()  
  
 params\_hex\_str = **None  
 if** params:  
 params\_hex\_str = {k: objectToBytes(v, group).hex() **for** k, v **in** params.items()}  
  
 message\_hex\_str = **None  
 if** message:  
 message\_hex\_str = serialize\_data(group, message)  
  
 rekey\_hex\_str = **None  
 if** rekey:  
 rekey\_hex\_str = serialize\_data(group, rekey)  
  
 **return** message\_hex\_str, rekey\_hex\_str, params\_hex\_str, id\_secret\_key\_hex\_str  
  
  
**def** convert\_hex\_str\_to\_object(group, message\_hex\_str=**None**, rekey\_hex\_str=**None**,  
 params\_hex\_str=**None**, id\_secret\_key\_hex\_str=**None**):  
 id\_secret\_key = **None  
 if** id\_secret\_key\_hex\_str:  
 id\_secret\_key\_bytes = bytes.fromhex(id\_secret\_key\_hex\_str)  
 id\_secret\_key = bytesToObject(id\_secret\_key\_bytes, group)  
  
 params = **None  
 if** params\_hex\_str:  
 params = {k: bytesToObject(bytes.fromhex(v), group) **for** k, v **in** params\_hex\_str.items()}  
  
 message = **None  
 if** message\_hex\_str:  
 message = deserialize\_data(group, message\_hex\_str)  
  
 rekey = **None  
 if** rekey\_hex\_str:  
 rekey = deserialize\_data(group, rekey\_hex\_str)  
  
 **return** message, rekey, params, id\_secret\_key  
  
  
**def** get\_key\_params(url, id, group):  
 message\_data = json.dumps({  
 **'sender\_id'**: id,  
 })  
 response = requests.post(url, data=message\_data)  
 id\_secret\_key\_hex\_str = json.loads(response.text).get(**"id\_secret\_key\_hex\_str"**)  
 params\_hex\_str = json.loads(response.text).get(**"params\_hex\_str"**)  
  
 \_, \_, params, id\_secret\_key = convert\_hex\_str\_to\_object(group, id\_secret\_key\_hex\_str=id\_secret\_key\_hex\_str,  
 params\_hex\_str=params\_hex\_str)  
  
 **return** id\_secret\_key, params  
  
  
**def** setup\_pre():  
 group = PairingGroup(**'SS512'**, secparam=1024)  
 pre = PreGA(group)  
 **return** pre, group  
  
  
**def** check\_private\_key\_path():  
 last\_modified\_time = os.path.getmtime(os.path.join())  
 last\_modified\_date = datetime.fromtimestamp(last\_modified\_time)  
 now = datetime.now()  
 time\_difference = now - last\_modified\_date  
  
 *# refresh private key every 2 days* **if** time\_difference < timedelta(days=2):  
 **pass** *# TBD add logic***def** send\_large\_message(socket, message\_data, chunk\_size=4096):  
 *# Конвертируем данные в байты* message\_bytes = message\_data.encode(**'utf-8'**)  
  
 *# Отправляем размер сообщения перед самим сообщением* total\_size = len(message\_bytes)  
 socket.sendall(total\_size.to\_bytes(4, **'big'**))  
  
 *# Разбиваем сообщение на части и отправляем каждую часть* **for** i **in** range(0, total\_size, chunk\_size):  
 chunk = message\_bytes[i:i + chunk\_size]  
 socket.sendall(chunk)  
  
  
**def** receive\_large\_message(socket, chunk\_size=4096):  
 *# Сначала получаем размер сообщения* total\_size\_bytes = socket.recv(4)  
 total\_size = int.from\_bytes(total\_size\_bytes, **'big'**)  
  
 *# Теперь читаем данные до тех пор, пока не получим полное сообщение* chunks = []  
 bytes\_recd = 0  
 **while** bytes\_recd < total\_size:  
 chunk = socket.recv(min(total\_size - bytes\_recd, chunk\_size))  
 **if** chunk == b'':  
 **raise** RuntimeError(**"Socket connection broken"**)  
 chunks.append(chunk)  
 bytes\_recd += len(chunk)  
  
 *# Собираем все части в одно сообщение* **return** b''.join(chunks).decode(**'utf-8'**)

constants.py

CA\_URL = **"http://127.0.0.1:1026/key"**CONTRACT\_ADDRESS = **"0x7Dc0fa574F7cb11FE3E86CcD1e39AC4b7e17d658"**