Министерство науки и высшего образования Российской Федерации
Санкт-Петербургский политехнический университет Петра Великого
Институт кибербезопасности и защиты информации

Работа допущена к защите
Директор Института
кибербезопасности и защиты
информации, д.т.н., проф.
_____ Д.П. Зегжда
«___»_____2021 г.

ВЫПУСКНАЯ КВАЛИФИКАЦИОННАЯ РАБОТА

РАБОТА БАКАЛАВРА

РАЗРАБОТКА ДОВЕРЕННОГО ОРАКУЛА ДЛЯ ОБРАБОТКИ ПЕРСОНАЛЬНЫХ ДАННЫХ В СЕТЯХ БЛОКЧЕЙН

по направлению подготовки (специальности)
10.03.01 Информационная безопасность

Направленность (профиль)
10.03.01_03 Безопасность компьютерных систем

Выполнил
студент гр. 4831001/70301 А.А. Северенчук

Руководитель
доцент ИКиЗИ,
к.т.н. Д.А. Москвин

Консультант
ассистент ИКиЗИ А.Д. Дахнович

Санкт-Петербург
2021

РЕФЕРАТ

На 66 с., 28 рисунков, 3 таблицы, 2 приложения.

КЛЮЧЕВЫЕ СЛОВА: ПЕРСОНАЛЬНЫЕ ДАННЫЕ, БЛОКЧЕЙН, ДОВЕРЕННАЯ СРЕДА ИСПОЛНЕНИЯ, HYPERLEDGER FABRIC, INTEL SGX, SIDE-CHANNEL ATTACKS, ИНДЕКС ДОВЕРИЯ КЛИЕНТОВ.

Тема выпускной квалификационной работы: «Разработка доверенного оракула для обработки персональных данных в сетях блокчейн».

Цель данной работы заключается в разработке доверенного оракула, обеспечивающего конфиденциальность персональных данных в блокчейн сети с сохранением прозрачности транзакций и использованием репутационной оценки участников. Объектом исследования являются механизмы передачи и обработки персональных данных, а предметом – степень безопасности найденных решений.

Задачи, которые решались в ходе исследования:

1. Рассмотрение технологии децентрализованных сетей на примере Hyperledger Fabric и механизма доверенной среды исполнения на примере Intel SGX.
2. Реализация тестовой сети блокчейн с подключением к ней внешнего оракула для надежной обработки персональных данных.
3. Описание смарт-контрактов для верификации кода анклава Intel SGX и создания рейтинга оракула на основе индекса доверия.
4. Тестирование доверенного оракула и сравнение с другими решениями.

Работа основана на базе собранных данных о методах построения децентрализованных сетей и разработки приложений с использованием анклавов. В разработке решения применялись язык программирования Go и Си, а также разрабатывался специальный shell-код для запуска тестовой сети.

В результате было разработано решение с использованием блокчейна и доверенной среды исполнения для защиты персональных данных от кражи злоумышленниками. Предложены дальнейшие направления развития данной работы для системы умного голосования и в медицинской сфере. Проведенные тесты показали эффективность работы доверенного оракула, обеспечивающего конфиденциальность используемых персональных данных.

THE ABSTRACT

66 pages, 28 figures, 3 tables.

KEY WORDS: PERSONAL INFORMATION, BLOCKCHAIN, TRUSTED EXECUTION ENVIRONMENT, HYPERLEDGER FABRIC, INTEL SGX, SIDE-CHANNEL ATTACKS, CUSTOMER CONFIDENCE INDEX.

The subject of the graduate qualification work is «Development of a trusted oracle for processing personal data in blockchain networks».

The purpose of this work is to develop a trusted oracle that ensures the confidentiality of personal data in the blockchain network while maintaining the transparency of transactions and using the reputation of the participants. The object of the research is the mechanisms of transmission and processing of personal data, and the subject is the degree of security of the solutions found.

The research set the following goals:

1. Consideration of the technology of decentralized networks using the example of Hyperledger Fabric and the mechanism of a trusted execution environment using the example of Intel SGX.

2. Implementation of a blockchain test network with an external oracle connected to it for reliable processing of personal data.

3. Description of smart contracts for verifying the Intel SGX enclave code and creating an oracle rating based on the trust index.

4. Testing of the trusted oracle and comparison with other solutions.

The work is based on the collected data on the methods of building decentralized networks and developing applications using enclaves. In the development of the solution, the Go and C programming languages were used, and a special shell code was developed to launch the test network.

As a result, a solution was developed using blockchain and a trusted execution environment to protect personal data from being stolen by hackers. Further directions for the development of this work are proposed for the smart voting system and in the medical field. The tests carried out have shown the effectiveness of the work of the trusted oracle, which ensures the confidentiality of the personal data used.

СОДЕРЖАНИЕ

ОПРЕДЕЛЕНИЯ, ОБОЗНАЧЕНИЯ И СОКРАЩЕНИЯ

Аутсорсинг	Передача организацией определённых видов или функций производственной предпринимательской деятельности другой компании, действующей в нужной области
Хеш	Результат обработки данных хеш-функцией
ПДн	Персональные данные
SGX	Sortware Guard Extentions
HLF	Hyperledger Fabric
HLA	Hyperledger Avalon
DLT	Distributed Ledger Technology
TEE	Trusted Execution Environment
CA	Certificate Authority
IAS	Intel Attestation Service

ВВЕДЕНИЕ

В V веке до н.э. в Древней Греции возникли первые признаки демократии, с тех пор прошло более двух тысяч лет и безусловно общество изменилось. Поменялись порядки, взаимоотношения между людьми и странами, в 2021 году все больше таких взаимоотношений регулируются с помощью современных технологий, которые эволюционируют каждый год и заставляют человека приспосабливаться к новым условиям. Но несмотря на эти изменения, некоторые принципы остаются прежними, так как они характерны природе человека. Желание людей объединяться в группы и сообщества привело к появлению глобальной сети интернет, стремление человека осваивать новые территории толкает вперед исследования в области космоса, а с появлением цивилизованного общества человеку стало важно, чтобы его мнение учитывалось в вопросах распределения власти и управления государством. Так, сосредоточение власти в руках небольшого количества людей накладывает ограничения на экономические процессы. Даже простые финансовые взаимодействия, такие как перевод денег с счета на счет, требуют регулирования со стороны банков, то есть человеку ничего не остается, как доверять третьим лицам. Вопрос доверия является краеугольным камнем в финансовой сфере, но, к сожалению, природа человека включает в себя и такие качества, как алчность и жадность, история знает много случаев обмана честных людей, когда у них не было возможности доказать свою правоту. А что, если бы люди полагались в спорных ситуациях на технологии, которые нельзя обмануть или подкупить? Одним из примеров такой технологии является блокчейн.

Блокчейн – это децентрализованная сеть, в которой доверие к общей сети строится на распределении информации между участниками. Таким образом, каждый участник или узел сети обладает той же информацией, что и все остальные участники. При этом, любое изменение состояния сети отражается на каждом узле. Такой принцип исключает возможность подделки информации, так как любые ложные данные на одном из узлов сети не будут совпадать с данными других узлов, и такая информация будет восприниматься как недостоверная.

Основное и самое известное применение блокчейн – это криптовалюты. «В 2008 году некто под псевдонимом «Сатоси Накомото» описал новый протокол для системы прямых электронных расчетов с помощью криптовалюты под названием «биткоин». Этот протокол установил ряд правил, которые обеспечивали целостность информации, передаваемой между миллиардами устройств напрямую, без обращения к надежной третьей стороне.» [1]. Кроме «биткоина» появилось множество других криптовалют, которые завоевали интерес инвесторов и разработчиков по всему миру.

Принцип децентрализации блокчейн также используется сегодня для построения корпоративных решений и взаимодействий B2B (Business to Business). Многие крупные компании и предприятия заинтересованы в применении децентрализованных решений по ряду причин. Во-первых, использование блокчейн облегчает юридические вопросы и экономит средства участников. «Передача ценностей как в больших, так и в очень малых объемах без посредников значительно уменьшит стоимость и увеличит скорость платежей» [1]. Во-вторых, прозрачность всех проводимых транзакций позволяет определить надежность и честность партнера до заключения с ним соглашений. В-третьих, использование «умных контрактов» в рамках сети блокчейн почти полностью исключает возможность жульничества сторон, так как их взаимодействие прописано в программе и выполняется автоматически, таким образом бизнес-процессы становятся автоматизированными.

В эпоху цифрового маркетинга и больших данных, компании все больше стремятся увеличить продажи и лояльность клиентов за счет современных технологий. Сегодня почти каждая крупная компания собирает данные о своих клиентах, проводя различные опросы и голосования. Создание статистики на основе большого количества данных от разных клиентов помогает компаниям выстраивать эффективную маркетинговую стратегию и делать свой продукт или услугу более адаптированной под потребителя. В медицинской сфере обработка большого потока данных о разных пациентах с похожим диагнозом помогает выстроить лучшие подходы к лечению. Помимо этого, компании собирают

персональные данные о клиентах для рассылки предложений, учета истории болезни пациента, создания целевой аудитории. Персональные данные являются «новой нефтью» для компаний, ценность этой информации также понимают и мошенники, так в 2016 году была совершена кибератака на американскую компанию Uber. «В ходе инцидента были украдены данные 57 млн клиентов сервиса. После кибератаки хакеры отправили в Uber письмо с требованием выплатить им деньги за украденную информацию. В итоге Uber заплатила хакерам $100 тыс. за удаление похищенных данных и неразглашение о кибератаке.» [2].

Компании, деятельность которых заключается в производстве и продаже товаров или в предоставлении услуг, часто не обладают достаточной технической базой для расчета статистики и анализа полученных данных о клиентах, поэтому они вынуждены передавать эти сложные процессы на аутсорсинг в другие компании. Проблема заключается в том, что для получения качественных результатов необходимо передавать и персональные данные о клиентах, что может повлечь за собой негативные последствия. Например, такие данные могут быть проданы третьим лицам или, как в случае с Uber, украдены. Данная проблема может быть решена с помощью системы, в которой персональные данные остаются конфиденциальными. В рамках выпускной квалификационной работы будет произведено исследование подходов к построению доверенной системы для обработки персональных данных.

Актуальность исследования заключается в том, что с каждым годом информации о пользователях в интернете становится все больше, компании собирают данные о клиентах и используют их в своих целях. При этом каждый гражданин имеет право на неприкосновенность частной жизни в цифровой среде и необходимо обеспечить безопасность персональных данных от несогласованной передачи их третьим лицам.

Объектом исследования настоящей выпускной квалификационной работы являются механизмы передачи и обработки персональных данных, которые обеспечивают конфиденциальность этих данных.

Предметом исследования является степень безопасности найденных

решений, то есть насколько предложенные механизмы предотвращают кражу и раскрытие персональных данных.

Цель настоящей дипломной работы – разработка доверенного оракула, позволяющего обрабатывать персональные данные в сетях блокчейн с защитой их конфиденциальности. Для достижения поставленной цели необходимо решить следующие задачи:

– построить блокчейн сеть на базе Hyperledger Fabric с несколькими участниками;

– подключить к блокчейн сети оракул с возможность обработки данных в доверенной среде исполнения (Trusted Execution Environment);

– настроить связь между сетью и оракулом, обеспечив конфиденциальность данных криптографическими методам;

– создать и установить смарт-контракт, который проверяет хеш анклава Intel SGX и высчитывает рейтинг участников.

Теоретической основой выпускной квалификационной работы послужили исследования компании Intel в области доверенной среды исполнения Intel® Software Guard Extensions [3], проект Hyperledger Avalon и Hyperledger Fabric от компании Linux Foundation, исследование А.И. Савельева о «Проблемах применения законодательства о персональных данных в эпоху «больших данных» [4].

Практическая часть работы выполнялась на основании документации к opensource проекту Hyperledger Fabric, курса Hyperledger Fabric for developers LD272 от Linux Foundation, документации по языку программирования Go и документации по Intel® Software Guard Extensions [3].

Информационную базу исследования составили знания, полученные при изучении учебных дисциплин «Программно-аппаратные средства обеспечения информационной безопасности», «Криптографические методы защиты информации», «Компьютерные сети», «Безопасность операционных систем». При разработке выпускной квалификационной работы использовались публикации и научные издания, затрагивающие теоретические и прикладные аспекты

построения блокчейн сетей, доверенной среды исполнения и закона о персональных данных.

Степень научной разработанности проблемы. Исследованиям в области разработки механизмов доверенного исполнения посвящены статьи от компании Intel и ARM. Статья А.И. Савельева [4] посвящена исследованию «обезличенных» персональных данных и указывает на явные недостатки такого подхода. Исследования компании Hyperledger в рамках проекта Hyperledger Avalon показывают возможность объединения блокчейн платформы с механизмами доверенной среды исполнения.

Научная новизна представленного исследования заключается в следующем:

– создание механизма верификации кода анклава Intel SGX с помощью смарт-контракта;

– создание репутационной системы для участников сети;

– предоставление возможности обработки данных без нарушения их конфиденциальности.

Практическая значимость выпускной квалификационной работы заключается в возможности использования предложенного механизма для передачи и обработки персональных данных о клиентах на аутсорсинг с минимальными рисками нарушения конфиденциальности этих данных. Защищенность конфиденциальности персональных данных обеспечивается криптографическими методами и полагается на прозрачность сети блокчейн. Данный механизм может применяться для сбора ценных данных от нескольких компаний, объединенных общей сетью блокчейн и проводить анализ этих данных с помощью методов машинного обучения, получая важные статистические и другие данные.

1 ОПИСАНИЕ СЕТИ БЛОКЧЕЙН

Разработка технологии блокчейн представляет собой значительный научно-технический прорыв [5]. С точки зрения информационной безопасности, принцип децентрализации, обеспечивает такие качества сети, как доступность, независимость и защищенность.

1.1 Основные принципы построения децентрализованных сетей

Основной принцип децентрализованных сетей заключается в распределенном реестре. Аналогично бухгалтерской книге, в которую записываются все осуществляемые денежные переводы и начисления, в блокчейне хранятся записи о проведенных ранее операциях. Эти записи могут описывать как денежные переводы, так и заключенные контракты. Копии реестра со всей историей хранятся у всех участников сети. Блокчейн – это одноранговая сеть, ресурсы которой используются для подтверждения и одобрения каждой транзакции. В такой сети нет центрального управления и отсутствует возможность изменения и подделки реестра.

Записи в блокчейне организованы в «блоки», которые связаны друг с другом посредством криптографической проверки.

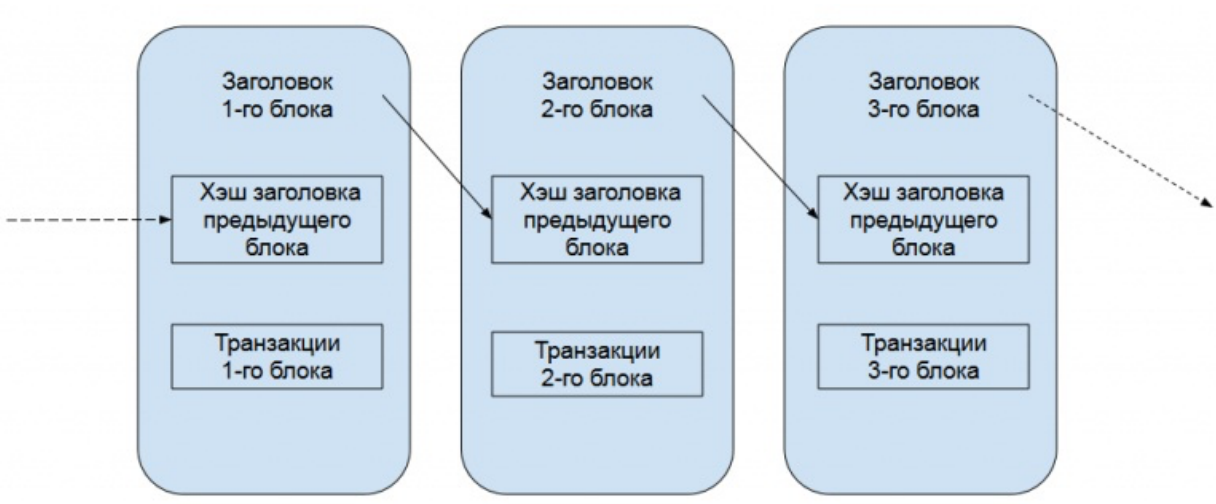

Рисунок 1.1 – Структура блоков в блокчейне

Каждая новая транзакция проходит проверку участниками, получает одобрение и сохраняется в блоке. Блоки формируют майнеры, решая сложную

математическую задачу. Обычно майнинг сводится к серии вычислений с перебором параметров для нахождения хеша с заданными свойствами, что требует больших операционных затрат [6]. Блок состоит из заголовка, хеша заголовка предыдущего блока и определенного количества транзакций. Блоки (blocks) объединяются в цепь (chain), отсюда и название «blockchain», то есть цепь, состоящая из блоков. Каждый блок действителен только тогда, когда он соотнесен с предыдущим. Данный принцип связности исключает возможность вносить изменения в реестр, так как для изменения одного блока необходимо будет переписать всю его историю в блокчейне. Потребуется более 50% от вычислительной мощности доступной в блокчейне, чтобы эффективно контролировать всю сеть, что в масштабах крупной системы практически невозможно, так, для проведения «атаки 51%» в сети биткоин потребуется взломать 150 000 серверов, одновременно [7].

Таким образом можно выделить несколько основных принципов в основе построения децентрализованных сетей:

1. Распределенность: блокчейн работает на компьютерах добровольцев по всему миру без центральной базы данных, которую можно было бы взломать [1].
2. Публичность: любой пользователь сети может просматривать блокчейн в любой момент и видеть историю транзакций в реестре.
3. Безопасность: для крупных сетей проведение «атаки 51%» практически неосуществимо из-за огромных вычислительных затрат. Помимо этого, в блокчейн используется система шифрования, применяющая публичные и приватные ключи для обеспечения конфиденциальности данных.

При построении децентрализованных сетей необходимо учитывать данные принципы в первую очередь. Сама по себе технология блокчейн гарантирует безопасность и стабильность операций, но при этом, также необходимо обеспечить правильную работу используемых протоколов и настройку компонентов сети, что является задачей разработчиков.

Существует множество блокчейн платформ, каждая обладает своими особенностями. Одни из самых распространенных платформ – Ethereum, Hyperledger Fabric и R3 Corda, их сравнение приведено в таблице 2.1.

Таблица 2.1 Сравнение блокчейн платформ

	Hyperledger Fabric	Ethereum	R3 Corda
Разработчики	Linux Foundation	Разработчики Ethereum	R3
Криптовалюта	Нет	ETH	Нет
Смарт контракты	Да (Go, JavaScript)	Да (Solidity)	Да (Kotlin, Java)
Достижение консенсуса	Протоколы консенсуса на основе согласованности участников	Майнинг, Proof-of-Work	Специфические способы согласования (нотариальные узлы)
Режим участия	Частная сеть, запрос разрешений	Открытая сеть, без запроса на разрешение	Частная, запрос разрешения
Особенности архитектуры	Модульная блокчейн-платформа	Открытая блокчейн-платформа	Специализированная платформа распределенного реестра для финансовой индустрии

Сравнив три распространённые блокчейн-платформы, была выбрана Hyperledger Fabric, так как в ней предусмотрено создание частной сети с обязательным запросом на разрешение участия, а также не используется криптовалюта, в отличии от Ethereum, что уменьшает вычислительные затраты. По сравнению с R3 Corda, Hyperledger Fabric имеет более обширную область применения и подойдет для самых разных сценариев, в ту очередь, как R3 Corda специализируется на финансовой индустрии. Также, возможность использования прикладных языков программирования, таких как Go и JavaScript облегчает процесс разработки смарт-контрактов для блокчейн сети Hyperledger Fabric.

1.2 Описание компонентов блокчейн на базе Hyperledger Fabric

Hyperledger Fabric (HLF) – это блокчейн бизнес-проект, поддерживаемый Linux Foundation [8]. HLF – это платформа с технологией распределенного реестра (distributed ledger technology - DLT), подходящая для промышленного использования. Это проект с открытым исходным кодом, спроектированный для решения промышленных задач [9]. Благодаря принципу открытого управления, проект развивается за счет независимых разработчиков, число которых достигает 200 человек из 148 организаций [10].

HLF имеет модульную архитектуру. В отличии от блокчейна с криптовалютой, HLF может использовать протоколы консенсуса, которые не требуют встроенной криптовалюты, а это позволяет реализовать сеть без процесса майнинга и уменьшить операционные затраты.

HLF состоит из следующих модульных компонентов:

– ordering service: устанавливает консенсус в последовательности транзакций и затем передает сформированные блоки пирам (peers);

– membership service provider (MSP): ставит в соответствие сущностям сети их криптографические учетные записи;

– смарт-контракты (Chaincode): определяют логику взаимодействия с блокчейном, выполняются внутри контейнерного окружения для изоляции (Docker). Они могут быть написаны на стандартных языках программирования (Go, JavaScript, Java);

– база данных: поддерживаются два типа одноранговых баз данных. LevelDB – это база данных состояний по умолчанию. CouchDB – настраиваемая база данных, которая позволяет моделировать данные в реестре в формате JSON;

– сменные политики по подтверждению и валидации (endorsement and validation policies);

– Certificate Authority: выдает сертификаты стандарта X.509 администраторам и узлам сети.

Как и любой другой блокчейн, HLF содержит реестр с историей транзакций. Реестр Hyperledger Fabric состоит из двух компонентов: World State

(состояние мира) и Blockchain (журнал транзакций). World State – база данных с кэшем текущих значений, которая меняется с каждым новым блоком. Blockchain – журнал транзакций, в который записываются все изменения, которые привели к текущему World State. Blockchain состоит из блоков с транзакциями, связанными хеш значениями, и он не может быть изменен. Каждый участник имеет копию реестра сети Hyperledger Fabric, в которой он состоит (рисунок 1.2).

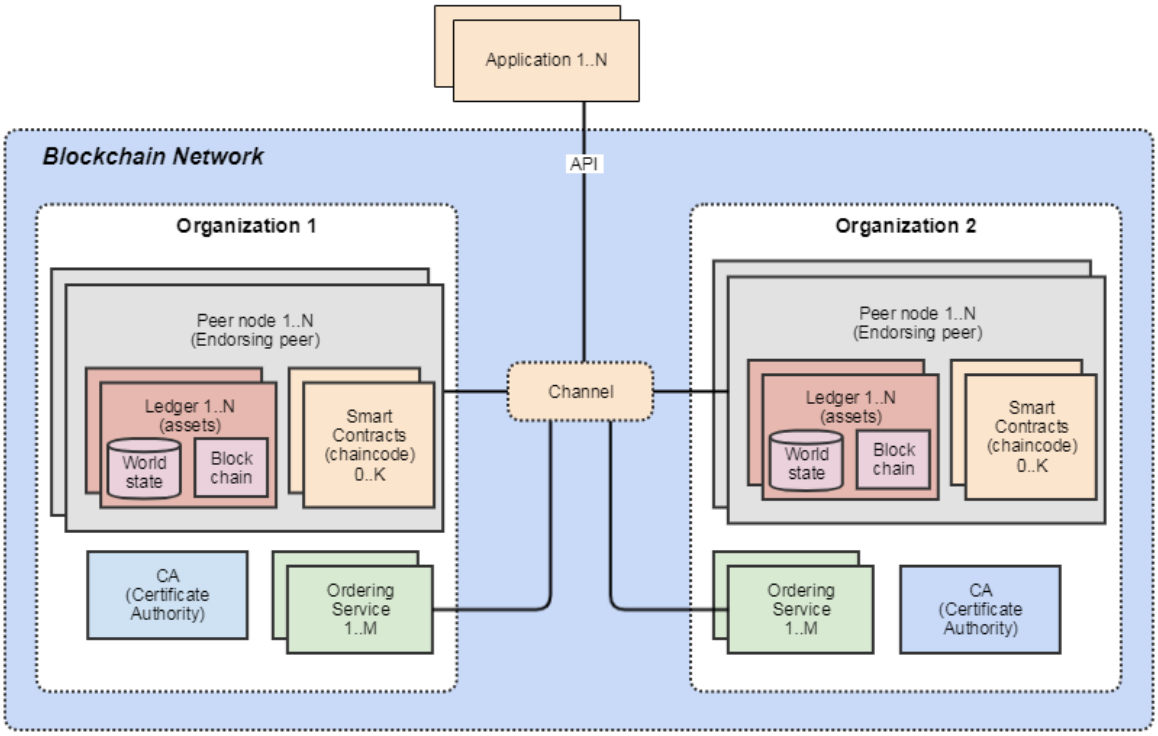

Рисунок 1.2 – Архитектура блокчейн сети HLF [9]

Участники сети общаются между собой через канал (channel). Одна блокчейн-сеть может содержать несколько каналов, каждый из которых работает независимо от других в своей изолированной среде, что обеспечивает изоляцию и конфиденциальность данных.

Для выполнения транзакций используются клиентские приложения, которые посредством API могут вызывать разные функции смарт-контракта и тем самым инициировать транзакции.

Каждая транзакция проходит несколько обязательных шагов перед тем, как стать частью блока:

1. Клиентское приложение отправляет предложение транзакции (transaction proposal) с вызовом функции чейнкода.

2. Endorsing Peer подтверждает transaction proposal с помощью MSP, если оно сформировано корректно и моделирует локальное выполнение транзакции, изменений реестра на этом этапе не происходит.

3. Клиентское приложение проверяет ответ от Endorsing Peer и отправляет новую транзакцию с подтверждением на Ordering service.

4. Ordering service принимает транзакции и упорядочивает их в блоки. Блоки доставляются всем участникам сети.

5. Endrorsing Peer проверяет транзакции в блоках, не были ли они изменены и фиксирует блоки в базе данных состояний, обновляя реестр.

Каждый участник сети имеет сертификат, подтверждающий его идентичность. Во время проверки транзакций, MSP проверяет и сертификат узла, генерирующего транзакцию. Центр сертификации (Certificate Authorities - CAs) генерирует пару ключей: приватный ключ для подписи транзакций и открытый для проверки транзакций. При этом, открытый ключ передается MSP для проверки транзакций (рисунок 1.3).

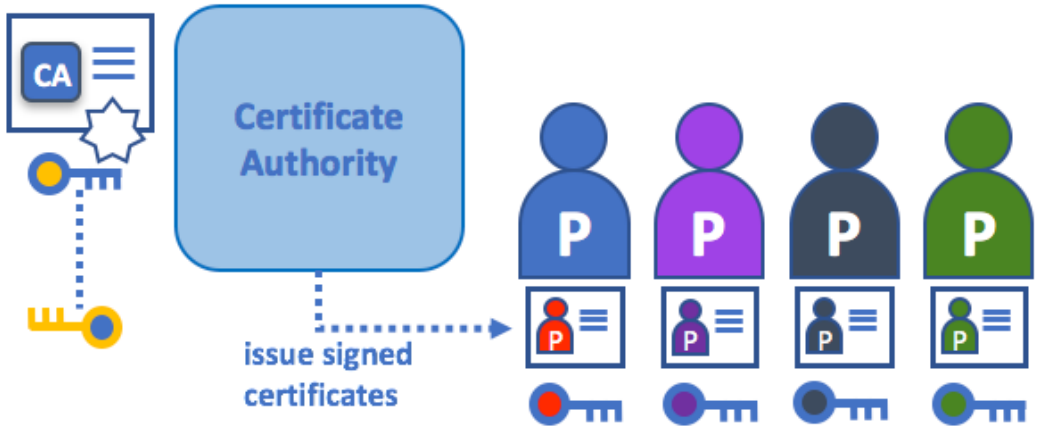

Рисунок 3 – Выдача сертификатов участникам сети [9]

Для проверки транзакций используется принцип консенсуса. Ordering service обрабатывает транзакции и проверяет, что достигнут консенсус, то есть

порядок и результат транзакций блока соответствуют явным критериям политики проверки (validation policies).

1.3 Смарт-контракты для блокчейн сетей

Первые упоминания о смарт-контрактах появились с платформой Ethereum. Основателем платформы является канадец русского происхождения Виталик Бутерин [1]. Основное отличие данного блокчейн проекта от уже существовавшего биткоина в том, что Ethereum поддерживает выполнение децентрализованных приложений, а именно смарт-контрактов.

Смарт-контракты – это код, выполняющийся в сети блокчейн, который описывает и регулирует взаимодействия между участниками сети и не нуждается в согласовании третьими лицами, например, нотариуса. Смарт-контракт описывает бизнес-логику блокчейн-приложений и распределен между узлами сети, чем обеспечивает безопасность и доверие к себе с помощью принципа консенсуса пиров.

В блокчейне Hyperledger Fabric смарт-контракты представлены чейнкодом (chaincode). Чейнкод – это программный код, написанный с использованием языков программирования Go, Node.js, или Java, для реализации необходимых интерфейсов [9]. Чейнкод выполняется из защищенного контейнера, изолированного от endorsing peers и является частью блокчейн-сети. Он инициализирует и управляет состоянием реестра посредством транзакций, отправляемых приложениями (рисунок 1.4).

Жизненный цикл чейнкода включает в себя несколько этапов. Перед запуском кода чейнкода, организации должны согласовывать основные параметры – название, версию и правила одобрения чейнкода. Члены канала приходят к соглашению в процессе, состоящем из следующих четырех шагов:

1. Installation (установка): на этом шаге исходный файл, содержащий чейнкод, пакуется в нужный формат SignedCDS и затем устанавливается на узлы сети, которые будут его запускать. Чейнкод необходимо установить на все одобряющие узлы (endorsing peers) в канале.

Команда установки чейнкода включает такие параметры, как имя чейнкода, его версию, путь до исходного кода, язык программирования:

```
# peer chaincode install -n my_chaincode -v 1.0 -p chaincode_source -l golang
```

2. Instantiation (создание экземпляра): это процесс связывания чейнкода с каналом. Чейнкод может быть привязан к любому количеству каналов и работать на каждом канале индивидуально и независимо. На данном этапе происходит проверка чейнкода на соответствие политике одобрения и инициализация начального состояния чейнкода.

Команда Instantiation включает те же параметры, что и Installation, и в дополнение массив аргументов для передачи в функцию инициализации Init, указание ordering service и политику одобрения:

```
# peer chaincode instantiate -n my_chaincode -v 1.0 -C my_channel -c '{"Args":["arg0","arg1"]}' -o orderer.example.com:7050 -P "OR('Org1MSP.member','Org2MSP.member')"
```

3. Invoke and Query: две команды для запуска выполнения функций чейнкода. Команда Invoke вызывает указанный чейнкод и фиксирует подтвержденную транзакцию в сети. После успешной отправки транзакции происходит обновление блокчейна. Команда Query возвращает подтвержденный результат вызова функции чейнкода, но не генерирует транзакцию. При этом, никакие изменения или информация о вызове чейнкода не будут отражены в блокчейне. Пример вызова данных команд, где в качестве функции чейнкода используется аргумент «func» с входными параметрами «arg»:

```
# peer chaincode invoke/query -n my_chaincode -my_channel -c '{"Args":["func","arg"]}' -o orderer.example.com:7050
```

4. Upgrade: используется в ситуации, когда в работающем чейнкоде обнаружена ошибка или необходимо обновить политику подтверждения, потому что к сети присоединился новый участник. Новая версия чейнкода устанавливается на требуемых endorsing peers, меняется только версия чейнкода, имя остается прежним. Команда вызова похожа на вызов Instantiate и включает аргументы инициализации, ordering service и политики одобрения:

```
# peer chaincode upgrade -n my_chaincode -v 1.1 -C my_channel
-c '{"Args":["arg0","arg1"]}' -o orderer.example.com:7050 -P
"OR('Org1MSP.member','Org2MSP.member')"
```

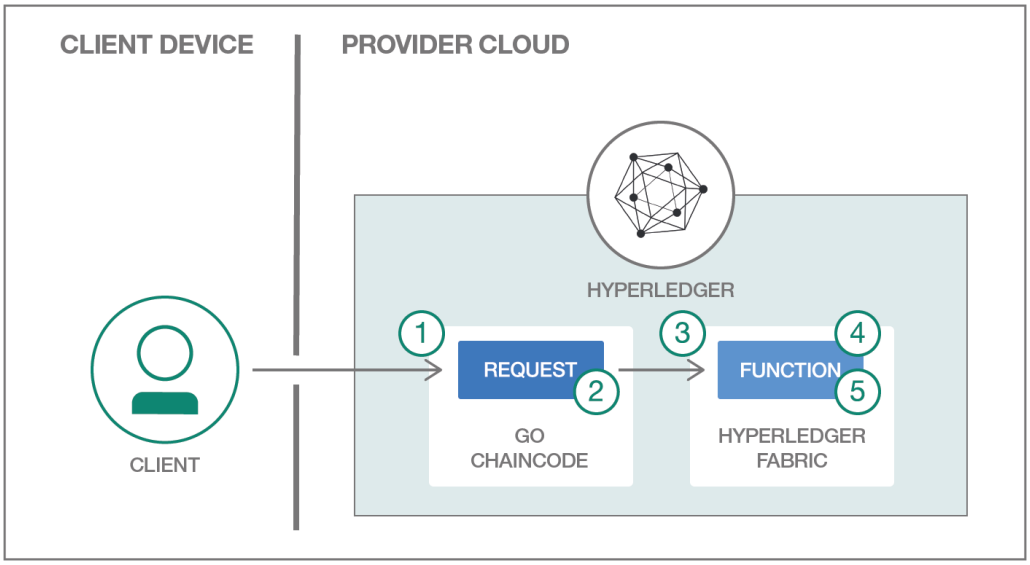

Рисунок 1.4 – Схема работы чейнкода в HLF [9]

Таким образом, в данном разделе выпускной квалификационной работы были описаны основные принципы построения децентрализованных сетей, рассмотрены особенности архитектуры блокчейн на базе Hyperledger Fabric и разобран жизненный цикл чейнкода. Для построения решения обработки персональных данных в доверенной среде была выбрана именно платформа HLF, так как она имеет гибкую структуру. Большой набор возможностей блокчейн сети на базе HLF включает реализацию смарт-контрактов на прикладных языках программирования, такие как Go и JavaScript.

2 ОПИСАНИЕ МЕХАНИЗМА ДОВЕРЕННОЙ СРЕДЫ ИСПОЛНЕНИЯ

Trusted Execution Environment (TEE) – доверенная среда исполнения, безопасная зона внутри основного процессора, которая работает параллельно с операционной системой в изолированной среде. Операционные системы предоставляют защиту от внедрения других процессов, но уязвимые места остаются всегда, так приложения не защищены от процессов с более высоким уровнем прав или от самой ОС [11]. Решение с TEE предоставляет высокоуровневую защиту секретов от атак со стороны ОС в том числе.

2.1 Intel Software Guard Extensions

Компания Intel в 2015 году представила свою технологию с использованием TEE – Intel Software Guard Extensions (Intel SGX).

Intel SGX – это набор инструкций центрального процессора, предоставляющий возможность создавать анклавы. Анклавы – области в виртуальном адресном пространстве, защищенные от чтения и записи другими процессорами (включая ядро ОС) извне этой области [3].

Intel SGX создавалась для безопасных удаленных вычислений, т.е. для запуска программного обеспечения на удаленном компьютере потенциально ненадежной стороны, при этом с гарантиями целостности и конфиденциальности информации.

Intel SGX обеспечивает следующие средства защиты от аппаратных и программных атак [11]:

– чтение из анклава и запись в анклав запрещены извне, независимо от уровня привилегий и режима центрального процессора;

– отладка эксплуатируемых анклавов недоступна, но разработчик может установить флаг отладки во время процесса разработки;

– общение с анклавом происходит с помощью специальных функций ECALL (Enclave Call) и OCALL (Outside Call). Никакие другие системные вызовы и операции с регистрами не связывают основную ОС с анклавом;

– память анклава зашифрована с использованием алгоритмов шифрования с открытым ключом;

– ключи шифрования меняются случайным образом при включении питания или перезагрузки системы. Ключи хранятся в ЦП в изолированном хранилище (KeyStorage);

– доступ к данным, изолированным внутри анклавов, может быть получен только с помощью кода, который использует этот анклав.

Пользователь может доверять производителю аппаратного обеспечения на удаленном компьютере и передавать свои данные в защищенный контейнер, размещенный на доверенном оборудовании (рисунок 2.1) [12].

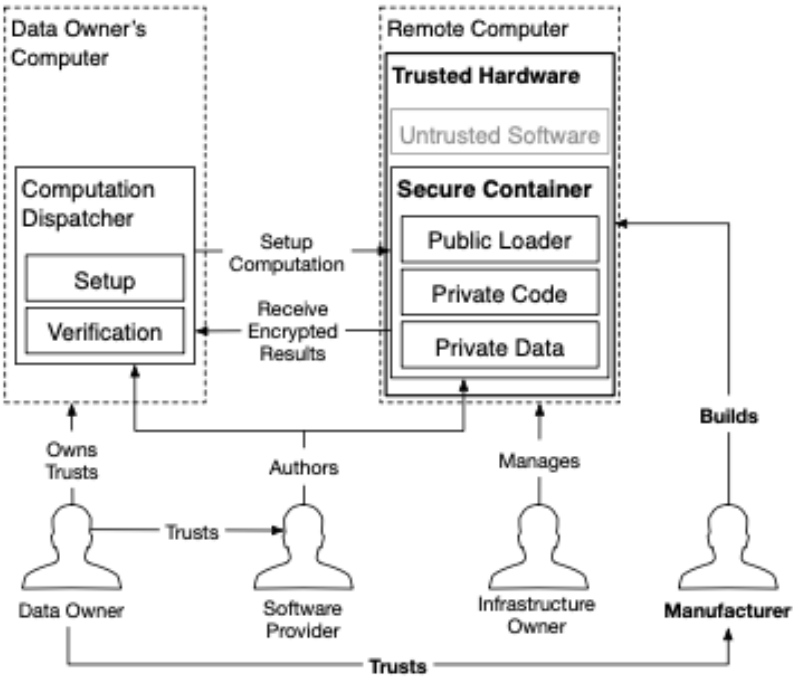

Рисунок 2.1 – Общая схема обработки данных на удаленном компьютере [12]

Данная схема может приняться в самых разных областях. Так, например, в медицинской сфере может быть реализован облачный сервис, который выполняет обработку конфиденциальных медицинских изображений для поиска потенциальных болезней у пациентов [12]. Изображения в зашифрованном виде передаются на удаленный компьютер с Intel SGX. Анклав содержит код для расшифровки изображений, алгоритм обработки изображений и код для

шифрования результатов. За пределами анклава выполняется код, который получает загруженные зашифрованные изображения и хранит их (Untrusted Part).

Процессор с поддержкой Intel SGX защищает целостность и конфиденциальность вычислений внутри анклава, изолируя код и данные анклава от внешней среды, включая операционную систему и гипервизор, а также аппаратные устройства, подключенные к системной шине. В то же время модель SGX остается совместимой с традиционными уровнями программного обеспечения в архитектуре Intel, где ядро ОС и гипервизор управляют ресурсами компьютера.

Большинство компьютеров работают на процессорах Intel и AMD. В Intel реализованы разные уровни привилегий – кольца привилегий (рисунок 2.2).

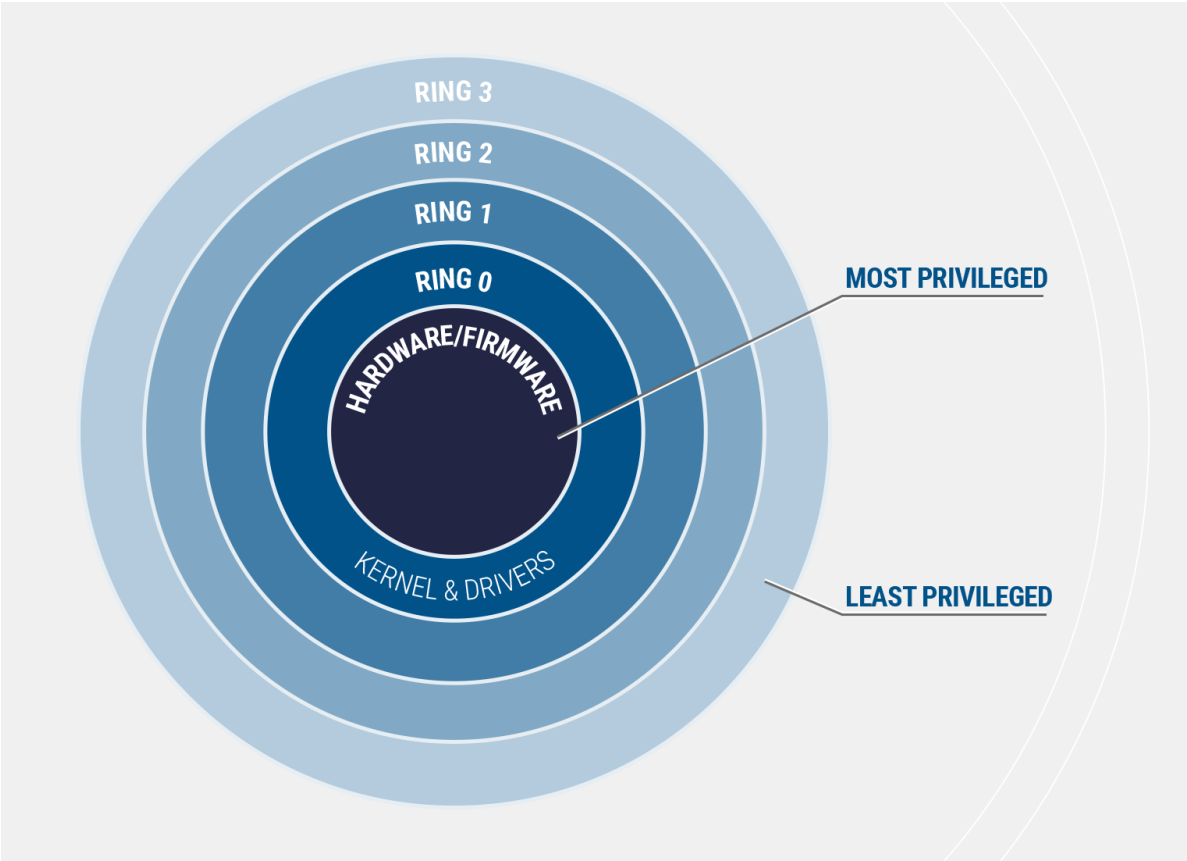

Рисунок 2.2 – Кольца привилегий [13]

Привилегии определяют разрешенные компьютерные операции и доступ к памяти в любой момент времени и используются для обеспечения безопасности в компьютерной системе. Значение 0 соответствует самому высокому уровню привилегий, а 3 самому низкому [13].

– кольцо 0: обычно ядро операционной системы и предназначено для выполнения наиболее критичных программ;

– кольцо 1: драйвера устройств;

– кольцо 2: операционные системы;

– кольцо 3: пользовательские приложения.

Использование всех четырех уровней привилегий не является необходимым. Также возможно добавление новых колец привилегий для ограничения полномочий компонентов иерархии, но усложнение архитектуры обычно не делает систему более защищенной, а наоборот, чем сложнее система, тем больше вероятность наличия в ней уязвимости. Предложенный механизм работы с анклавами помогает создавать безопасные приложения с минимальными затратами, не меняя архитектуры системы.

2.2 Особенности приложений для анклавов Intel SGX

С широким распространением программного обеспечения все более значимым стал вопрос безопасности для всей системы. Появление вредоносного программного обеспечения повлекло за собой создание критериев оценки безопасности системы. Для удобства оценки используется концепция Trusted Computing Base (TCB). Именно TCB определяет компьютерную систему как доверенную. TCB включает оборудование, прошивку и программное обеспечение, критически важное для защиты, и оно должно быть спроектировано и реализовано таким образом, что не полагаться на доверие элементам системы, исключенных из нее. Идентификация интерфейса и элементов TCB вместе с их правильной функциональностью, таким образом, формирует основу для оценки безопасности системы [14].

Технология Intel SGX позволяет минимизировать TCB и тем самым уменьшает поверхность атаки (attack surface) для злоумышленников (рисунок 2.3). Так, поверхность атаки до применения Intel SGX включала уровни приложения, ОС и VMM, а с использованием анклавов уменьшилась до области самого анклава.

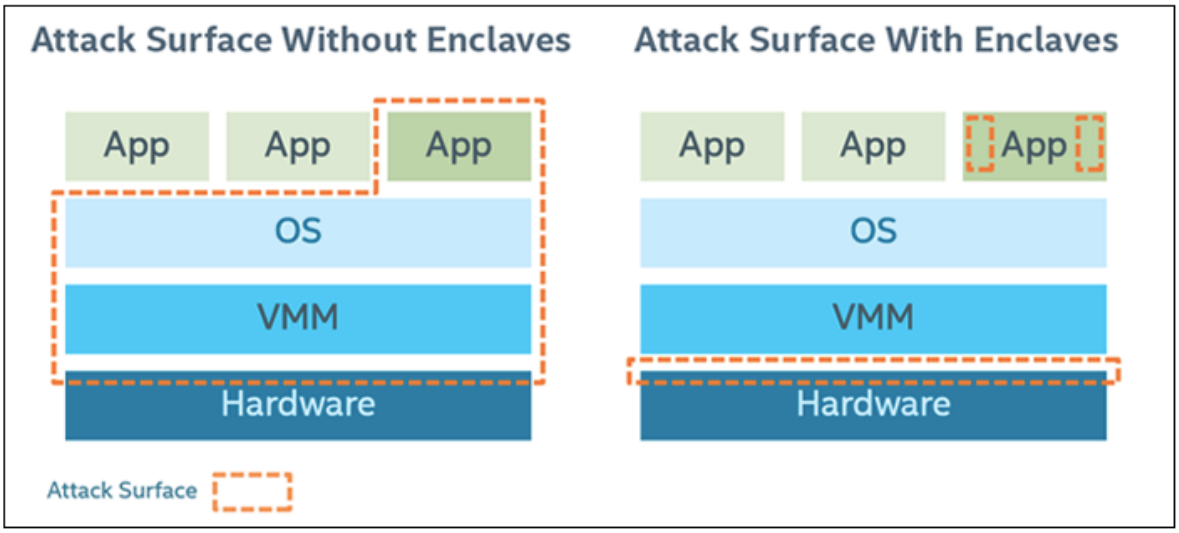

Рисунок 2.3 – Изменение TCB до внедрения Intel SGX и после [3]

Существенным ограничением на использование анклавов в Intel SGX является размер защищаемой памяти, который устанавливает BIOS. Стандартные размеры памяти анклавов – 64 и 128 Мб. А это означает, что код в анклаве должен быть небольшим и занимать мало памяти. Данная проблемы решается запуском нескольких анклавов параллельно, в зависимости от системы, в ней может функционировать от 5 до 20 анклавов.

Intel SGX полагается на аттестацию программного обеспечения. Аттестация доказывает пользователю, что он взаимодействует с определенной частью программного обеспечения, работающей в защищенном контейнере, размещенном на доверенном оборудовании. Доказательством является криптографическая подпись, которая удостоверяет хеш содержимого защищенного контейнера. Отсюда следует, что владелец удаленного компьютера может загрузить любое программное обеспечение в защищенный контейнер, но пользователь службы удаленных вычислений откажется загружать свои данные в защищенный контейнер, хеш содержимого которого не соответствует ожидаемому значению (рисунок 2.4) [12].

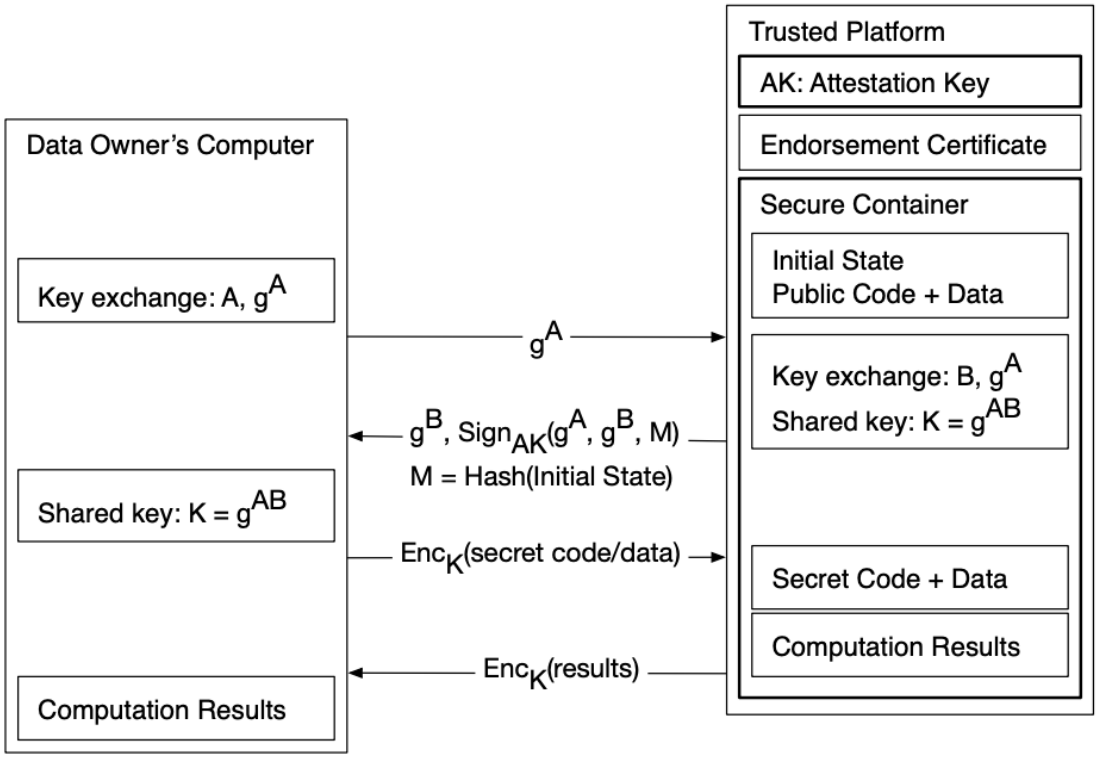

Рисунок 2.4 – Проверка подлинности защищенного контейнера [12]

Как видно на рисунке 2.4, для создания общего секретного ключа используется протокол обмена ключами Диффи-Хеллмана. Порядок работы протокола для двух участников Alice и Bob:

1. Alice и Bob генерируют случайные параметры a и b.

a, b – натуральные числа порядка 10^{100}.

2. Обе стороны устанавливают открытые общие параметры p и g. p – случайное простое число порядка 10^{300}, (p-1)/2 – также случайное простое, g – первообразный корень по модулю p ($g^{\varphi(p)} \equiv 1(mod\ p)$).

3. Alice и Bob вычисляют и обмениваются отрытыми ключами A и B. $A \equiv g^a(mod\ p)$, $B \equiv g^b(mod\ p)$.

4. На основании открытого ключа второй стороны и своего закрытого ключа, обе стороны вычисляют общий секретный ключ K. Для Alice $K \equiv B^a(mod\ p)$, для Bob $K \equiv A^b(mod\ p)$.

K для обеих сторон одинаковый, так как $B^a(mod\ p) = g^b(mod\ p)^a\ mod\ p = g^{ab}(mod\ p) = g^a(mod\ p)^b\ mod\ p = A^b(mod\ p)$

Таким образом, обе стороны имеют общий ключ K, который используется в качестве секретного ключа для шифрования передаваемых данных между Data Owner's Computer и доверенной платформой. Конструкция. Криптографическая стойкость данного алгоритма основана на предполагаемой сложности задачи дискретного логарифмирования, то есть задача сводится к вычислению $K = g^{ab}(mod\ p)$ по известным открытым параметрам $A \equiv g^{a}(mod\ p)$, $B \equiv g^{b}(mod\ p)$ [15].

Протоколы совместной выработки ключа, такие как Диффи-Хеллмана, требуют наличия канала связи с гарантиями целостности. Если злоумышленник может вмешаться в сообщения, передаваемые между Alice и Bob, то он может выполнить атаку «человек посередине» (MITM), как показано на рисунке 2.5.

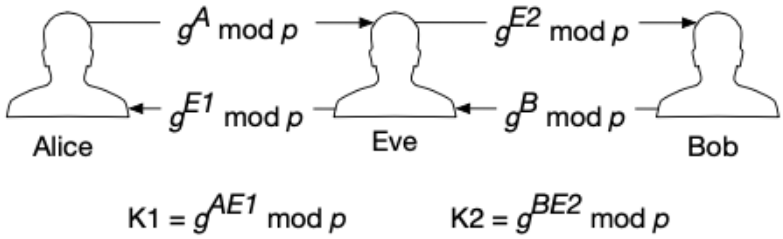

Рисунок 2.5 – Атака man-in-the-middle на протокол Диффи-Хеллмана [12]

В MITM-атаке Eve перехватывает первое сообщение Alice об обмене ключами и отправляет Bob свое собственное сообщение. Затем Eve перехватывает ответ от Bob и заменяет его своим собственным и отправляет Alice. Eve эффективно осуществляет обмен ключами как с Alice, так и с Bob, устанавливая общий секрет с каждым из них, причем ни Alice, ни Bob могут не догадываться о присутствии злоумышленника в канале связи [12]. После процесса установления ключей, Eve может перехватить зашифрованные сообщения от Alice, расшифровывать их ключом K1, посмотреть содержимое, затем зашифровывать уже на ключе K2 (установленный для Bob) и передать дальше Bob. До того, как Bob еще не получил сообщение от Alice, Eve смогла увидеть его содержимое. Более того, Eve может выдавать себя за любую из сторон в общении.

Атаки MITM на протоколы совместной выработки ключа могут быть предотвращены путем аутентификации стороны, которая отправляет последнее сообщение в протоколе (в наших примерах, Bob). При наличии центра сертификации (Certificate Authority), Bob использует свой открытый ключ для подписи сообщений в процессе совместной выработки ключа, а также отправляет Alice свой сертификат вместе с сертификатами для любых промежуточных CA. Alice проверяет сертификат Bob, это гарантирует, что субъект, указанный в сертификате, является тем, кого ожидают, и проверяет, что сообщения в процессе выработки ключа соответствуют представленной подписи.

Приложение, использующие Intel SGX состоят из двух частей: доверенная (trusted) и ненадежная (untrusted) часть.

Доверенный компонент – это анклав. Код в надежной части обращается к секретам приложения. Приложение может иметь более одного доверенного анклава.

Ненадежный компонент – это остальная часть приложения и любые его модули. С точки зрения анклава, ОС и VMM считаются ненадежными компонентами.

Доверенный компонент должен ограничиваться данными, которые нуждаются в максимальной защите, и теми операциями, которые должны действовать непосредственно на них. Анклав больших размеров потребует больше памяти и увеличит поверхность атаки.

Анклавы должны иметь минимальное взаимодействие между доверенными и ненадежными компонентами [3]. Хотя анклавы могут покидать защищенную область памяти и вызывать функции в ненадежном компоненте (с помощью специальной инструкции), ограничение этих зависимостей укрепит анклав от атак.

Процесс выполнения приложения Intel SGX представлен на рисунке 2.6

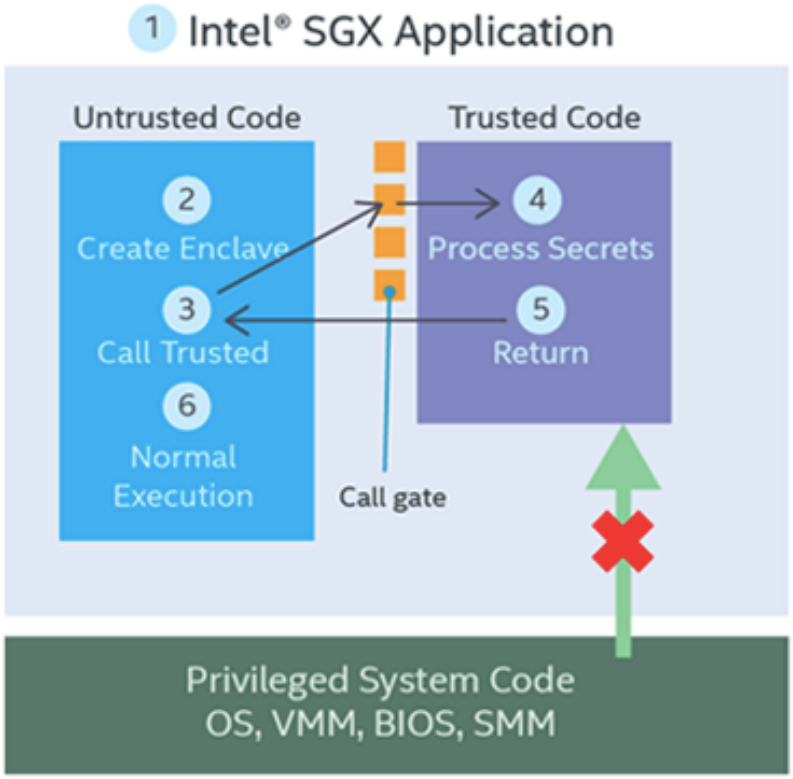

Рисунок 2.6 – Процесс выполнения приложения Intel SGX [3]

При выполнении приложения с Intel SGX можно выделить следующие этапы:

1. Разделение приложения на два компонента: доверенный и ненадежный.
2. Создание анклава в защищенном компоненте.
3. Передача выполнения коду анклава с помощью вызова доверенной функции из ненадежной части приложения.
4. Код анклава обрабатывает данные в открытом виде, при этом доступ к ним с привилегированного уровня ОС запрещен.
5. Передача результатов работы кода анклава в ненадежную часть приложения.
6. Исполнение в обычном режиме в ненадежном компоненте.

Intel SGX использует память под названием Processor Reserved Memory (PRM) для хранения данных анклава. Центральный процессор запрещает доступ к ней извне, даже со стороны гипервизора и ядра. PRM содержит Enclave Page Cache (EPC), который состоит из блоков страниц, каждый из которых занимает

4 Кбайт, при этом одна страница принадлежит одному анклаву. Состояние страниц контролируется центральным процессором и записывается в Enclave Page Cache Metadata (EPCM). Каждый анклав имеет свой собственный SECS (SGX Enclave Control Structure), один или несколько TCS (Thread Control Structure) и соответствующие SSA (Save State Area). SECS содержит метаданные анклава, такие как хеш и размер. TCS содержит точку входа в анклав. Для TCS существует своя SSA, содержащая состояние процессора во время выполнения анклава. У EPC в целом есть SIGSTRUCT и VA Page, которые содержат удостоверение подлинности анклава и аттестации (рисунок 2.7) [16].

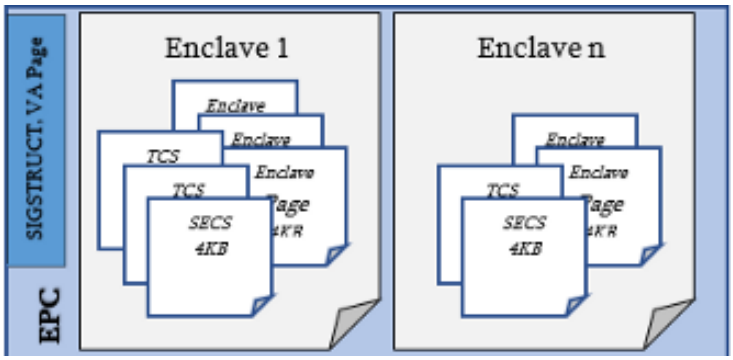

Рисунок 2.7 – Структура памяти анклава [16]

Безопасность Enclave Page Cache обеспечивает Memory Encryption Engine (MEE), здесь генерируются ключи шифрования, которые потом хранятся в процессоре. Данные ключи используется для расшифрования страниц внутри физического ядра процессора [17].

2.3 Аттестация кода анклава

Аттестация кода анклава необходима для повышения доверия к исполняемому коду в доверенной части приложения Intel SGX. Без аттестации невозможно создание анклава. В Intel SGX предусмотрено 2 вида аттестации:

1. Локальная: два анклава на одной платформе выполняют взаимную проверку подлинности.
2. Удаленная: подлинность анклава проверяется удаленной стороной.

Локальная аттестация используется при наличии двух и более анклавов на устройстве. Для проверки подлинности один анклав выполняет инструкцию

EREPORT. Данная инструкция создает REPORT – отчет об аттестации, связывая сообщение анклава с хешем анклава и сертификатом. В данном случае может возникнуть ситуация, когда двум анклавам необходимо использовать общие данный. Для этого они устанавливают общий сеансовый ключ по протоколу Диффи-Хеллмана и используют его для шифрования передаваемых сообщений.

Удаленная аттестация происходит в несколько этапа [18]: процессор с Intel SGX вычисляет хеш кода и данных анклава. Затем ПО внутри анклава вычисляет подпись аттестации анклава (хеш анклава и сообщение анклава). Подпись аттестации выполняется с помощью Quoting Enclave, который имеет доступ к уникальному ключу аттестации. Данный ключ выдается с помощью Provisioning Enclave. Если анклав создан верно и работает на подлинном процессоре, удаленная сторона будет доверять ему и передавать данные по доверенному каналу (рисунок 2.8).

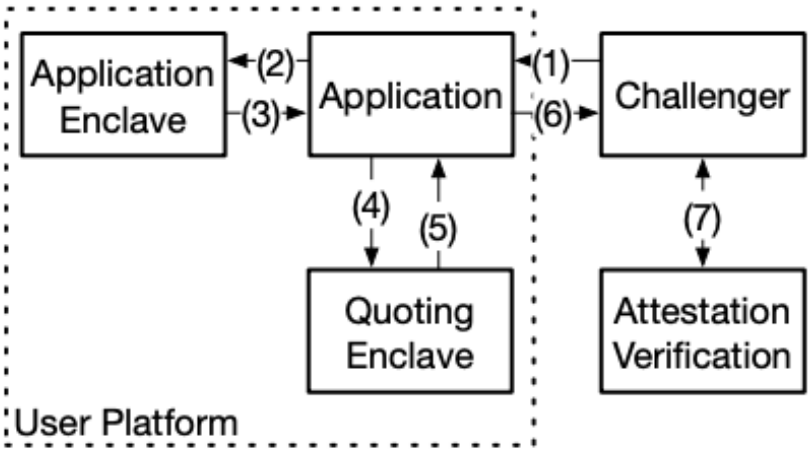

Рисунок 2.8 – Удаленная аттестация анклава [18]

Существует несколько моделей аттестации анклава Intel SGX.

Первоначальные версии Intel SGX были ориентированы на клиентские платформы, чувствительные к конфиденциальности. Протокол Enhanced Privacy ID (EPID) позволяет идентифицировать системы как подлинные платформы SGX без раскрытия их личности в процессе. Это важно для случаев использования клиентских платформ, когда поставщик услуг хочет гарантировать доступность

SGX, но конечный пользователь хочет сохранить конфиденциальность при аттестациях.

При аттестации EPID ключ аттестации платформы устанавливается посредством протокола слепого соединения, выполняемого между платформой и серверной службой Intel. В результате платформа имеет ключ закрытой группы, ключ аттестации, с помощью которого можно подписывать аттестации. Любой, у кого есть доступ к общему ключу группы, может проверить, была ли подпись сгенерирована одним из многих закрытых ключей группы, не узнавая, какой именно закрытый ключ группы сгенерировал подпись.

Вместо того, чтобы публиковать открытые ключи группы, Intel создала службу аттестации (Intel Attestation Service - IAS) для проверки аттестаций. В рамках модели удаленной аттестации EPID проверяющий отправляет аттестацию в службу аттестации IAS, которая отвечает отчетом проверки аттестации, подтверждающим или опровергающим подлинность аттестации и анклава, из которого она исходит. Верификатор проверяет отчет о подтверждении аттестации, чтобы определить надежность анклава и платформы. Ответ от IAS включает исходную аттестацию и, таким образом, может быть проверен третьими сторонами, кроме первоначального претендента.

2.4 Атаки на память Intel SGX

Несмотря на предусмотреные механизмы безопасности через аттестацию и изоляцию памяти анклава от основной памяти, Intel SGX подвержен ряду атак.

Атака Prime+Probe [20].

Процессор Intel имеет трехуровневую иерархию кэша:

– уровень L3: самый большой и самый медленный. Распределяется между всеми ядрами процессора;

– уровень L2 и L1: каждое ядро ЦП имеет выделенный кэш L1 и L2, но они используются совместно исполнительными модулями ядра с одновременной многопоточностью. В L1 и L2 память кода (code memory) и память данных (data memory) конкурируют за доступное пространство кэша;

– L1 разделяется на кэш данных (data cache L1D) и кэш инструкций (instruction cache L1I).

Атаки по сторонним каналам (Side-channel attacks) – тип атак, использующий мониторинг доступа к памяти и кэшу анклава. Prime and Probe является атакой по сторонним каналам (рисунок 2.9).

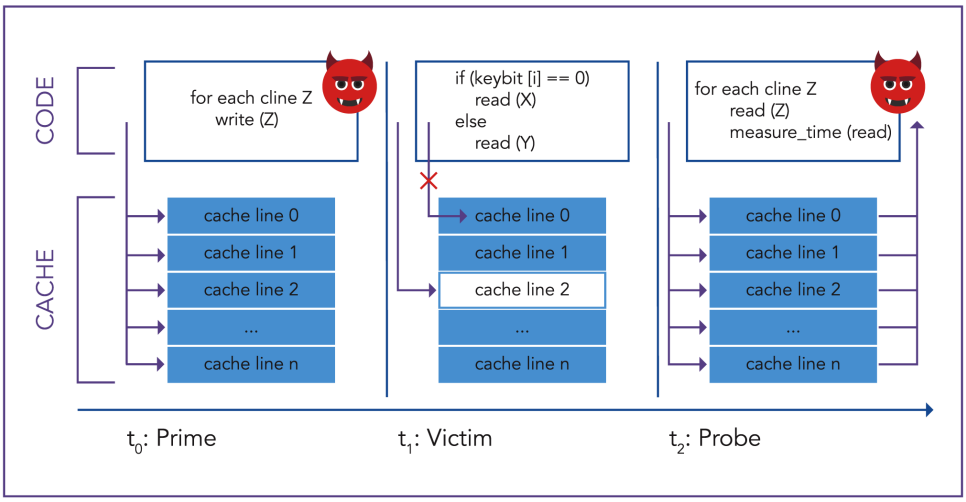

Рисунок 2.9 – Схема атаки Prime and Probe

Порядок проведения атаки [21]:

1. t_0: Злоумышленник получает доступ к памяти из ненадежной части приложения и заполняет весь кэш данных процесса жертвы.

2. t_1: Жертва обращается к памяти, в которой хранятся конфиденциальные данные, такие как криптографические ключи. cache line выбирается по значению keybit. Данные, хранящиеся в X, загружаются в кэш, вытесняя данные, которые были там до этого.

3. t_2: Злоумышленник проверяет, какие из его строк кэша были вытеснены – строки, в которые жертва делала записи. Для определения таких строк проводится измерение времени доступа. Повторяя эту операцию для каждого из keybit, злоумышленник получает весь ключ.

В Intel SGX есть защита от атаки по сторонним каналам, которая запрещает мониторинг событий, связанных с кэшем, но атака Prime and Probe все равно

работает, так как злоумышленник наблюдает именно за событиями кэша своего процесса и разделяет кэш с жертвой.

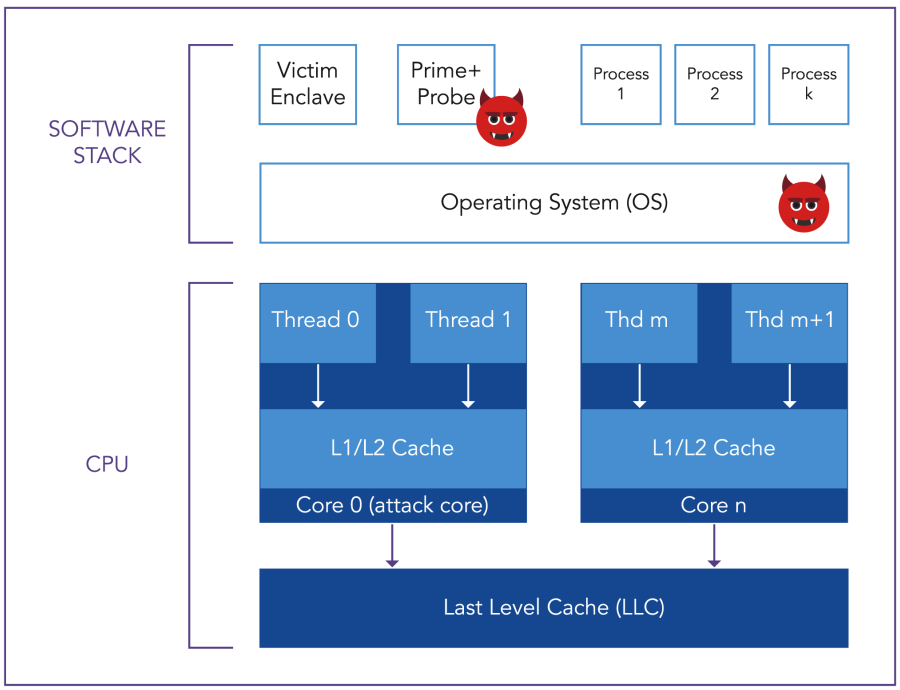

Рисунок 2.10 – Защита от атаки Prime and Probe

Известны также атаки типа Spectre и Foreshadow (L1TF). Они позволяют производить чтение данных из кэш-памяти через сторонний канал. Предусмотрена защита от уязвимости Spectre-v2, работающая против двух данных атак.

В июне 2020 года опубликованы две новые атаки на память анклава Intel SGX: SGAxe и CrossTalk [22]. Данные атаки позволяют получить доступ к конфиденциальным данным из анклава.

Атака SGAxe основана на атаке CacheOut, где злоумышленник вытесняет данные из кэша. В SGAxe используется усовершенствованный подход, а именно, схему разбиения памяти на страницы (paging), тем самым перемещая данные анклава в кэш L1, там они расшифровываются и перемещаются в буфер, где извлекаются с помощью метода RIDL, который анализирует буферы заполнения строки (LFB) и порты загрузки. Целью такой атаки являются ключи «аттестации», с помощью которых может быть расшифрованы конфиденциальные данные.

Атака CrossTalk на анклав основана на ранее неизвестном буфере, который используют все процессорные ядра Intel. Его назвали «промежуточный буфер» (staging buffer), он сохраняет результаты ранее выполненных команд во всех ядрах процессора и выходные данные от RDRAND и RDSEED – эти инструкции обеспечивают случайные числа для генерации криптоключей (рисунок 2.11).

Получив случайные числа от RDRAND и RDSEED, злоумышленники могут использовать их для вывода ключа, таким образом, CrossTalk атака извлекает ключ, основанный на криптографическом алгоритме ECDSA из анклава SGX. Данная атака ставит под сомнение эффективность механизма спекулятивного исполнения, когда происходит разделение приложения на доверенный и ненадежный код между разными ядрами процессора.

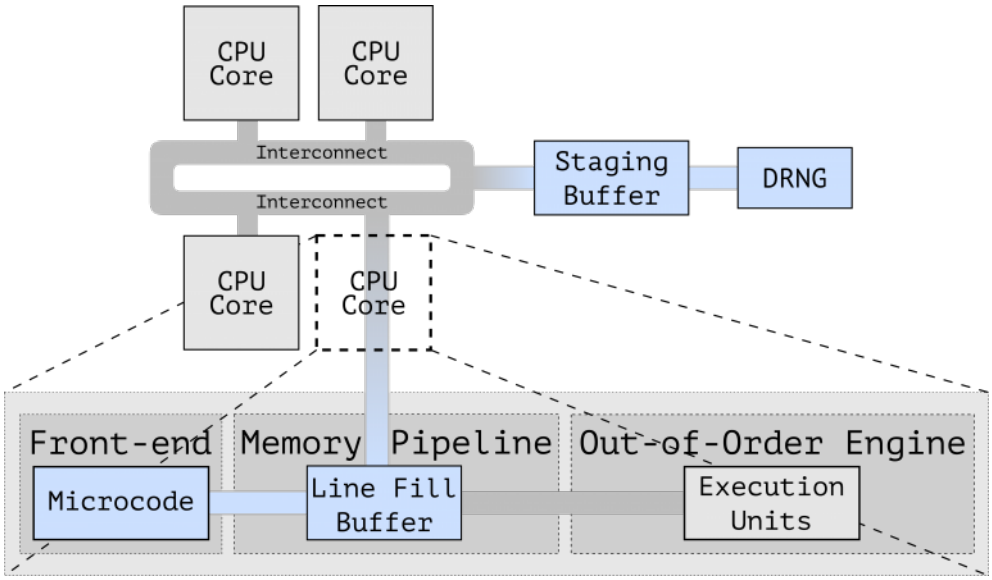

Рисунок 2.11 – Схема атаки CrossTalk [22]

Список процессоров Intel, подверженных атакам CrossTalk представлен на рисунке 2.12.

CPU	Year	Microcode	Staging Buffer Present	Supports SMT	Vulnerable to Cross-Core Attacks
Intel Xeon Scalable 4214 (Cascade Lake)	2019	0x500002c	?	✓	✗
Intel Core i7-0850H (Coffee Lake)	2019	0xca	✓	✓	✓
Intel Core i7-8665U (Whiskey Lake)	2019	0xca	✓	✓	✓
Intel Xeon E-2288G (Coffee Lake)	2019	?	✓	✓	✓
Intel Core i9-9900K (Coffee Lake R)	2018	0xca	✓	✓	✓
Intel Core i7-7700K (Kaby Lake)	2017	0xca	✓	✓	✓
Intel Xeon E3-1220V6 (Kaby Lake)	2017	0xca	✓	✗	✓
Intel Core i7-6700K (Skylake)	2015	0xc2	✓	✓	✓
Intel Core i7-5775C (Broadwell)	2015	0x20	✓	✓	✓
Intel Xeon E3-1240V5 (Skylake)	2015	0xd6	✓	✓	✓

Рисунок 2.12 – Уязвимые к атаке CrossTalk процессоры Intel [22]

Компания Intel выпустила обновление микрокода, которое исправляет этот баг. Обновление блокирует всю шину памяти перед обновлением staging buffer (промежуточного буфера), а разблокирует её только после очистки содержимого. Intel применяет изменения только к особенно важным с точки зрения безопасности инструкциям, включая RDRAND, RDSEED. Данные другой инструкции можно все так же прочитать из буфера.

Таким образом, в данном разделе выпускной квалификационной работы был рассмотрен механизм доверенной среды исполнения на примере Intel SGX. Были описаны особенности реализации приложений для Intel SGX с применением анклавов, а также разобраны известные атаки на память анклавов. На данный момент, компания Intel успешно выпускает обновления для микрокода процессоров и устраняет найденные уязвимости, но несмотря на это, эксперты считают, что в скором времени будут выявлены новые эксплоиты. В создании доверенного оракула для обработки персональных данных была выбрана технология доверенной среды исполнения Intel SGX за счет самого подходящего механизма «слепой» обработки конфиденциальных данным для данной работы. Применение анклавов позволит передавать персональные данные пользователей на удаленный оракул с уверенность в безопасности этих данных.

3 ВЗАИМОДЕЙСТВИЕ ОРАКУЛА С СЕТЬЮ БЛОКЧЕЙН

Основу доверенного оракула для обработки персональных данных составляют два механизма, описанные в первой и второй главе данной выпускной квалификационной работы: децентрализованная сеть блокчейн и доверенная среда исполнения. Сочетание этих двух технологий позволит увеличить доверие к оракулу и передавать конфиденциальные данные компаний на удаленную обработку.

3.1 Пример с Hyperledger Avalon

Hyperledger Avalon (HLA) – проект альянса Enterprise Ethereum. HLA обеспечивает конфиденциальность транзакций блокчейна, используя доверенную среду исполнения Intel SGX. В случае HLA блокчейн используется для обеспечения соблюдения политик выполнения и обеспечения возможности аудита транзакций, в то время как связанные с ней доверенные вычислительные ресурсы вне сети выполняют транзакции. Используя доверенные вычислительные ресурсы вне сети, разработчик может увеличить пропускную способность и улучшить конфиденциальность данных. Сохранение целостности выполнения и обеспечение соблюдения гарантий конфиденциальности достигается за счет использования опции доверенных вычислений в Trusted Execution Environment.

Служба доверенных вычислений (TCS) размещает доверенных рабочих (Trusted Workers) и делает их доступными для выполнения рабочих заданий (Work Orders), отправленных запрашивающими через интерфейс пользователя или инструменты командной строки. Заказы на выполнение работ также могут быть отправлены с помощью смарт-контрактов (в зависимости от корпоративного приложения), работающих на DLT (рисунок 2.13).

Компоненты архитектуры Hyperledger Avalon [23]:

– worker registry: здесь перечислены доверенные вычислительные работники. Он включает в себя отчет о проверке аттестации, открытый ключ шифрования RSA и открытый ключ проверки ECDSA SECP256K1;

– workload: включает в себя сервис аттестации (IAS), Key Management Enclave, который имеет доступ к закрытым ключам подписи и шифрования работника;

– front-end application: внешнее приложение может быть пользовательским интерфейсом или сценарием, который используется для инициирования запросов на рабочие задания и получения ответов на них.

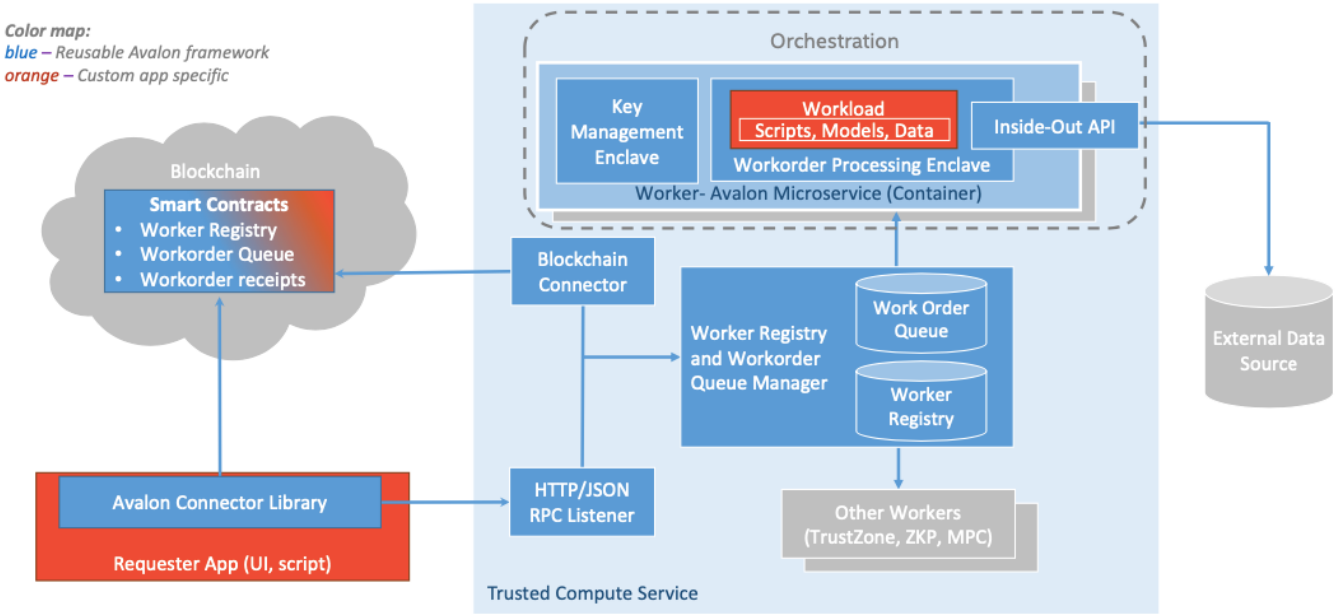

Рисунок 2.13 – Архитектура HLA [23]

Структура HLA требует множество компонентов для работы и таким образом, настройка сети с использованием HLA представляется довольно сложным процессом. В новой схеме предлагается более простое решение – реализация верификации кода в TEE с помощью смарт-контракта в Hyperledger Fabric перед каждым новым исполнением данного кода. Такой подход облегчает взаимодействие компонентов и обеспечивает более удобный аудит системы, при этом для бизнес-компаний решение остается автоматизированным, что сокращает экономические издержки и экономит время.

3.2 Общая схема передачи персональных данных между участниками сети

Общая схема включает в себя блокчейн сеть Hyperledger Fabric со всеми необходимыми компонентами. Для примера в схему добавлены 5 участников сети (Company A – Company E), каждая компания имеет копию смарт-контракта

для передачи персональных данных на удаленный оракул, Membership Service, сертификат подлинности, приватный/публичный ключ и персональные данные.

На первом этапе каждая компания вызывает функцию invoke чейнкода для записи персональных данных в смарт-контракт (рисунок 3.1).

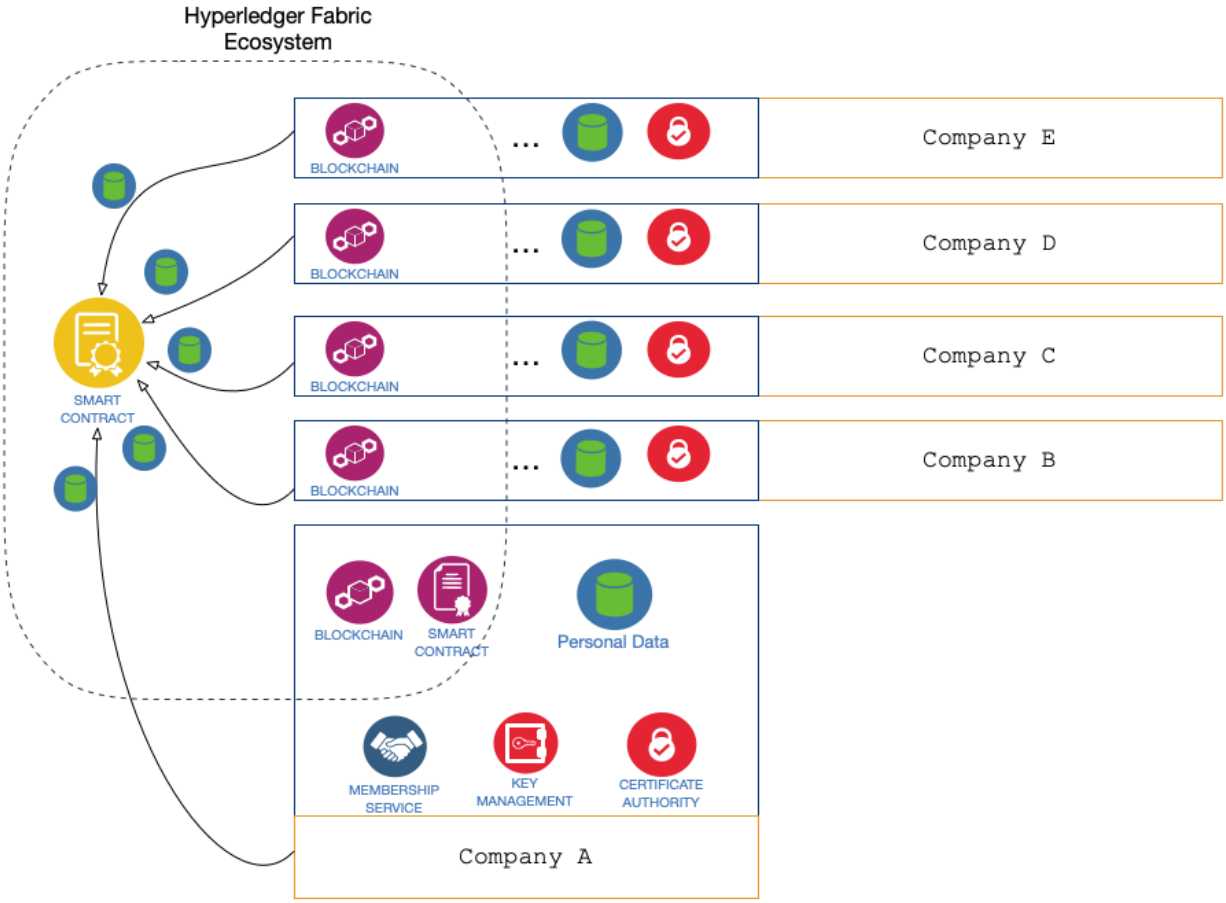

Рисунок 3.1 – I этап обработки персональных данных

На I этапе происходит добавление новые данных в смарт-контракт. При этом происходит проверка каждого peer на наличие сертификата и совпадение подписи с помощью MSP. Транзакция по добавлению данных записывается в блокчейн.

На втором этапе оракул создает запрос (request) для смарт-контракта, содержащий хеш кода анклава и открытый ключ шифрования, а также параметры для совместной выработки ключа. Смарт-контракт сравнивает хеш с заранее установленным значением. Если хеш совпадает, смарт-контракт зашифровывает персональные данные на открытом ключе и возвращает оракулу (3.2).

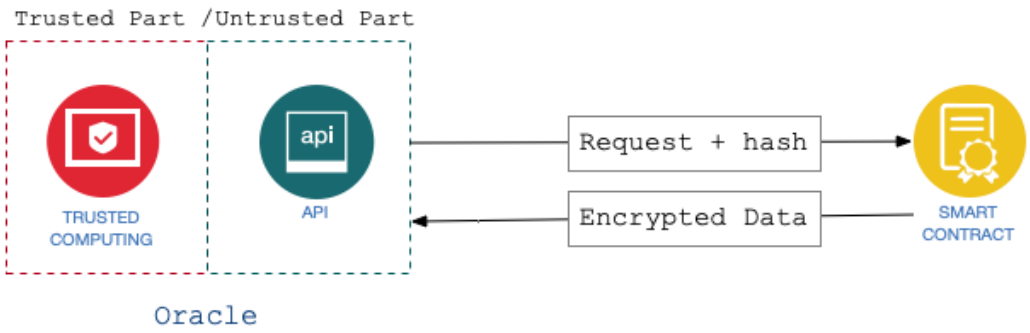

Рисунок 3.2 – II этап обработки персональных данных

На III этапе персональные данные расшифровываются в памяти анклава с помощью приватного ключа и обрабатываются в доверенной среде исполнения. Результаты обработки персональных данных зашифровываются на открытом ключе и передаются в смарт-контракт.

На IV этапе результаты обработки данных расшифровываются приватным ключом в смарт-контракте и передаются всем участникам сети, прописанным в контракте. У участников есть время ознакомиться с результатами и отправить оценку выполненного задания в смарт-контракт, тем самым формируется рейтинг оракула (рисунок 3.3). В случае утечки конфиденциальных данных, у участников есть возможность подать жалобу через смарт-контракт, предоставив доказательства. В этом случае проводится аудит оракула на наличие вмешательства злоумышленников и выплата компенсации.

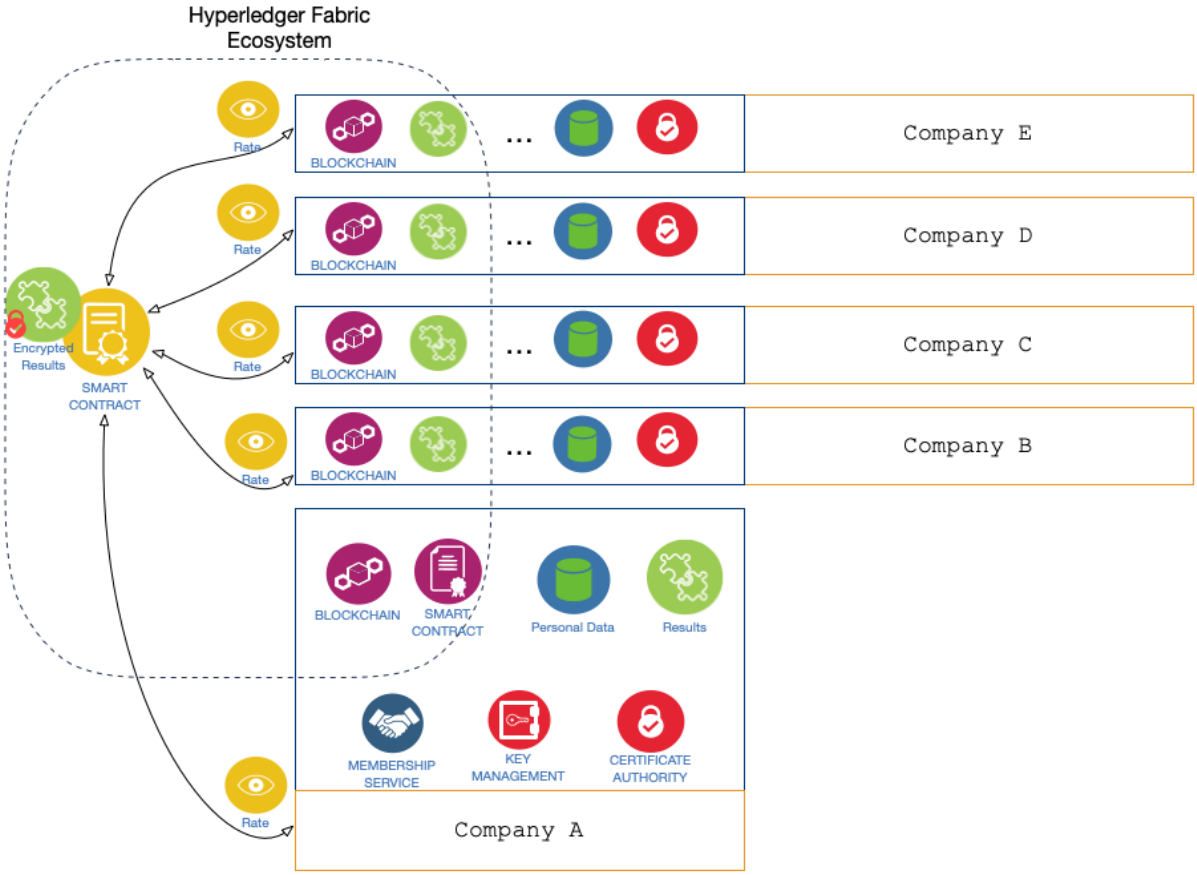

Рисунок 3.3 – IV этап обработки персональных данных

Диаграмма последовательности потока данных для трех участников (Oracle, Smart-Contract, Peer) представлена на рисунке 3.4. В данной диаграмме формально описаны все четыре этапа обработки персональных данных, кроме запроса с жалобой к стороне оракула при утечке конфиденциальных данных. Данный случай рассматривается как отдельная инструкция в смарт-контракте и может быть выполнена peer'ом в любой момент времени. Пример такой инструкции:

```
# peer chaincode invoke -n my_chaincode -my_channel -c '{"Args":["complaint","reasons.txt"]}' -o orderer.example.com:7050
```

В качестве аргумента функции «complaint» могут быть основания, описанные в виде доказательств утечки данных. Так как HLF корпоративное решение, возможно рассмотрение данного обращения с помощью администратора сети, аудит оракула и выплата компенсации.

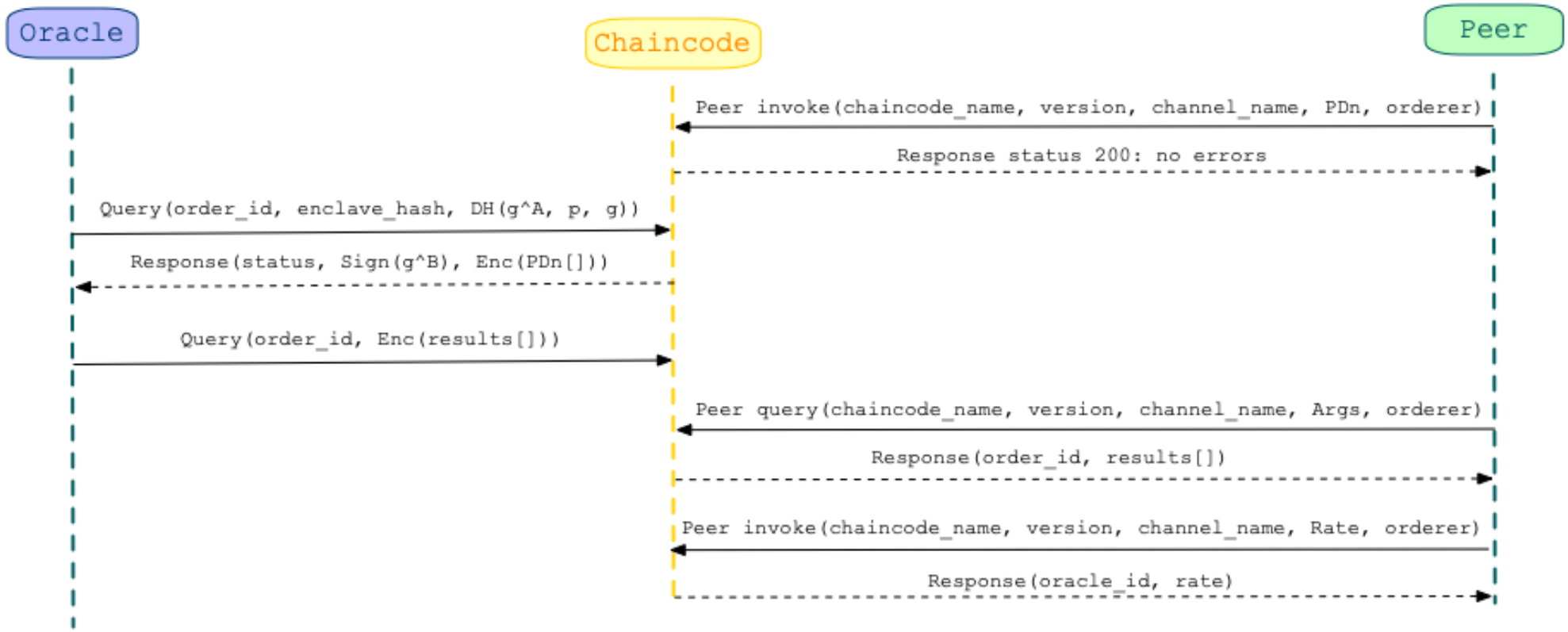

Рисунок 3.4 – Диаграмма последовательности потока данных

3.3 Структура чейнкода

Основной чейнкод сети включает в себя два смарт-контракта:

1. PDnSC: отвечает за принятие персональных данных от peers, сохранение их в формате JSON в базе данных в виде ключ-значение, установку общего секретного ключа с анклавом SGX, проверку хеша анклава, шифрование персональных данных на открытом ключе, отправку данных на оракул, расшифровку и запись результатов обработки в базу данных, из которой peers могут получить результаты в открытом виде.
2. RepSC: отвечает за репутацию (рейтинг) оракулов, принимает запросы с оценкой работы оракулов от peers, высчитывает общий рейтинг для каждого используемого оракула, принимает жалобы (complaint) от peers.

Перед началом работы с оракулом, стороны устанавливают хеш кода, который имеет сертификат от IAS и работает в анклаве в доверенной части оракула. Хеш кода записывается в смарт-контракте и хранится там до момента установления связи с оракулом, после этого сохраненный хеш сравнивается с полученным, и в случае совпадения значений, процесс продолжается, иначе возвращается ошибка.

Код смарт-контракта PDnSC представлен в приложении 1. Для работы был приведен пример оценки эффективности использования вакцины от Covid-19. Смарт-контракт реализован на языке Go и имеет следующую структуру:

Определение структуры с 5 полями для хранения в базе данных по ключу, указанном в структуре Records:

```
type Records struct {
     Fullname  string `json:"fullname"` // полное имя пациента.
     DOB       string `json:"dob"`// дата рождения.
     Insurance string `json:"insurance"` // номер страховки с указанием страховой фирмы.
     Vaccine   string `json:"vaccine"` // название используемой вакцины.
     Status    string `json:"status"`// состояние после применения вакцины.
}
```

Функция инициализации initLedger используется при установке чейнкода, принимает на вход контекст выполнения чейнкода и инициализирует базу данных начальными значениями:

```
func (s *SmartContract) initLedger(APIstub shim.ChaincodeStubInterface) sc.Response
```

Функция recordPD добавляет в базу данных новые записи от peers, принимает на вход контекст выполнения чейнкода и сами данные в виде массива строк:

```
func (s *SmartContract) recordPD(APIstub shim.ChaincodeStubInterface, args []string) sc.Response
```

Функция queryPD выводит записи из базы данных по ключу, принимает на вход контекст выполнения чейнкода и ключ, по которому находятся данные:

```
func (s *SmartContract) queryPD(APIstub shim.ChaincodeStubInterface, args []string) sc.Response
```

Функция changPD позволяет редактировать данные пользователей, заменяя поля в базе данных, на вход принимает контекст выполнения чейнкода, ключ, по которому находятся данные и новые данные для замены:

```
func (s *SmartContract) changePD(APIstub shim.ChaincodeStubInterface, args []string) sc.Response
```

Функция CheckHash сравнивает хеш со значением, хранящимся в базе данных, на вход принимает контекст выполнения чейнкода, идентификатор оракула и сравниваемый хеш:

```
func (s *SmartContract) checkHash(APIstub shim.ChaincodeStubInterface, args []string) sc.Response
```

Для шифрования подключен дополнительный модуль, из которого используются методы Decrypter для расшифрования и Encrypter для шифрования.

3.4 Репутационная модель в сети блокчейн

Для создания доверенного оракула необходима оценка доверия. Помимо создания смарт-контракта для сбора рейтинговых данных от участников сети, использовался индекс уровня доверия клиентов [24].

Методика измерения уровня доверия была предложена в 1994 году профессором Национального совета исследований Канады Стивеном П. Маршем. Индекс доверия рассчитывается по формуле:

$$Tx(y,a) = Ux(a) \times Ix(a) \times Tx^{C}(y)$$

$Ux(a)$ – «ценность ситуации a для субъекта доверия x». Этот параметр характеризует количество «полезности», то есть насколько данное действие было полезно клиенту в данной ситуации. $Ux(a)\epsilon[-1;1]$.

$Ix(a)-$ «важность ситуации a для субъекта доверия x», то есть важность самой ситуации взаимодействия.

– 0 – ситуация не важна;

– 1 – ситуация важна.

$Tx^{C}(y)-$ степень «основного доверия» субъекта x объекту доверия y на основе прошлого опыта взаимодействия. $Tx^{C}(y)\epsilon[-1;1]$.

$Tx(y,a)\epsilon[-1;1]$, где (-1) – абсолютно негативное отношение, (0) – нейтральное отношение и (+1) – позитивное отношение.

Таким образом, рейтинг оракула составляется на основе опроса peers.

Опросы могут быть следующие:

1. Оцените, насколько полезным является данный оракул для вашей компании:
   1) качество предоставленных результатов (1-5) – 3;
   2) скорость обработки данных (1-5) – 2;
   3) надежность ваших персональных данных (1-5) – 5.
2. Оцените, насколько для вас важны следующие критерии в работе с оракулом:
   1) качество предоставленных результатов (1-5) – 5;
   2) скорость обработки данных (1-5) – 1;
   3) надежность ваших персональных данных (1-5) – 3.
3. Оцените качество услуг во время последнего обращения к оракулу:
   1) качество предоставленных результатов (1-5) – 3;
   2) скорость обработки данных (1-5) – 1;

3) надежность ваших персональных данных (1-5) – 5.

Далее предложен расчет индекса доверия для ответов, данных на вопросы 1-3:

$$Ux(a) = \frac{0 + 0.5 + 1.0}{3} = 0.5$$

$$Ix(a) = 0$$

$$Tx^{C}(y) = 0.5$$

Расчет индекса производится по таблице 3.1 [24]:

Таблица 3.1 – Расчет индекса доверия

		$Ux(a)$				
		-1	-0,5	0	0,5	1
$Tx^{C}(y)$	-1	1_1	$0{,}5_2$	0_3	$0{,}5_4$	1_5
	-0,5	$0{,}5_6$	$0{,}25_7$	0_3	$-0{,}25_8$	$-0{,}5_9$
	0	0_3	0_3	0_3	0_3	0_3
	0,5	$-0{,}5_{10}$	$-0{,}25_{11}$	0_3	$0{,}25_{12}$	$0{,}5_{13}$
	1	-1_{14}	$-0{,}5_{15}$	0_3	$0{,}5_{16}$	1_{17}

$Tx(y,a) = Ux(a) \times Ix(a) \times Tx^{C}(y) = 12$ – позитивное отношение, принятие решения о дальнейшем сотрудничестве с компанией.

Смарт-контракт RepSC высчитывает рейтинг для каждого оракула, исходя из индекса доверия, полагаясь на результаты опросов клиентов.

3.5 Экспериментальные результаты

В ходе разработки доверенного оракула для обработки персональных данных была настроена тестовая блокчейн сеть Hyperledger Fabric, включающая 2 организации org1 и org2, с двумя peers в каждой.

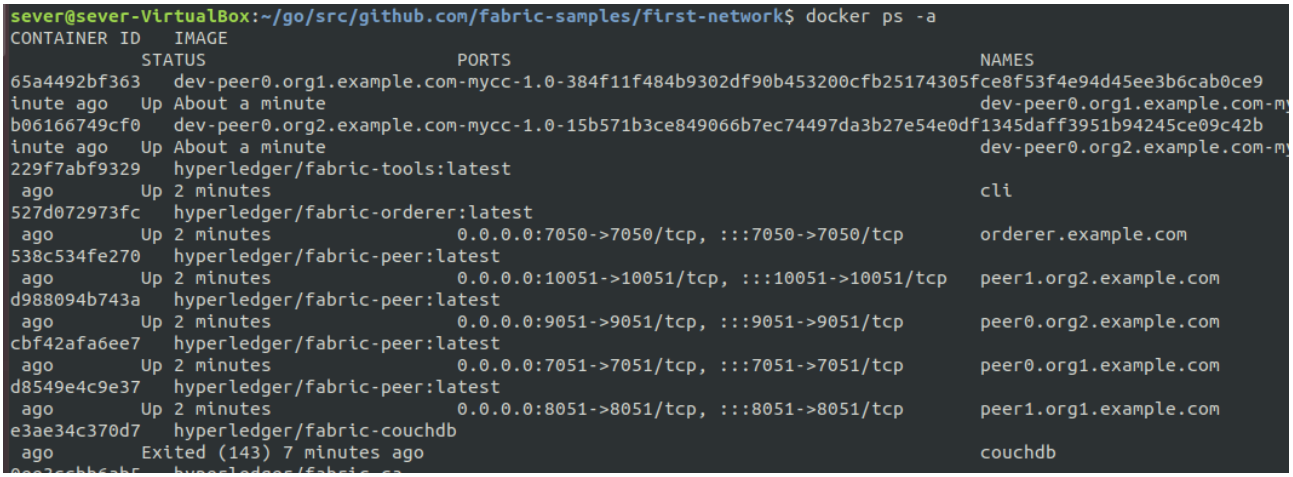

Рисунок 3.5 – Запущенные docker контейнеры с узлами сети

Далее был написан специальный shell-код (Приложение 2), который выполняет установку чейнкода в сеть HLF и инициирует базу данных записями с персональными данными клиентов.

Команда для установки чейнкода следующая:

```
sudo docker exec -it cli peer chaincode install -n mycc -v 1.0 -p "github.com/chaincode/"
```

Сообщение об успешной установки чейнкода выводится в консоль (рисунок 3.6):

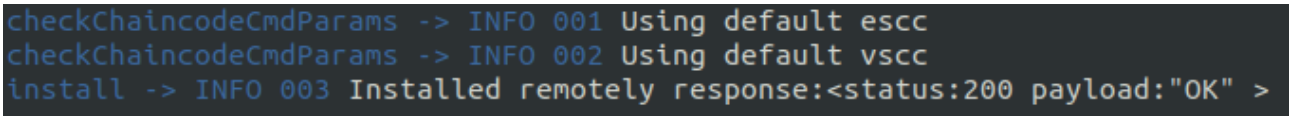

Рисунок 3.6 – Успешная установка чейнкода

Команда для инициализации чейнкода следующая:

```
sudo docker exec -it cli peer chaincode instantiate -o orderer.example.com:7050 -C mychannel -n mycc -v 1.0 -c '{"Args":[]}'
```

Результат выводится в консоль (рисунок 3.7):

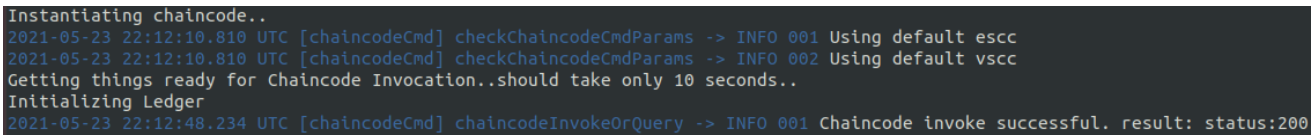

Рисунок 3.7 – Успешная инициализация чейнкода

Используя инструкцию query можем просмотреть, какие записи хранятся на данный момент в чейнкоде (рисунок 3.8).

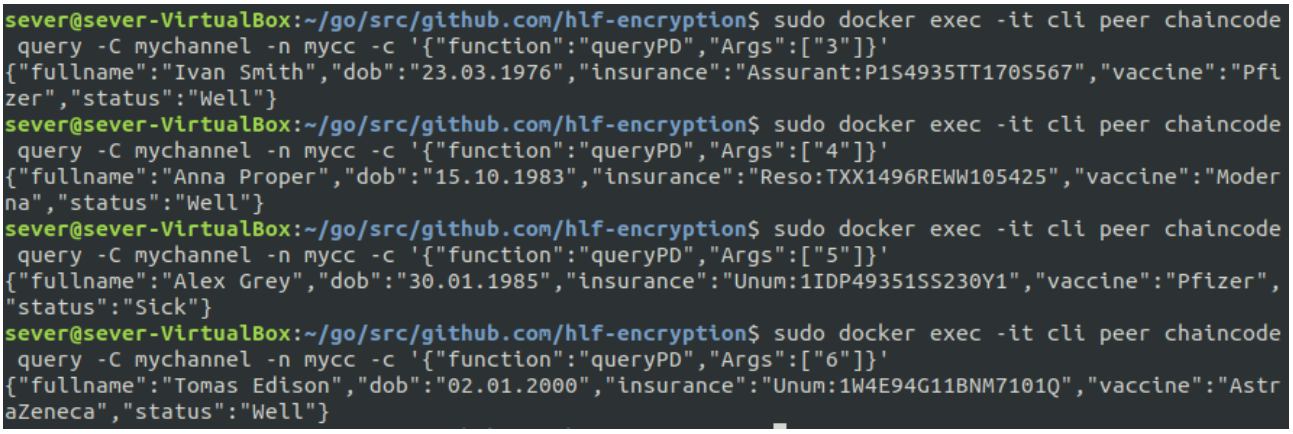

Рисунок 3.8 – Просмотр записей после инициализации

Как видно на рисунке 3.8, записи содержат такие конфиденциальные данные о клиентах, как ФИО, дата рождения, номер страховки.

Далее на оракуле в BIOS был активирован Intel SGX, виртуальная машина позволяет поддерживать выполнение кода в доверенной среде исполнения в режиме симулятора. После этого был написан простой код на Си, с функцией main, принимающей в качестве аргументов массив строк с персональными данными и возвращающий сообщение в зашифрованном виде. В рамках данной работы не требовалось создания кода для непосредственной обработки персональных данных, в анклаве могут быть реализованы собственные методы по обработке данных.

Intel SGX устанавливает соединение с блокчейн сетью, происходит верификация кода анклава и установление общих ключей для асимметричного шифрования данных. После этого данные в зашифрованном виде передаются на оракул (рисунок 3.9).

После обработки в анклаве, чейнкод получает ответ с результатами и сохраняет их. Далее пользователи могут получить доступ к результатам, выполнив запрос к чейнкоду query.

Последним этапом происходит оценка работы оракула и расчет рейтинга в смарт-контракте по индексу доверия.

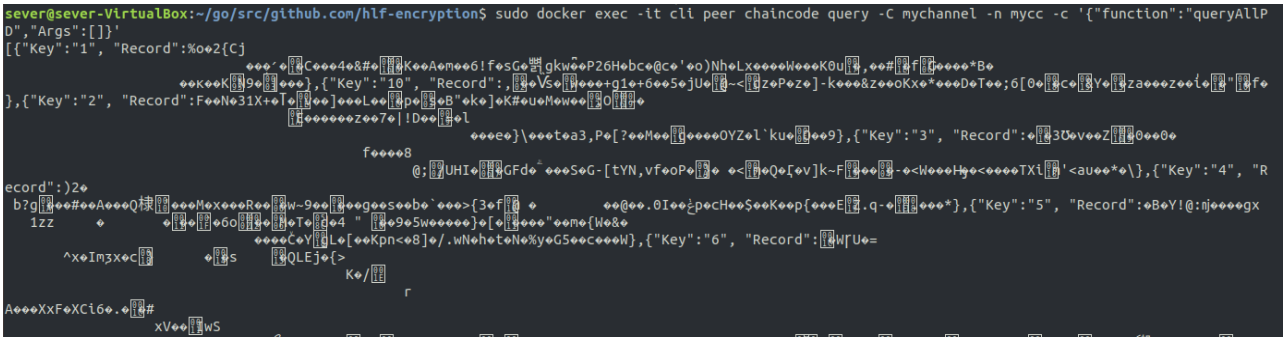

Рисунок 3.9 – Зашифрованные данные для передачи в анклав оракула

Разработанное решение позволяет обрабатывать персональные данные участников децентрализованной сети блокчейн с сохранением их конфиденциальности.

Механизм верификации кода анклава позволяет участникам быть уверенными в том, что персональные данные их клиентов не будут проданы или переданы третьим лицам, так как код прошел аттестацию IAS и соблюдает политику безопасности.

В 4 пункте 3го раздела данной работы не просто так были рассмотрены различные атаки на память анклавов Intel SGX – эта проблема является актуальной и до сих пор, каждый год появляются новые виды атак по сторонним каналам, целый отдел компании Intel с лучшими разработчиками работает над исправлением описанных уязвимостей, но есть тенденции к появлению новых эксплоитов. Предложенная репутационная модель для оракулов не сможет защитить конфиденциальные данные от утечки в открытом виде из анклава в случае новой атаки на память, но сможет определять, какие оракулы можно полагать более безопасными и эффективными, судя по рейтингу. Такой механизм не только создаст правильную конкуренцию среди производителей платформ для удаленных вычислений, но и замотивирует разработчиков находить новые методы защиты от атак.

Сравнение с существующими решениями, такими как HLA, метод «обезличивания данных» и ShadowEth представлено в таблице 3.2. ShadowEth – проект на платформе Ethereum с использованием обработки конфиденциальных данных вне сети в TEE [25].

Таблица 3.2 – Сравнение разработанного решения с уже созданными

	Предложенное решение	Hyperledger Avalon	Метод «Обезличивания данные»	ShadowEth
Управление	Децентрализованное (администратор сети HLF)	Децентрализованное (администратор сети HLA)	Централизованное (центр обработки данных)	Децентрализованное
Шифрование персональных данных	Да, с использованием AES-256	Да, AES-GCM-256 над SHA256 [23]	Нет	Да, RSA с 4096 битным ключом [25]
Использование аттестации кода	Да, IAS + верификация через смарт-контракт	Да, EPID или DCAP [23]	Нет	Да, IAS
Возможность просмотра истории сообщений	Да, через блокчейн HLF	Да, через блокчейн HLA	Да, но возможна подделка логов	Да, через блокчейн Ethereum
Основа доверия при удаленной обработке ПДн	Проверка сертификата + репутационная модель	Проверка сертификата	Договор через нотариуса	Проверка сертификата
Автоматизация	Частично, требуется администратор для установки и обновления чейнкода	Частично, требуется администратор для установки и обновления чейнкода	Нет	Исполнение смарт-контракта не требует поддержки со стороны админа, только для обновления

Исходя из таблицы 3.2 можно сделать следующие выводы: разработанное решение гораздо безопаснее и эффективнее метода «обезличивания данных». Новое решение так же использует TEE для обработки конфиденциальных данных и шифрование, что и решение с Hyperledger Avalon и ShadowEth, но имеет ряд отличий. Во-первых, дополнительный этап проверки при аттестации кода исключает использование небезопасного кода в анклаве. Во-вторых, с применением репутационной модели, возникла качественная характеристика оракулов, что является преимуществом над остальными решениями. Из минусов, сочетание симметричного и асимметричного методов шифрования для передачи приватного ключа в HLA можно считать более надежным, чем формирование общего ключа с помощью протокола Диффи-Хеллмана в разработанном решении.

Таким образом, в данном разделе выпускной квалификационной работы был рассмотрен пример с уже работающим проектом Hyperledger Avalon, использующим сочетание блокчейна и TEE. Была разобрана общая схема передачи персональных данных между участниками сети, выделены основные этапы в последовательности потока данных. Разработанное решение было протестировано в сети Hyperledger Fabric с подключенным внешним оракулом с анклавом Intel SGX, приведены результаты работы смарт-контрактов и передачи зашифрованных персональных данных, также проведено сравнение разработанного решения с уже существующими, выделены плюсы и минусы нового решения.

4 ПРАКТИЧЕСКОЕ ПРИМЕНЕНИЕ

Разработанный оракул для обработки персональных данных отвечает всем требованиям безопасности, чтобы эффективно использоваться в самых разных сферах. Использование блокчейн в данном решении позволяет объединять данные различных независимых компаний в одну систему и производить наиболее точные вычисления.

4.1 Оракул для умного голосования

Голосование на выборах всегда было предметом для споров о фальсификации голосов, дошло до обвинений в международном вмешательстве в выборы президента США. Подобного бы не случилось, если бы использовалась технология распределенного реестра, где доказательство отданного голоса записано в цепочке блоков и не может быть изменено (рисунок 4.1).

Блокчейн сам по себе публичный, то есть данные в нем хранятся в открытом виде и могут быть прочитаны участниками сети. Так как голосование подразумевает обязательную идентификацию личности с помощью документа, удостоверяющего личность, хранить такие данные в открытом виде недопустимо, поэтому предлагается использовать разработанное решение: оракул с доверенной средой исполнения. В качестве участников сети могут выступать избирательные участки, которые отправляет собранные голоса с персональными данными в оракул, где происходит обработка голосов и их подсчет. Такое решение позволило бы автоматизировать процесс голосования, сократить расходы и повысить точность подсчетов, исключив ложные бюллетени.

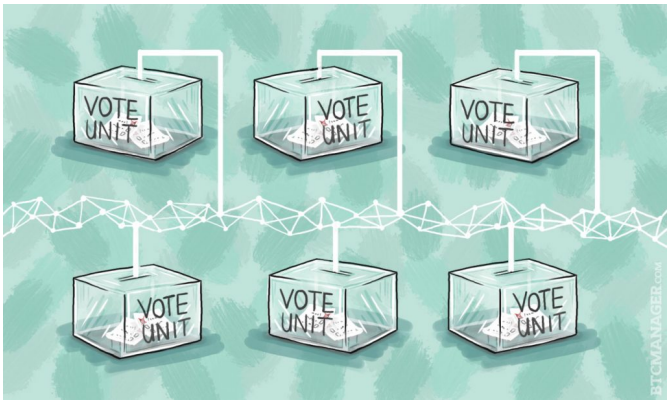

Рисунок 4.1 – Схема голосования с блокчейн

4.2 Возможные области применения разработанных механизмов

К наиболее эффективным областям применения разработанных механизмов относятся медицинская и банковская системы.

Для повышения качества лечения, частные клиники и научно-исследовательские институты смогут обмениваться данными без страха утечки конфиденциальных данных своих клиентов. Объединив усилия, множество медицинских центров сможет передавать важные данные о здоровье пациентах, их историю болезни и информацию об исследуемых вакцинах на доверенный оракул с применением искусственного интеллекта. Как известно, для качественной работы нейросети необходимо большое количество статистических данных и решение именно в объединении больших данных от разных источников. Разработанный оракул предоставляет необходимые механизмы для безопасной передачи и обработки таких данных.

Доверенный оракул с нейросетью, помогающей найти вакцину от коронавирусной инфекции, мог бы ускорить процесс разработки наиболее действенной вакцины, если бы медицинские центры объединили данные о результатах применения новых вакцин в своих лабораториях.

Использование блокчейна в банковской системе с доверенной средой обработки данных позволит осуществлять моментальные или почти моментальные платежи, глобально совместимые с высоким уровнем безопасности и без утечки личных данных [1].

4.3 Выявление недостатков и способов их устранения

Из недостатков можно отметить низкую скорость работы приложений с анклавами Intel SGX [26]. Группа исследователей провела эксперимент на двух машинах, оснащенных процессором Intel Core i7-7700 с поддержкой Intel SGX (четырехъядерный с SMT; 3,60 ГГц) и 32 ГБ оперативной памяти, работающим под управлением Ubuntu 18.04.3 LTS и подключенными через гигабитный Ethernet. Выборочное исследование состояло из короткого демографического опросника из 10 пунктов.

Исследователи последовательно отправляли на платформу до 10 000 участников для каждого прогона оценки перед переходом к выполнению статистического анализа. Для каждой отправки ответа на опрос измерялось общее время ответа на стороне клиента. Для каждого запуска исследования также измерялось общее время выполнения анализа после сбора всех ответов. Реализация без анклава заняла в среднем 15,8 мс для обработки. В свою очередь, для обработки в анклаве требуется 29,7 мс. На рисунке 4.2 показано влияние различных размеров выборки (от 10 до 1000) на время выполнения с применением SGX и без него. Для размера выборки N = 10 000 среднее время вычислений с использованием SGX в 15,7 раз больше, чем без SGX (4,6 с и 72,4 с).

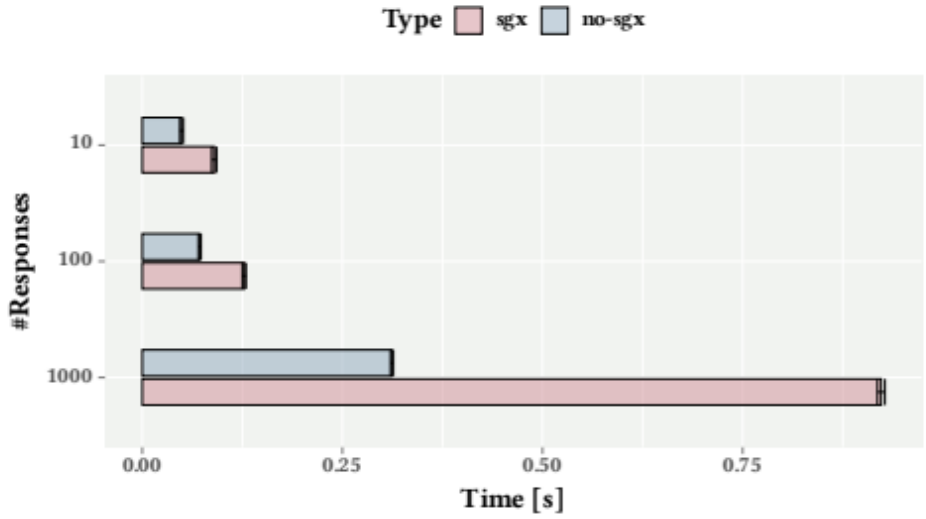

Рисунок 4.2 – Сравнение времени работы с анклавом и без него

Для устранения данной проблемы следует сокращать код анклава до минимума, оставляя самые необходимые функции и добавлять параллельные анклавы, чтобы распределять добавочную нагрузку.

Еще одним недостатком может быть то, что использование Intel SGX накладывает ограничения на используемую технику. Данное расширение поддерживают все современные процессоры Intel, начиная с 5го поколения, на более старых версиях использовать Intel SGX не получится. Решением может служить запуск SGX на виртуальной машине в режиме симуляции или альтернативное решение с ARM TrustZone.

Появление новых side-channel attacks создает тенденцию к постоянному обновлению микрокода процессоров Intel, что сказывается на работе самих процессора, иногда замедляя их работу. Решением может послужить переход компании Intel на новый тип процессоров, применяющих более надежные методы защиты от атак по сторонним каналам.

Таким образом, в данном разделе выпускной квалификационной работы был предложений сценарий использования доверенного оракула для умного голосования, рассмотрены возможные варианты применения в других сферах и выявлены недостатки решения.

ЗАКЛЮЧЕНИЕ

В данной выпускной квалификационной работе были исследованы современные механизмы, предоставляющие возможность доверенных удаленных вычислений и рассмотрены подходы к построению децентрализованных сетей блокчейн. Объединение этих двух механизмов дополнило друг друга и позволило расширить количество подходов к увеличению доверия к системе.

Разработанный оракул для обработки персональных данных работает отдельно от основной сети и использует создание анклавов Intel SGX для изоляции персональных данных от ненадежной части ОС. Использование надежных алгоритмов шифрования и схемы подписи обеспечивает конфиденциальность и целостность критически важных данных.

Для увеличения доверия к оракулу были предложены следующие инструменты:

- верификация кода анклава через смарт-контракт;
- создание репутационной модели для оракула.

Данные решения позволяют проводить дополнительную проверку кода, выполняемого в защищенной среде анклава и формировать рейтинг оракула на основе индекса доверия через смарт-контракт.

Разработанное решение подходит для совместного решения сложных задач обработки больших данных многими компаниями. Технология блокчейн автоматизирует процесс и позволяет делиться данными компаний с другими участниками, тем самым по принципу «win-win» все участники сети получают более качественные статистические данные, сохраняя основные принципы безопасности по отношению к персональным данным пользователей.

В дальнейшей разработке может быть реализован маркетплейс с множеством оракулов для обработки больших данных, в котором будет эффективно работать репутационная модель для создания рейтинга и поддержания конкуренции среди участников рынка.

СПИСОК ИСПОЛЬЗОВАННЫХ ИСТОЧНИКОВ

1. Дон Тапскотт, Алекс Тапскотт Технология блокчейн. То, что движет финансовой революцией сегодня – 2017. – С. 95-96.

2. Григорий Дубов Три менеджера Uber подали в отставку после хищения данных 57 млн клиентов [Электронный ресурс]. URL: https://www.rbc.ru/rbcfreenews/5a21f7a79a7947420515b399 - (дата обращения 15.05.2021).

3. John P Mechalas Introducing the intel® software guard extensions tutorial series [Электронный ресурс]. URL: https://software.intel.com/content/www/us/en/develop/articles/introducing-the-intel-software-guard-extensions-tutorial-series.html - (дата обращения 15.05.2021).

4. Савельев А.И. Проблемы применения законодательства о персональных данных в эпоху «Больших данных» (Big Data) // Право. Журнал Высшей школы экономики. 2015. No1. С. 43–66.

5. Дорохов В. В. Блокчейн-технологии: будущее финансовой системы [Электронный ресурс]. URL: https://cyberleninka.ru/article/n/blokcheyn-tehnologii-buduschee-finansovoy-sistemy - (дата обращения 17.05.2021).

6. Radhesh Krishnan Konoth, Emanuele Vineti MineSweeper: An In-depth Look into Drive-by Cryptocurrency Mining and Its Defense [Электронный ресурс]. URL: https://dl.acm.org/doi/pdf/10.1145/3243734.3243858 – (дата обращения 17.05.2021).

7. Forsage.io Взломать блокчейн. Возможно ли это? [Электронный ресурс]. URL: https://vc.ru/crypto/202105-vzlomat-blokcheyn-vozmozhno-li-eto - (дата обращения 17.05.2021).

8. Abdul Wahab extending Hyperledger Fabric network: Adding a New Peer [Электронный ресурс]. URL: https://wahabjawed.medium.com/extending-hyperledger-fabric-network-adding-a-new-peer-4f52f70a7217 - (дата обращения 18.05.2021).

9. Hyperledger Fabric Documentation [Электронный ресурс]. URL: https://hyperledger-fabric.readthedocs.io/ru/latest/whatis.html - (дата обращения 18.05.2021).

10. Что такое Hyperledger Fabric Блог компании IBM [Электронный ресурс]. URL: https://clck.ru/UsfHm - (дата обращения 18.05.2021).

11. Intel Software Guard Extensions, серия учебных материалов. Часть 1, основы Intel SGX Блог компании Intel [Электронный ресурс]. URL: https://habr.com/ru/company/intel/blog/312002/ - (дата обращения 19.05.2021).

12. Victor Costan, Srinivas Devadas Intel SGX Explained [Электронный ресурс]. URL: http://css.csail.mit.edu/6.858/2020/readings/costan-sgx.pdf - (дата обращения 19.05.2021).

13. 5.3 Уровни привилегий // Современные процессоры Intel IA-32 в ПК [Электронный ресурс]. URL: https://dims.karelia.ru/x86/pr_3.shtml - (дата обращения 19.05.2021).

14. Robert Watson, Trusted Code Base in a UNIX Environment [Электронный ресурс]. URL: https://lists.freebsd.org/pipermail/trustedbsd-discuss/2000-April/000050.html - (дата обращения 19.05.2021).

15. Основы шифрования (часть 1) - Алгоритм Диффи-Хеллмана [Электронный ресурс]. URL: https://www.securitylab.ru/analytics/478912.php - (дата обращения 20.05.2021).

16. Daniel Ehnas The Magic of Intel’s SGX [Электронный ресурс]. URL: https://medium.com/magicofc/the-magic-of-intels-sgx-how-to-hello-it-sec-world-fb0295d6c33b - (дата обращения 21.05.2021).

17. Стражи публичных облаков: как мы внедряли анклавы Intel SGX для защиты чувствительных данных [Электронный ресурс]. URL: https://habr.com/ru/company/gcorelabs/blog/537316/ - (дата обращения 20.05.2021).

18. Fan Zhang, Ethan Cecchetti Town Crier: An Authenticated Data Feed for Smart Contracts [Электронный ресурс]. URL: https://town-crier.org/files/2016/168.pdf - (дата обращения 20.05.2021).

19. Thomas Knauth, Michael Steiner Integrating Intel SGX Remote Attestation with Transport Layer Security [Электронный ресурс]. URL: https://arxiv.org/pdf/1801.05863.pdf - (дата обращения 20.05.2021).

20. Ferdinand Brasser, Urs Muller Software Grand Exposure: SGX Cache Attacks Are Practical [Электронный ресурс]. URL: https://www.usenix.org/system/files/conference/woot17/woot17-paper-brasser.pdf - (дата обращения 20.05.2021).

21. Town Crier vs DECO: какой оракул использовать в блокчейне? [Электронный ресурс]. URL: https://habr.com/ru/post/518234/ - (дата обращения 20.05.2021).

22. Новые атаки на процессоры Intel достают криптоключи из анклава SGX даже после изоляции ядер CPU [Электронный ресурс]. URL: https://habr.com/ru/company/itsumma/news/t/506162/ - (дата обращения 20.05.2021).

23. Hyperledger Avalon Architecture Overview Revision 0.3 [Электронный ресурс]. URL: https://hyperledger.github.io/avalon/docs/avalon-arch.pdf - (дата обращения 21.05.2021).

24. Индекс уровня доверия клиентов к компании. Ещё один шаг к абсолютной лояльности [Электронный ресурс]. URL: https://blog.anketolog.ru/2016/02/uroven_doveria/ - (дата обращения 22.05.2021).

25. Rui Yuan, Hai-Bo Chen ShadowEth: Private Smart Contract on Public Blockchain [Электронный ресурс]. URL: https://www.trustkernel.com/uploads/pubs/shadoweth.pdf - (дата обращения 22.05.2021).

26. Dominik Meißner, Felix Engelmann PeQES: A Platform for Privacy-enhanced Quantitative Empirical Studies [Электронный ресурс]. URL: https://arxiv.org/pdf/2103.05544.pdf - (дата обращения 22.05.2021).

ПРИЛОЖЕНИЕ 1

```
package main
/* Imports
* 4 utility libraries for handling bytes, reading and writing JSON,
formatting, and string manipulation
* 2 specific Hyperledger Fabric specific libraries for Smart Con-
tracts
*/
import (
     "bytes"
     "encoding/json"
     "fmt"
     "strconv"
        fc "github.com/chaincode/fabcrypt"
     "github.com/hyperledger/fabric/core/chaincode/shim"
     sc "github.com/hyperledger/fabric/protos/peer"
)

// Define the Smart Contract structure
type SmartContract struct {
}

/* Define Records structure, with 4 properties.
Structure tags are used by encoding/json library
*/
type Records struct {
     Fullname  string `json:"fullname"`
     DOB         string `json:"dob"`
     Insurance string `json:"insurance"`
     Vaccine   string `json:"vaccine"`
     Status    string `json:"status"`
}

/*
 * The Init method *
 called when the Smart Contract is instantiated by the network
 * Best practice is to have any Ledger initialization in separate
function
 -- see initLedger()
 */
func (s *SmartContract) Init(APIstub shim.ChaincodeStubInterface)
sc.Response {
     return shim.Success(nil)
}
```

```
/*
 * The Invoke method *
 called when an application requests to run the Smart Contract
 The app also specifies the specific smart contract function to call
with args
 */
func (s *SmartContract) Invoke(APIstub shim.ChaincodeStubInterface)
sc.Response {

     // Retrieve the requested Smart Contract function and arguments
     function, args := APIstub.GetFunctionAndParameters()
     // Route to the appropriate handler function to interact with
the ledger
     if function == "queryPD" {
          return s.queryPD(APIstub, args)
     } else if function == "initLedger" {
          return s.initLedger(APIstub)
     } else if function == "recordPD" {
          return s.recordPD(APIstub, args)
     } else if function == "queryAllPD" {
          return s.queryAllPD(APIstub)
     } else if function == "changePD" {
          return s.changePD(APIstub, args)
     }

     return shim.Error("Invalid Smart Contract function name.")
}

/*
 * The queryPD method *
Used to view the records of one particular tuna
It takes one argument -- the key for the tuna in question
 */
func (s *SmartContract) queryPD(APIstub shim.ChaincodeStubInter-
face, args []string) sc.Response {

     if len(args) != 1 {
          return shim.Error("Incorrect number of arguments. Expect-
ing 1")
     }

     tunaAsBytes, _ := fc.Decrypter(APIstub, args[0])
     if tunaAsBytes == nil {
          return shim.Error("Could not locate tuna")
     }
```

```
     return shim.Success(tunaAsBytes)
}

/*
 * The initLedger method *
Will add test data (10 records)to our network
 */
func (s *SmartContract) initLedger(APIstub shim.ChaincodeStubInter-
face) sc.Response {
     tuna := []Records{
          Records{Fullname: "Dylan Freedman",     DOB:
"08.03.1996", Insurance: "Aflac:AOP1504XZ05A225",          Vac-
cine: "Sputnik V",  Status:"Well"},
          Records{Fullname: "Omar Garden",   DOB: "02.04.1999", In-
surance: "Aflac:R1C50405782DGL5",      Vaccine:    "Sputnik    V",
     Status:"Sick"},
          Records{Fullname: "Ivan Smith",         DOB:
"23.03.1976", Insurance: "Assurant:P1S4935TT170S567", Vaccine:
"Pfizer",      Status:"Well"},
          Records{Fullname: "Anna Proper",   DOB: "15.10.1983", In-
surance: "Reso:TXX1496REWW105425",     Vaccine: "Moderna",
     Status:"Well"},
          Records{Fullname: "Alex Grey",          DOB:
"30.01.1985", Insurance: "Unum:1IDP49351SS230Y1",     Vaccine:
"Pfizer",      Status:"Sick"},
          Records{Fullname: "Tomas Edison", DOB: "02.01.2000", In-
surance: "Unum:1W4E94G11BNM7101Q",     Vaccine:      "AstraZeneca",
     Status:"Well"},
          Records{Fullname: "Mia Heigl",          DOB:
"24.12.1971", Insurance: "GEICO:VF49655F00104F301",   Vaccine:
"Pfizer",      Status:"Sick"},
          Records{Fullname: "Akay Fox",      DOB: "19.07.1994", In-
surance: "Omega:1445NHD8506D6L691A",   Vaccine:    "Sputnik    V",
     Status:"Well"},
          Records{Fullname: "Elvis Hughes", DOB: "28.09.1964", In-
surance: "Reso:2F14S85153CVB091P",     Vaccine:    "Sputnik    V",
     Status:"Well"},
          Records{Fullname: "Tanya Makarova", DOB: "14.02.1973", In-
surance: "Reso:PL1487A745DC091BN",     Vaccine: "Moderna",
     Status:"Well"},
     }

     i := 0
     for i < len(tuna) {
          fmt.Println("i is ", i)
```

```
          tunaAsBytes, _ := json.Marshal(tuna[i])
          fc.Encrypter(APIstub,  strconv.Itoa(i+1),  []byte(tunaAs-
Bytes))
          fmt.Println("Added", tuna[i])
          i = i + 1
     }

     return shim.Success(nil)
}

/*
 * The recordPD method *
This method takes in five arguments (attributes to be saved in the
ledger).
 */
func (s *SmartContract) recordPD(APIstub shim.ChaincodeStubInter-
face, args []string) sc.Response {

     if len(args) != 6 {
          return shim.Error("Incorrect number of arguments. Expect-
ing 5")
     }

     var tuna = Records{ Fullname: args[1], DOB: args[2], Insurance:
args[3], Vaccine: args[4], Status: args[5] }

     tunaAsBytes, _ := json.Marshal(tuna)
      error := fc.Encrypter(APIstub, args[0], []byte(tunaAsBytes))
     if error != nil {
          return  shim.Error(fmt.Sprintf("Failed  to  Encrypt  tuna
catch: %s", args[0]))
           }

     return shim.Success(nil)
}

/*
 * The queryAllPD method *
allows for assessing all the records added to the ledger(all tuna
catches)
This method does not take any arguments. Returns JSON string con-
taining results.
 */
func (s *SmartContract) queryAllPD(APIstub shim.ChaincodeStubInter-
face) sc.Response {
```

```
	startKey := "0"
	endKey := "999"

	resultsIterator, err := APIstub.GetStateByRange(startKey, end-
Key)
	if err != nil {
		return shim.Error(err.Error())
	}
	defer resultsIterator.Close()

	// buffer is a JSON array containing QueryResults
	var buffer bytes.Buffer
	buffer.WriteString("[")

	bArrayMemberAlreadyWritten := false
	for resultsIterator.HasNext() {
		queryResponse, err := resultsIterator.Next()
		if err != nil {
			return shim.Error(err.Error())
		}
		// Add comma before array members,suppress it for the first
array member
		if bArrayMemberAlreadyWritten == true {
			buffer.WriteString(",")
		}
		buffer.WriteString("{\"Key\":")
		buffer.WriteString("\"")
		buffer.WriteString(queryResponse.Key)
		buffer.WriteString("\"")

		buffer.WriteString(", \"Record\":")
		// Record is a JSON object, so we write as-is
		buffer.WriteString(string(queryResponse.Value))
		buffer.WriteString("}")
		bArrayMemberAlreadyWritten = true
	}
	buffer.WriteString("]")

	fmt.Printf("- queryAllPD:\n%s\n", buffer.String())

	return shim.Success(buffer.Bytes())
}

/*
```

```
 * The changePD method *
The data in the world state can be updated with who has possession.
This function takes in 2 arguments, tuna id and new holder name.
 */
func (s *SmartContract) changePD(APIstub shim.ChaincodeStubInter-
face, args []string) sc.Response {

     if len(args) != 2 {
          return shim.Error("Incorrect number of arguments. Expect-
ing 2")
     }
     tunaDec, error := fc.Decrypter(APIstub, args[0] )
     if error != nil {
          return shim.Error("Could not  get or Decrypt tuna")
     }

     tuna := Records{}

     json.Unmarshal(tunaDec, &tuna)
     // Normally check that the specified argument is a valid holder
of tuna
     // we are skipping this check for this example
     tuna.Fullname = args[1]

     tunaDec, _ = json.Marshal(tuna)
        err := fc.Encrypter(APIstub, args[0], []byte(tunaDec))
     if err != nil {
       return shim.Error(fmt.Sprintf("Failed to Encrypt changed tuna
holder: %s", args[0]))
           }

     return shim.Success(nil)
}
/*
* Check hash of the Enclave SGX
*/
func (s *SmartContract) checkHash(APIstub shim.ChaincodeStubInterface,
args []string) sc.Response
{
     if len(args) != 1 {
          return shim.Error("Incorrect number of arguments. Expect-
ing 1")
     }
     tunaAsBytes, _ := fc.Decrypter(APIstub, args[0])
     if tunaAsBytes == nil {
          return shim.Error("Could not locate tuna")
```

```
     }
if(tunaAsBytes == args[1])
     return shim.Success(tunaAsBytes)
else return shim.Error("Invalid hash, enclave is not valid")
}
/*
 * main function *
calls the Start function
The main function starts the chaincode in the container during in-
stantiation.
 */
func main() {

     // Create a new Smart Contract
     err := shim.Start(new(SmartContract))
     if err != nil {
          fmt.Printf("Error creating new Smart Contract: %s", err)
     }}
```

ПРИЛОЖЕНИЕ 2

```
echo "Removing key from key store..."
rm -rf ./hfc-key-store
cd chaincode
rm -rf errors
git clone https://github.com/pkg/errors.git
cd ..

cd basic-network
./start.sh

sudo docker ps -a

sudo docker-compose -f docker-compose.yml up -d cli

sudo docker ps -a


echo 'Installing chaincode..'
sudo docker exec -it cli peer chaincode install -n mycc -v 1.0 -p
"github.com/chaincode/"

echo 'Instantiating chaincode..'
sudo docker exec -it cli peer chaincode instantiate -o orderer.ex-
ample.com:7050 -C mychannel -n mycc -v 1.0 -c '{"Args":[]}'

echo 'Getting things ready for Chaincode Invocation..should take
only 10 seconds..'

sleep 10
echo 'Initializing Ledger'

sudo docker exec -it cli peer chaincode invoke -o orderer.exam-
ple.com:7050 -C mychannel -n mycc -c '{"function":"in-
itLedger","Args":[]}'
sleep 3
echo 'querying test record..'
sudo docker exec -it cli peer chaincode query -C mychannel -n mycc
-c '{"function":"queryPD","Args":["3"]}'
echo 'querying all records..'
sudo docker exec -it cli peer chaincode query -C mychannel -n mycc
-c '{"function":"queryAllPD","Args":[]}'
sleep 1
echo 'Success!'
exit 1
```