ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Согласовано

Профессор департамента
программной инженерии факультета
компьютерных наук, канд. техн. наук

_____ С.М. Авдошин

”____”_____ 2019 г

Утверждаю

Академический руководитель
образовательной программы
«Программная инженерия»
профессор, канд. техн. наук

_____ В. В. Шилов

”____”_____ 2019 г

Выпускная квалификационная работа по теме Криптографические алгоритмы и протоколы для распределенных реестров

по направлению подготовки 09.03.04 «Программная инженерия»

Студент группы БПИ 151 НИУ ВШЭ
_____ Куприянов К. И.
”____”_____ 2019 г

2019

Реферат

Технология блокчейн обычно ассоциируется с криптовалютой биткойн, потому что биткойн - первая повсеместно используемая система, использующая блокчейн как основу. По мере развития технологий число различных блокчейнов со множеством способов их приложения резко возросло. Факт существования такого значительного их количества можно объяснить тем, что при их реализации могут варьироваться используемые криптографические алгоритмы и протоколы. В связи с этим возникла проблема отсутствия систематически собранной и структурированной информации о криптографических алгоритмах и протоколах в существующих распределенных реестрах. Главной целью данной работы является сбор и обобщение известных и распространенных на сегодняшний день криптографических алгоритмов и протоколов. Предложен сравнительный анализ алгоритмов, используемых в блокчейнах, по общим показателям. Также в качестве инструмента для разработчиков при создании персонального распределенного реестра в образовательных целях разработана библиотека на языке Python3.6, в которой собраны реализации проанализированных алгоритмов и протоколов.

Ключевые слова — блокчейн, биткоин, распределённый реест, технология распределённого реестра, криптография, классификация, Python.

Abstract

The Blockchain technology is typically associated with Bitcoin, because it was the first system which has been distributed using the Blockchain technology. As the technologies evolved, the number of various blockchains with different kinds of applications had been drastically risen. A huge amount of blockchains can be explained by various cryptographic algorithms and protocols usage in them. It brought a problem of the absence of systematically gathered and structured information about cryptographic algorithms and protocols in existing distributed ledgers. The main objective of this work is to generalise all known common cryptographic algorithms and protocols, which are being used nowadays. The algorithms used in blockchains are going to be classified by common metrics: time complexity, space complexity, and the resistance to hacking. It is also intended to bring a programming library in Pyhon3.6, where analyzed algorithms and protocols are gathered in one place. The library would serve as a toolbox for developers when creating a personal distributed ledger.

Index terms — blockchain; bitcoin; distributed ledger; DLT; cryptography; classification; python

Содержание

Определения

1.1 Терминология 5

Введение

Глава 1. Обзор распределённых реестров, источников и решений

3.1 Существующие решения 8

3.1.1 Аналитическая составляющая 8

3.1.2 Программная составляющая 9

3.2 Источники для изучения 10

3.3 Виды распределённых реестров 10

3.4 Структура реестров 11

3.4.1 Блокчейны 11

3.4.2 Направленные ациклические графы (DAGs) 11

3.4.3 Плюсы и минусы DAG 11

3.5 Алгоритмы 12

3.5.1 SHA-256 12

3.5.2 ECDSA 12

3.6 Прочие алгоритмы 13

3.6.1 Ring Signatures 13

3.6.2 Stealth Addressess 13

3.6.3 Coinjoin, coinshuffle 14

3.6.4 Sigma Protocol 14

3.6.5 Zerocash 15

3.6.6 CT, RingCT 15

3.6.7 MimbleWimble 15

3.6.8 Zexe 15

3.7 Протоколы консенсуса 16

3.7.1 Proof of Work (PoW) 16

3.7.2 Proof of Stake (PoS) 16

3.7.3 Delegated Proof-of-Stake (DPoS) 16

3.7.4 Proof-of-Authority (PoA) 17

3.7.5 Byzantine Fault Tolerance (BFT) 17

3.8 Выводы по главе 17

Глава 2. Проектирование сервиса: архитектура, процессы

4.1 Архитектурные особенности 19

4.1.1 Общая структура проектного решения 19

4.1.2 Архитектура компоновщика 20

4.1.3 Порядок работы компоновщика 21

4.1.4 Архитектура реализации блокчейна 22

4.1.5 Работа с данными 24

4.2 Выводы 25

Глава 3. Программная реализация

5.1 Функциональные требования 26

5.2 Описание реализации процесса генерации кода компоновщиком 27

5.2.1 Сбор опций пользователя 27

5.2.2 Поиск алгоритмов 28

5.2.3 Установка 29

5.2.4 Запись реализации блокчейна 29

5.3 Описание реализации блокчейна 30

5.3.1 wallet.py 30

5.3.2 miner.py 32

5.4 Описание процесса автообновления 34

5.5 Описание реализации хранилища данных 36

Глава 4. Проведение эксперементов

6.1 Анализ времени исполнения алгоритмов в реализации блокчейнов 38

6.2 Выводы 41

Заключение

Список использованных источников

1. Определения

1.1. Терминология

Распределённый реестр (Distributed Ledger) — В примитивной своей реализации это распределённая база данных между сетевыми узлами или вычислительными устройствами. Каждый из узлов может получать данные других, при этом храня полную копию реестра. Обновления этих узлов происходят независимо друг от друга.

Блокчейн — Постоянно растущий список записей, называемых блоками, которые связаны и защищены с помощью криптографии. Он копируется его пользователями и устойчив к модификации. Машина с рабочей копией называется узлом.

DAG — Направленный ациклический граф. Это ориентированный граф с данными, использующий топологическую сортировку (от ранних узлов к более поздним).

Биткоин (Bitcoin) — Электронная пиринговая платёжная система, используемая в качестве финансовой единицы (криптовалюты) одноимённую сущность. Создателем биткоина выступает некто Satoshi Nakomoto [21].

Эфириум (Ethereum) — Открытая, общедоступная, вторая по популярности, распределенная вычислительная платформа на основе технологии блокчейн и операционная система с функциональностью смарт-контрактов [34]

Алгоритм консенсуса — Набор математических операций, которые необходимо выполнять для поддержания консистентности всей сети.

2. Введение

В последние годы наблюдается усиленный рост интереса общества к криптовалютам, блокчейнам и распределённым реестрам. Последние десять лет показали, что прогресс в изучении области распределенных реестров и криптовалюты огромен по сравнению с объёмом прогресса в любой другой областью в прошлом. Один из результатов растущего интереса — тысячи исследований, проведенных учеными и исследователями.

Эти исследования положили начало созданию сотен криптовалют и других видов применений распределенных реестры. Некоторые криптовалюты повторяют друг друга в плане алгоритмов (хэширования, других крипто-алгоритмов) и протоколов. Относительно мало исследований было проведено на используемых алгоритмах и еще меньше на протоколах. Обобщённость многих статей на эти темы проблематична. В то время как большинство веб-сайтов предоставляют ненаучные бизнес-объяснения, есть одно исследование [29], в котором присутствует некоторая агрегация информации об алгоритмах и протоколах в распределенных реестрах.

Однако, во-первых, остаётся ряд неотъемлемых технических вопросов о классификации алгоритмов и протоколов для распределенных реестров, а во-вторых, данное исследование имело место быть в 2014 году, что делает информацию в нем устаревшей. Появляются новые алгоритмы и современные приложения старых, новые криптовалюты и их протоколы публикуются и запускаются каждый год, и в в настоящий момент есть много возможностей для расширения в этой области.

Данная работа сфокусирована на исследовании использования современных алгоритмов и протоколов в распределенных реестрах, актуализации их классификации и создании программного решения по автоматизации работы создания блокчейна.

Цель состоит в том, чтобы **расширить существующую классификацию** новыми алгоритмами и протоколами, фокусируясь на ширине охвата современных технологий, а не на их количестве. Обновлённая классификация отражает фактическое состояние алгоритмов и протоколов в распределённых реестрах на вторую четверть 2019 года.

Помимо этого, важной частью работы является **создать удобное приложение для автоматизации программирования**, которое генерировало бы готовый код блокчейна с использованием алгоритмов, выбранных пользователями. Данный продукт будет служить “инструментарием” для оператора или любого другого интересующегося криптографическими алгоритмами и протоколами, который имел бы потребность интегрировать блокчейн в своё приложение (регистрация гостей в отеле, социальную сеть, переводы, учёт документов). Также программа будет полезна людям, которые хотят узнать как работают современные распределённые реестры с рассмотренными аспектами. Это позволит быстро получать необходимую техническую информацию, которую с трудом можно найти в общем доступе. Программа должна предоставлять не только генерацию кода, но и дружелюбный интерфейс командной строки, в которой форматирование и подсветка не будут сбивать с толку неподготовленного пользователя. Главной чертой данного приложения является самоподдерживаемая система по работе с исходными кодами алгоритмов, расположенными удалённо. А также лёгкая, быстрая масштабируемость и модульность программного кода.

Научная новизна работы заключается в том, что разработана новая актуальная классификация алгоритмов и протоколов для распределённых реесстров (и самих реестров), тем самым дополнив существующую (но устаревшую) на сегодняшней день классификацию 2014-го года.

Практическая значимость работы заключается в том, что разработанная программа способна автоматизировать процесс программирования распределённых реестров и может быть использована организациями и физическими лицами для создания приложений с использованием

готовой архитектуры блокчейна с уникальными (выбранными пользователем) алгоритмами.

3. Глава 1. Обзор распределённых реестров, источников и решений

В данной главе описаны возможные технологии распределённых реестров. Под технологиями понимаются алгоритмы, протоколы, а также общая структура реестра. Распределённые реестры делятся на открытые (Public), закрытые (Private), эксклюзивные (Permissioned), и инклюзивные (Permissionless). По структуре реализации распределённые реестры делятся на 2 вида: блокчейны и направленные ациклические графы (DAGs). Алгоритмы, представленные во всех типах являются неотъемлемой их частью. Это алгоритмы хэширования, электронных подписей, генерации случайных чисел. Специфические для конкретных реестров алгоритмы, обеспечивающие должный уровень безопасности/анонимности, например Ring signatures, Coinjoin, Coinshuffle, stealth addresses, MimbleWimble, etc. будут рассмотрены в Главе 2. Помимо алгоритмов, будут рассмотрены и различные протоколы консенсуса, (общей сложностью более 20 штук), обеспечивающие защиту и надёжность транзакций в открытых блокчейнах.

3.1. Существующие решения

3.1.1. Аналитическая составляющая

На сегодняшний день существует следующая классификация алгоритмов и протоколов для распределёных реестров (Рис. 4):

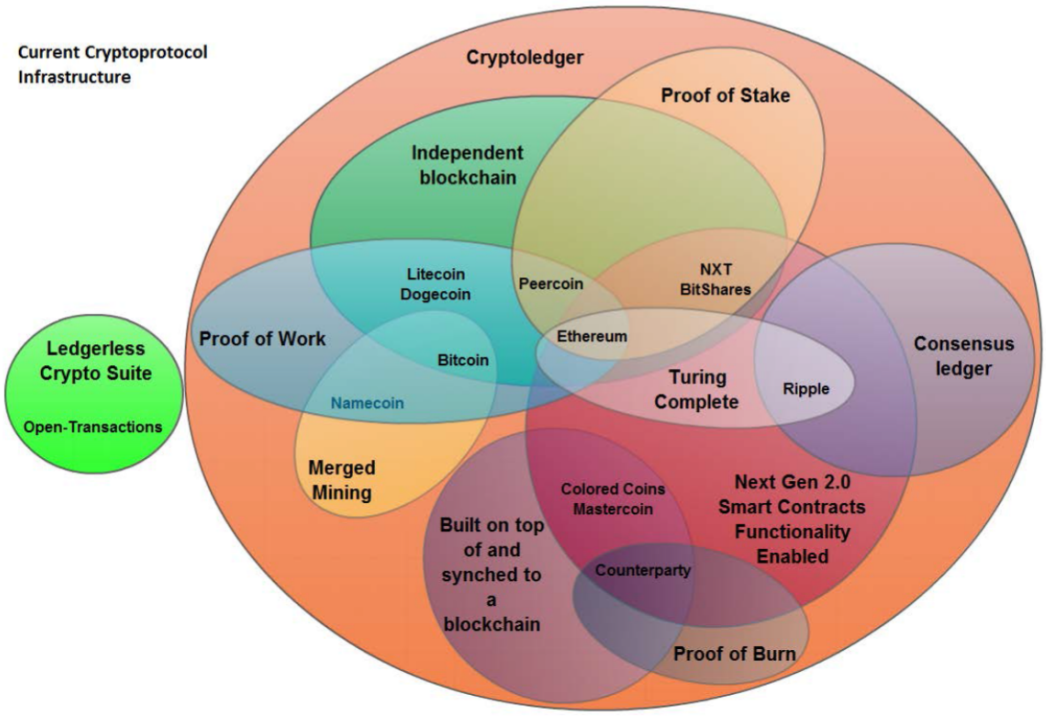

Рис. 1: Классификация на 2014 год [29]

Данная классификация представлена в книге Тима Суэнсона [29] и являлась актуальным на 2014 год представлением классификации криптографических алгоритмов и протоколов для распределённых реестров. Сегодня, когда появилось множество новых алгоритмов и были изобретены уникальные типы реестров, данная классификация утратила свою значимость и нуждается в обновлении и актуализации. В рамках данной работы будет представлена новая классифика-

ция алгоритмов и протоколов для распределённых реестров. Классификация сохранит принципы построения и представления своего предшественника.

3.1.2. Программная составляющая

Программной составляющей настоящего проекта является приложение из двух частей:

1. Части, позволяющей сгенерировать код блокчейна с использованием выбранных пользователем алгоритмов
2. Части, которая является выходом первой компоненты, и по своей сути обособленным приложением — блокчейном

В дальнейшем (1) будет именоваться **компоновщик**, а (2) — **реализация блокчейна**.

3.1.2.1 Codecreator.com

Данный ресурс [7] (на вкладке blockchain) представляет собой сервис, предназначенный для развёртывания реестра на сервере Amazon AWS. Ресурс позволяет развернуть реестры, основанные на Ethereum и Hyperledger. Ресурс предназначен для крупных бизнес проектов, требующих рабочий код быстро. Имеет бесплатную пробную версию.

3.1.2.2 Azure BaaS

Данная платформа [20], Microsoft Azure Blockchain as a Service, обеспечивает быструю, относительно недорогую, платформу для создания и развертывания блокчейн-приложений. Azure предлагает Blockchain как сервис (BaaS), а значит, встраивание кода блокчейна в код не предусматривается. Есть поддержка самых популярных реестров, таких как Ethereum, Quorum, Hyperledger Fabric, Corda.

3.1.2.3 Magic Code Generator

Релиз данной платформы [17] ещё не состоялся. Документы, содержащиеся на страницах описывают сервис, предназначенный для построения приложений, взаимодействующих с сетями реестров. Например, онлайн биржу валют. Также сервис имеет возможность создания blockchain-based приложений.

Данный проект *прямым* аналогом ни одного из приведённых приложений не является. Это позволяет сказать, что проект уникальный в своём роде. Тем не менее, есть сходства и различия, по которым можно привести следующее сравнение (Табл. 1).

Таблица 1. Сравнение с аналогами

Признак	Codecreator	Azure BaaS	MagicCodeGenerator[1]	**GSL**
Встраивание кода в своё приложение	Частично	Частично	—	**Да**
Моментальное развёртывание	—	—	—	**Да**
Вариаций алгоритмов использования	< 5	< 10	0	**24**
Стоимость	$\approx$5600 руб./мес.	$\approx$3300 руб./мес.	Не известно	**0 руб. / мес.**

[1] Релиз MagicCodeGenerator ещё не состоялся.

3.2. Источники для изучения

Источниками для изучения тематических материалов, отслеживания свежих обновлений информации и публикаций будут служить:

- Топик “Блокчейн” на Researchgate [1]
- Топик “Криптография” на Researchgate [2]
- Книга Great Chain of Numbers (Swanson, Tim) [29]
- Тред “r/Blockchain” [6]
- Тред “r/CryptoCurrency” [9]
- Статьи на Medium [4]
- Статьи на Wikipedia [35]

3.3. Виды распределённых реестров

Для классификации алгоритмов и протоколов по распределённым реестрам и их семействам, был проведён обзор используемых в настоящие дни их представителей.
Для разделения реестров на группы по признакам открытости и закрытости существует несколько подходов.

- Первый — канадский, основанный на публикации статьи [34] создателя криптовалюты Ethereum Виталика Бутерина. Автор разделяет 3 типа реестров:
  1. Публичный (Public), где каждый может принять участие в создании блоков, которое никем не контролируются и выполняется в свободном порядке;
  2. Приватный (Fully Private), где все транзакции отслеживаются и контролируются централизованной сущностью;
  3. Реестры консорциума (Consortium), где только избранные узлы цепи контролируют создание новых блоков.
- Второй — британский. Определения Сэра Марка Уолпорта, главного научного советника Соединённого Королевства, в докладе о распределённых реестрах [26], совпадают по своей сути с определениями Бутерина, но отличаются названиями:
  1. Unpermissioned public;
  2. Permissioned private;
  3. Permissioned public.
- Третий — **российский**. Часто, для избежания недопониманий и сложностей в определении, мировые эксперты используют понятия Публичный (Открытый) и Приватный (Закрытый) реестры. Российские эксперты не исключение. В своём докладе [39], Ольга Скоробогатова, заместитель председателя ЦБ РФ, разделяет реестры именно таким образом. Данное разделение будет использоваться в настоящей работе:
  1. Публичный (Открытый, Public, Open). В открытых реестрах нет контролирующей стороны, как это реализовано в закрытых реестрах, все транзакции происходят в свободном порядке, а для подтверждения легитимности транзакции используются специальные протоколы консенсуса (3.7)

2. Приватный (Закрытый, Private, Closed). Несмотря на то, что сама рассматриваемая технология является распределённой, элементы централизованности присутствуют в таких вариантах, как закрытые (3.3) реестры. Закрытым реестр может являться благодаря первичному блоку (в случае блокчейна), который будет использоваться. Любой узел может присоединиться к приватному реестру, если он знает адрес начального узла для синхронизации и идентификатор сети. Этот узел может выполнять любые действия в закрытом реестре: майнить, совершать транзакции, заключать контракты, и т.д.

   Такие реестры зачастую используюся фирмами, банками и т.д. для организации внутренних операций обмена и регистрирования.

3.4. Структура реестров

По своей внутренней структуре распределённые реестры делятся на блокчейны (3.4.1) и направленные ациклические графы (3.4.2). На рисунке 2 изображены виды современных распределённых реестров.

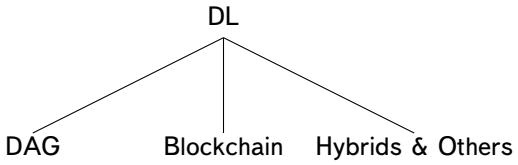

Рис. 2: Виды распределённых реестров

3.4.1. Блокчейны

Блокчейны — наиболее распространённые сегодня виды реестров. Набор транзакций (операций) собирается в один блок. Проходит определенное время и этот блок добавляется в общую цепочку. Для подтверждения транзакции существует множество протоколов консенсуса — стандартов, исходя из которых можно говорить, что создание и добавление данного блока имеет место быть. Подтверждение легальности добавления блоков в других представлениях распределённых реестров происходит с применением других алгоритмов и протоколов, поэтому далее для протоколов консенсуса будет использоваться слово “блокчейн” вместо “реестр” для исключения возможности двоякой интерпретации (на самом деле, в некоторых реализациях DAG [24] может применяться протокол консенсуса Proof-of-Work 3.7.1 для защиты от спама). Блокчейны завоевали стратегические позиции на площадке распределённых реестров и “прошли проверку временем”, но появляются новые технологии, такие как DAG и другие, призванные решить некоторые недостатки сетей блокчейн, речь о которых пойдёт позднее.

3.4.2. Направленные ациклические графы (DAGs)

В DAG все новые записи добавляются в общую цепочку (корректнее — граф) асинхронно. История записи операций выглядит как направленный ациклический граф ([38]). DAG топологически отсортирован так, что каждое ребро направлено от более раннего ребра к более позднему.

3.4.3. Плюсы и минусы DAG

Рассмотрим некоторые преимущества и недостатки DAG по отношению к блокчейнам.
Плюсы:

- Масштабируемость
- Мгновенные транзакции
- Отсутствие либо чрезмерно малые (незаметные) комиссии за переводы

Эти плюсы открывают дорогу для огромного количества микротранзакций, делая систему пригодной, для, например, интернета вещей.
Минусы:

- Возможные в будущем проблемы с масштабируемостью
- Нет подтверждённой учёными информации о защите от взлома основанных на DAG систем

3.5. Алгоритмы

Рассмотрим некоторые криптографические алгоритмы, которые используются в описанных системах.

3.5.1. SHA-256

Используется в таких реестрах как: **Bitcoin, ILCoin**

Алгоритм хэширования SHA-256, работая с данными, разбитыми на кусочки по 512 бит (64 байт), смешивает их криптографически и выдаёт 256-битный (32 байта) хэш — уникальную (почти: [15]) сигнатуру входного текста.

3.5.2. ECDSA

Используется в таких реестрах как: **Bitcoin**

Алгоритм цифровой подписи ECDSA (Elliptic Curve Digital Signature Algorithm) — криптографический алгоритм, используемый в целях гарантировать, что узлы сети могут быть использованы только их законными владельцами. Использует концепты публичного и приватного ключа. В алгоритме публичный ключ — это уравнение для эллиптической кривой (Рис. 3) и точка, лежащая на этой кривой. Приватный ключ — это простое число.

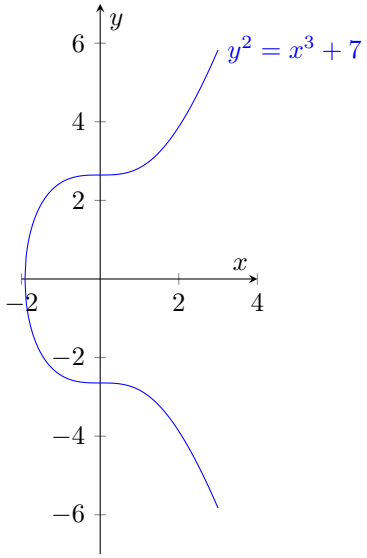

Рис. 3: Эллиптическая кривая

3.6. Прочие алгоритмы

Помимо стандартных криптографических алгоритмов шифрования, генерации электронной подписи и случайных чисел, в распределённых реестрах повсеместно используются алгоритмы, обеспечивающие сокрытие персональных данных и адресов отправителей и получателей, алгоритмы по защите и обфусцированию данных, отправляемых по небезопасным каналам связи.

3.6.1. Ring Signatures

Используется в таких реестрах как: **Monero, Particl, Bytecoin**

Представлен впервые в Декабре 2012 [32] Cryptonote. Вторая версия документа Cryptonote v2 — Октябрь 2013 [33]. Такая криптовалюта как Monero использует технологию кольцевой подписи для защиты конфиденциальности пользователя при проведении транзакций. Кольцевые подписи скрывают информацию об отправителе, заставляя отправителя подписывать транзакцию с помощью подписи, которая может принадлежать нескольким пользователям. Это делает транзакцию неотслеживаемой.
Интересно рассмотреть вариацию данного алгоритма RingCT, которая будет рассмотрена далее (3.6.6)

3.6.2. Stealth Addressess

Используется в таких реестрах как: **Monero, Particl, Bytecoin**

Представлены в том же документе. Stealth addresses позволяют получателю предоставлять для получения один адрес, который генерирует другой публичный адрес для средств, которые будут получены каждый раз при отправке на него. Это делает транзакцию также, неотслеживаемой.

Поскольку алгоритмы определены в одной работе и имеют схожие цели, можно, объединив их, назвать основные плюсы и минусы:
Плюсы

- Обеспечивает конфиденциальность отправителя и получателя
- Конфиденциальность может гарантироваться по умолчанию
- Проверенная технология
- Не требует каких-либо третьих сторон

Минусы

- Конфиденциальность не очень эффективна, имея недостаточный объем хранилища
- Не скрывает информацию о самой транзакции (если алгоритм используется сам по себе, не объединяясь с другими)

3.6.3. Coinjoin, coinshuffle

Используется в таких реестрах как: **Dash**

Разработчик Биткоина Грегори Максвелл предложил набор решений для обеспечения конфиденциальности в распределённых реестрах и криптовалютах, первым из которых являлся CoinJoin. [18]. CoinJoin (иногда называемый CoinSwap) позволяет нескольким пользователям объединять свои транзакции в одну транзакцию, получая входы от нескольких пользователей, а затем отправляя свои выходы нескольким пользователям, независимо от того, от кого в группе пришли входы. Таким образом, получатель получит назначенную сумму, но будет невозможным отследить транзакцию вплоть до отправителя.
Плюсы

- Обеспечивает конфиденциальность отправителя и получателя
- Простота реализации на любой криптовалюте
- Легковесная реализация в коде
- Проверенная технология

Минусы

- Суммы смешанных транзакций могут быть высчитаны

3.6.4. Sigma Protocol

Используется в таких реестрах как: **Zcoin**

Sigma Protocol — активно исследуется командой Zcoin с 2018 года для замены протокола Zerocoin zk-Snarks (zero knowledge succinct non-interactive arguments of knowledge) для того, чтобы не требовалась доверенная преднастройка. [13]. Существует возможная замена zk-Snarks, называемая zk-Starks, другая форма доказательства нулевого знания, которая может сделать доверенную преднастройку ненужной для монет с нулевым знанием.

3.6.5. Zerocash

Используется в таких реестрах как: **Zcash, Horizen, Komodo, Zclassic, Bitcoin Private**

В мае 2014 года был создан нынешний преемник протокола Zerocoin, Zerocash. Он улучшил концепцию Zerocoin, воспользовавшись уже известным zk-Snarks. В отличие от Zerocoin, который скрывал происхождение оригинальных монет и историю платежей, Zerocash был быстрее, с меньшими размерами транзакций и скрывает информацию о транзакции отправителя, получателя и суммы транзакций. Zcash-первая криптовалюта, реализовавшая протокол Zerocash в 2016 году. [36]

3.6.6. CT, RingCT

Используется в таких реестрах как: **Monero, Particl**

Confidential Transactions были представлены обществу в 2015 году [19] Было предложено скрыть сумму транзакции и тип передаваемых данных (например, депозиты, валюты, акции), чтобы только отправитель и получатель знали о сумме (до тех пор, пока они не решат сделать сумму общедоступной). В алгоритме используется гомоморфное шифрование для шифрования входов и выходов.

Затем появились Ring Confidential Transactions [22]. RingCT сочетает использование кольцевых подписей для сокрытия информации об отправителе с использованием конфиденциальных транзакций (которые также используют кольцевые подписи) для сокрытия сумм. В работе описан новый тип кольцевой подписи — A Multi-layered Linkable Spontaneous Anonymous Group, многослойная связанная спонтанная анонимная групповая подпись, которая “позволяет скрывать суммы, происхождение и назначение транзакций, с разумной эффективностью и качественной проверкой, с возможностью не-доверительной генерации монет”.

3.6.7. MimbleWimble

Используется в таких реестрах как: **Grin**

Mimblewimble [23] использует концепцию Confidential Transactions для скрытия сумм, а для доказательства права на собственность, смотрит на закрытые ключи и информацию о транзакциях, а не на адреса, также, связывает транзакции вместе, а не рассматривает их отдельно на блокчейне. Grin — это криптовалюта, находится в состоянии разработки и применяет Mimblewimble.

3.6.8. Zexe

В Октябре 2018 Zexe был представлен новый криптографичекий концепт — “децентрализованные частные вычисления” (decentralized private computation) [37]. Он позволяет пользователям распределённых реестров “выполнять автономно вычисления у себя на машинах, результатами которых будут настоящие транзакции”, а также сохраняет суммы транзакций скрытыми и позволяет проверять транзакции в любое время независимо от вычислений, выполняемых в интернете.

3.7. Протоколы консенсуса

Протокол консенсуса — стандарт, описывающий правила создания блоков в открытых распределённых реестрах. В работе рассмотрены все протоколы консенсуса, используемые в распространённых на сегодняшний день распределённых реестрах.

3.7.1. Proof of Work (PoW)

Используется в таких реестрах как: **Bitcoin, Litecoin, Dogecoin**

Впервые концепция Proof of Work была описана в 1993 году в работе “Pricing via Processing, Or, Combatting Junk Mail, Advances in Cryptology” [11].

Идея авторов заключалась в следующем: чтобы получить доступ к общему ресурсу, пользователь должен вычислить некоторую достаточно сложную функцию, и так защитить ресурс от злоупотребления. В 1997 году Адам Бэк запустил проект Hashcash, посвященный той же защите от спама. Задача формулировалась следующим образом: «Найти такое значение x, что хеш SHA(x) содержал бы N старших нулевых бит».

Алгоритм PoW используется в открытых реестрах для создания новых блоков. Алгоритм хорошо проявляет себя для доказательства легитимности транзакции, выполняя которое, майнеры вознаграждаются процентом, соответствующим вычислительной сложности всей сети. Зарекомендовавшее себя средство, оно не является выгодным с точки зрения расхода природных ресурсов, поскольку объём энергозатрат для выполнения вычислений, необходимых на добавление одного блока, сопоставим с объемом энергии, потребляемым двумя средними американскими домами: “ Bitcoin transaction uses roughly enough electricity to power 1.57 American households for a day.” — [5]

3.7.2. Proof of Stake (PoS)

Используется в таких реестрах как: **Ethereum**

Алгоритм доказательства доли владения — основной конкурент предыдущего алгоритма. Вероятность формирования участником очередного блока в реестре пропорциональна доле, которую составляют принадлежащие этому участнику расчётные единицы данного реестра (напр., кол-во единиц криптовалюты) от их общего количества. Поскольку для принятия решения о том, кто будет являться создателем нового блока, нет необходимости в большом объёме вычислений, как в PoW, зачастую является выбором, когда экологический фактор поставлен на одно из главных мест.

3.7.3. Delegated Proof-of-Stake (DPoS)

Используется в таких реестрах как: **Oxycoin, Lisk, Ark**

Delegated PoS применяет PoS с расширенным функционалом. Поэтому остается быстрым (и даже ещё быстрее), не требует больших вычислительных мощностей по сравнению с алгоритмом Proof of Work (PoW). Суть DPoS состоит в том, что узлы сети методом голосования выбирают узел, который будет генерировать блоки. В этом случае работает правило: чем большим количеством монет обладает узлы, тем больший вес имеет ее голос. Правила начисления вознаграждения также определяются также участниками сети. В некоторых сообществах вознаграждение начисляется не только узлы, которой делегировали право генерировать блоки, но

и остальным участникам. Первая криптовалюта, в которой был применен алгоритм DPoS — Bitshares. Также он применяется в следующих проектах: EOS, Steemit.

3.7.4. Proof-of-Authority (PoA)

Используется в таких реестрах как: **POA Network**

Алгоритм доказательства полномочий — это ещё один алгоритм консенсуса, в котором транзакции проверяются специальными учетными записями (валидаторами) системы. То есть майнить могут только валидаторы. Как пример использования можно привести систему Kovan Testnet [12].

3.7.5. Byzantine Fault Tolerance (BFT)

Используется в таких реестрах как: **NEO (dBFT)**

Лэсли Лэмпорт в 1982 году [16] со своими соавторами представили общественности задачку о Византийских генералах, позже использованную для создания алгоритма консенсуса BFT.

В BFT, как и в PoA существуют валидаторы, и только им позволено совершать быстрые транзакции, управлять каждым состоянием сети и обмениваться сообщениями друг с другом, чтобы получить правильную запись транзакции и обеспечить честность. Данный алгоритм реализуется компанией Ripple, где валидаторы выбираются предварительно. Любой может быть валидатором — доверие устанавливается сообществом. В отличие от блокчейнов, основанных на PoW, блокчейны BFT не подвергаются нападению, если только сами пользователи сети не координируют атаку. BFT считается выгодным алгоритмом, поскольку он масштабируем и охватывает транзакции с низкой стоимостью, но, как и DPoS, внедряет компонент централизации.

3.8. Выводы по главе

В данной главе были рассмотрены основные и наиболее распространённые алгоритмы и протоколы в распределённых реестрах сегодняшнего дня. Поскольку в расчёт брались только популярные, этого должно хватить для понимания общей картины текущего криптопротокола, но работа нацелена на глубокое изучение. На рис. 4 [29] приведена структура используемых алгоритмов и протоколов в распределённых реестрах на 2014 год. На сегодняшней день многое поменялось и добавились новые алгоритмы и протоколы, появились новые распределённые реестры, не основанные на блокчейне. В качестве результата проведённого изучения алгоритмов и протоколов в данной работе, на рис.5 приведена классификация обновлённая, на Q2 2019 года. На ней очевидны изменения в количественную сторону.

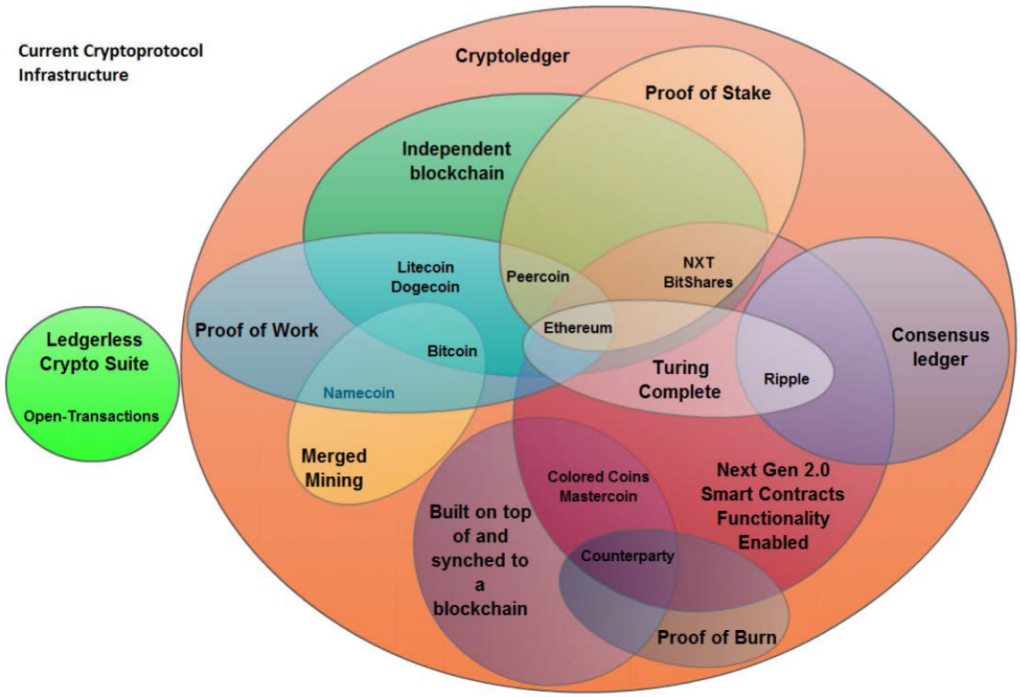

Рис. 4: Классификация на 2014 год [29]

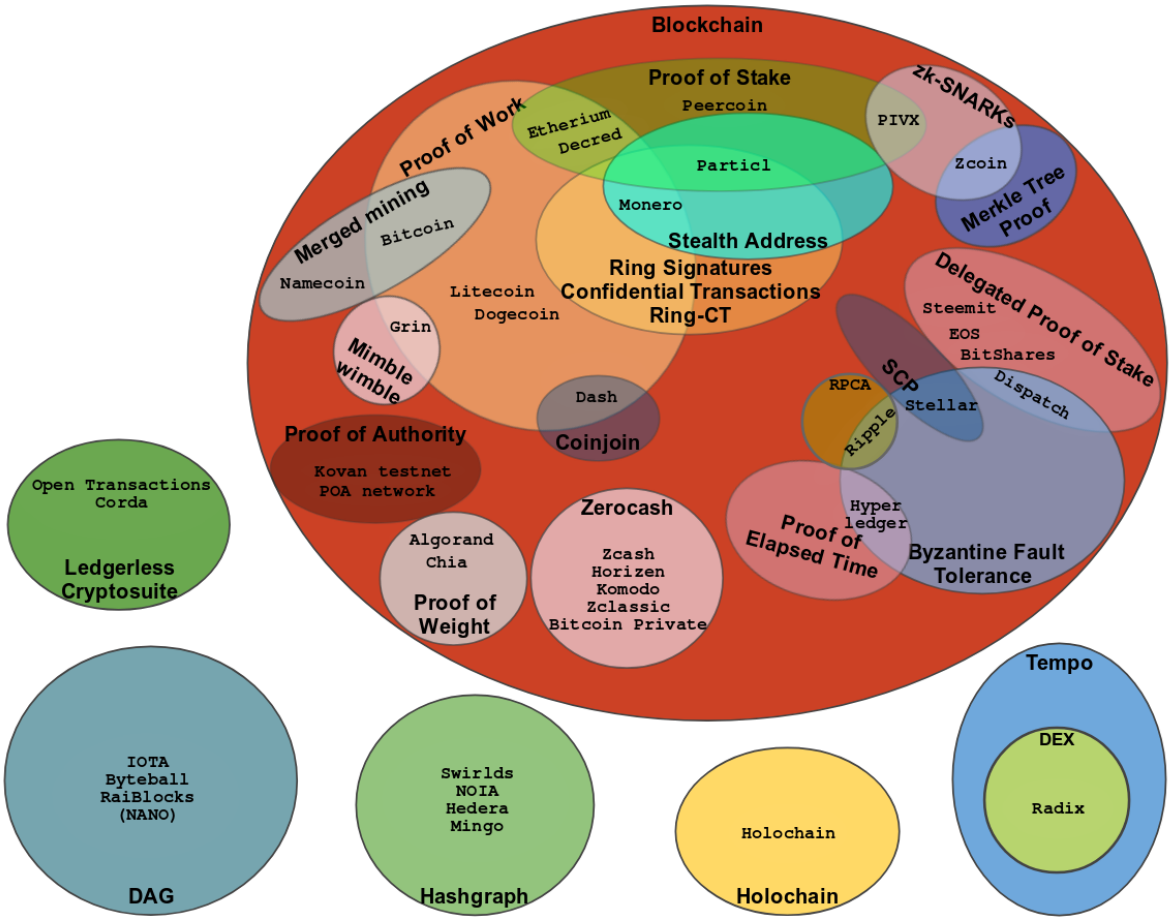

Рис. 5: Классификация на 2019 год

4. Глава 2. Проектирование сервиса: архитектура, процессы

В данной главе рассматриваются особенности реализации сервиса для автоматизации программирования с использованием технологии блокчейн на основе различных алгоритмов. В начале приведены архитектурные составляющие проекта **с обоснованием использования** средств реализации приведённой архитектуры, а в заключении собраны выводы по главе.

4.1. Архитектурные особенности

4.1.1. Общая структура проектного решения

Глобально проект состоит из двух приложений. Первый отвечает за интерактивное взаимодействие с пользователем и генерацию кода второго с использованием указанных пользователем алгоритмов (далее — компоновщик). Вторая — это имплементация блокчейна (далее — реализация блокчейна). Приложение содержит код для кошелька (wallet.py) и майнера (miner.py) с определённой функциональностью. Код второго приложения структурирован для удовлетворения нужд использования указанных пользователем методов. Методы и классы генерируются at-runtime первого приложения.

4.1.1.1 Язык программирования

Использованный язык программирования — **Python** верисии 3.6.5.
Рассматриваемые аналоги: C, Java

Выбран вследствие своей универсальности применения относительно (а) алгоритмов, (б) платформы для запуска; а также простоты реализации побочных, вспомогательных компонент, посредственно относящихся к данному проекту. Реализация их на других языках была бы необходимостью — и, в следствии их посредственного отношения к проекту, необоснованной тратой временного ресурса.
*Реализации алгоритмов **keccak-256** и **keccak-512** были переписаны с Python2 на Python3.6.5 для совместимости с данной программой.*

4.1.1.2 Тип приложения

Приложение является консольной утилитой, которая может быть установлена в систему семейства GNU/Linux при помощи программы **python3-pip**.
Рассматриваемые аналоги: Приложение с графическим интерфейсом, приложение в веб-интерфейсом

Отсутствие графического интерфейса обосновано отсутствием необходимости произведения манипуляций при помощи мыши, и отсутствием необходимости отображения графических изображений и другой информации. Интерактивный диалог производится посредством вывода в стандартный выход (*stdout*) консоли текста с опциями; а выбор пользователя регистрируется посредством считывания стандартного ввода (*stdin*).

4.1.1.3 Протокол обмена данными между компонентами

Выбранной опцией является передача **json** файлов посредством **http** протокола.
Рассматриваемые аналоги: **grpc** [14], **zmq** [10]

Обеспечивается использованием модуля *Flask* [3] и *json*. В сравнении с *grpc* и *zmq* протоколами, *http* представлялся наиболее подходящим вследствие своей популярности и удобства подключения и использования совместно с языком Python.

4.1.1.4 Хранилище

В качестве хранилища было реализовано **key-value** (ключ-значение) хранилище.
Рассматриваемые аналоги: **etcd** [31], **sqlite** [27]

Подключение сторонней библиотеки или базы данных сильно увеличило бы вес приложения в целом, а также добавило бы ещё несколько зависимостей. Вследствие этого было решено придумать импровизированное key-value хранилище на основе структуры данных словарь (dictionary). Реализация и функциональность описана в настоящем техническом задании (Приложение 1).

4.1.1.5 Автообновление

Задачей автообновления занимается UNIX утилита **cron**.
Рассматриваемые аналоги: Импровизированный планировщик событий

В связи с существованием качественного и надёжного решения в лице **cron**'а, было решено использовать его. Настроен на сервере. Подробнее — в п. 5.4.0.2

4.1.1.6 Continuous integration

В качестве средства CI был выбран **Shippable** [25].
Рассматриваемые аналоги: **TravisCI** [30], **CircleCI** [8]

Данное средство распространяется с возможностью использовать бесплатную версию программы, поэтому выбор пал на неё. *Необходимость* в сервисе CI выражается в том, чтобы данное приложение генерировало рабочие стабильные коды даже после обновления используемых реализаций алгоритмов, что не зависит от разработчика.

4.1.1.7 Конфигурация

Конфигурационный файл программы написан на языке **YAML** [28].
Рассматриваемые аналоги: **JSON**, **Plain text**

Выбран *YAML* по причине хорошей поддерживаемости в Python и своей эстетической непритязательности по сравнению с аналогами.

4.1.2. Архитектура компоновщика

Компоновщик — часть проекта, автоматизирующая процесс программирования и позволяющяя тем самым создавать готовые решения. Решением может быть рабочий код блокчейна с исполь-

зованием 24 вариаций алгоритмов.

4.1.3. Порядок работы компоновщика

Программно компоновщик был назван gsl. Основная команда с которой придётся иметь дело — init. Пример:

```
gsl --init --name myledger --path ~/tmp/gsl
```

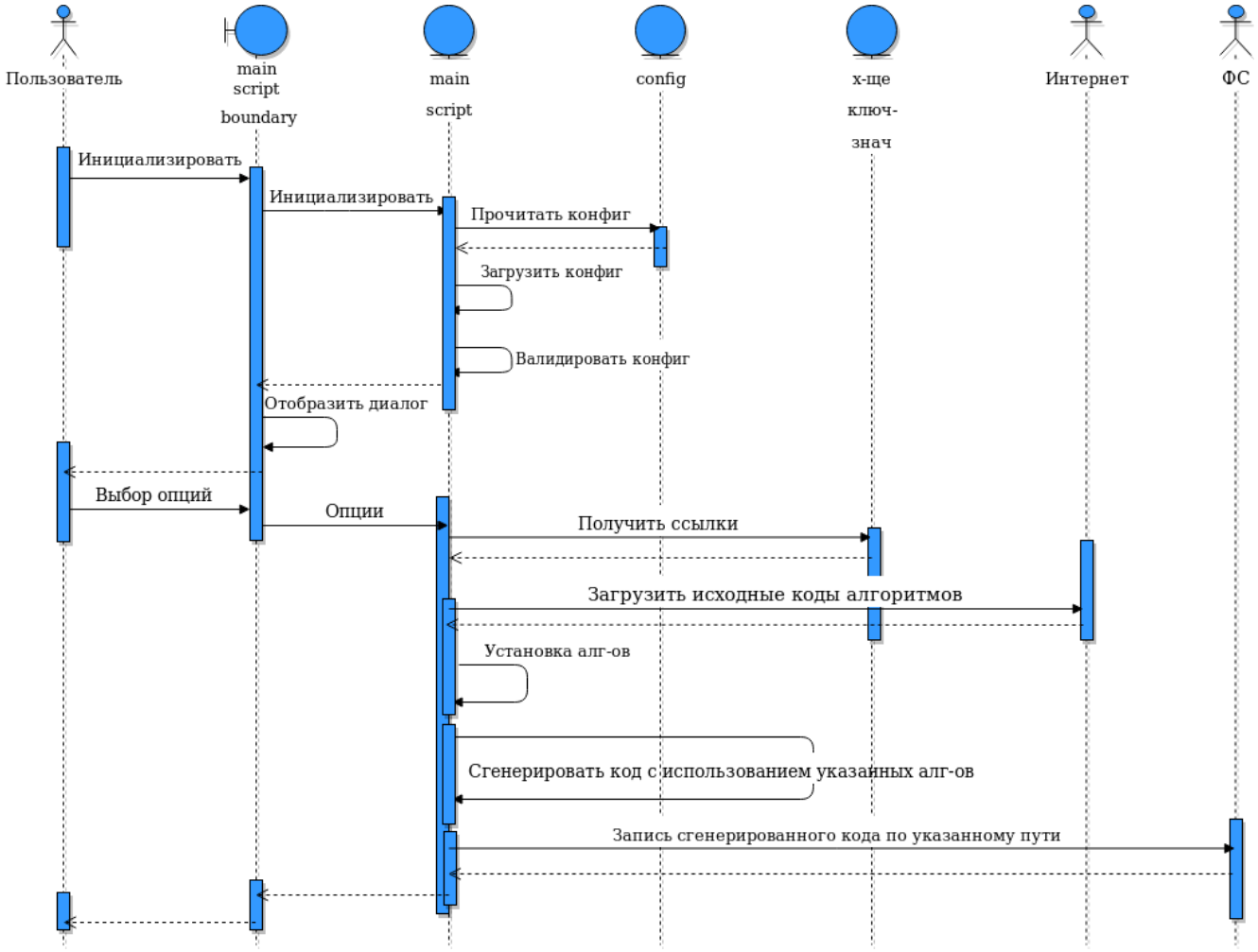

Рис. 6: Sequence диаграмма последовательности работы первой компоненты — компоновщика

По вызову этой команды происходят процессы, обозначенные на диаграмме последовательностей (Рис. 6). Зачитывается, загружается в программу и валидируется конфигурационный файл программы.

Затем начинается диалог с пользователем (Рис. 7).

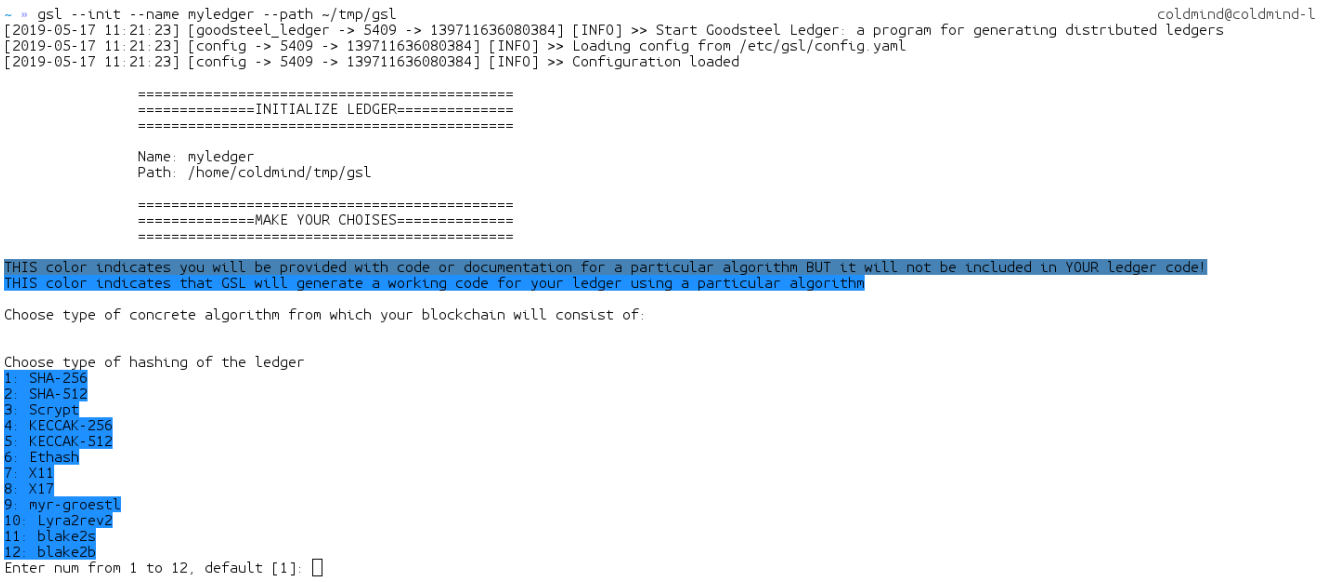

Рис. 7: Начало диалога выбора алгоритмов

В этом диалоге пользователь выбирает какие алгоритмы хэширования и цифровой подписи будут использованы в его будущей реализации блокчейна. После этого, пользователю предоставляется выбор остальных параметров блокчейна, по которым ему будут предложены справочные ссылки для изучения (Рис. 8). После выбора происходит установка данных библиотек и генерация кода реализации блокчейна по указанному пути (Рис. 9).

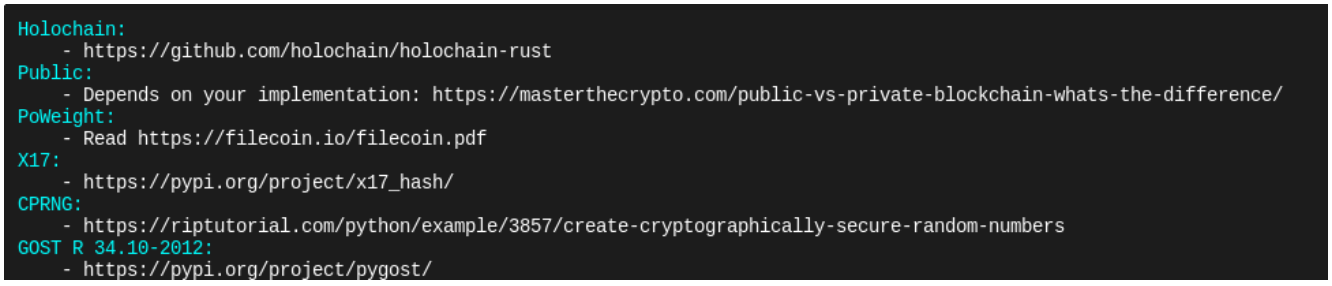

Рис. 8: Справочная информация по выбранным параметрам

4.1.4. Архитектура реализации блокчейна

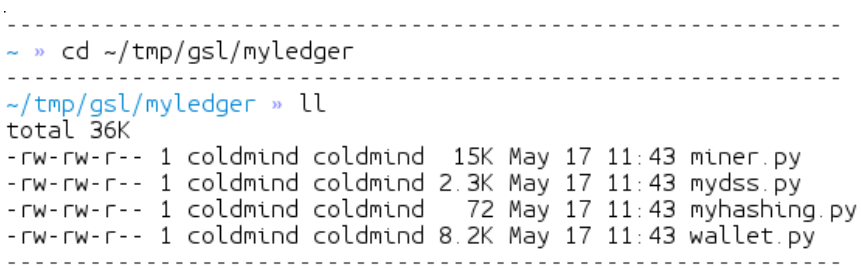

Рис. 9: Директория со сгенерированным кодом реализации блокчейна

Код реализации блокчейна запускается интерпретатором языка Python 3.6.5. Скрипты miner.py

и wallet.py запускаются без аргументов командной строки. Запустив miner.py (Рис. 10), можно запускать wallet.py (Рис. 11), в котором есть возможности:

1. Сгенерировать кошелёк: пару публичный-приватный ключи и записать их в файл
2. Отправить с одного кошелька на другой N условных еденицы
3. Провалидировать транзакции

Рис. 10: Лог запуска майнера

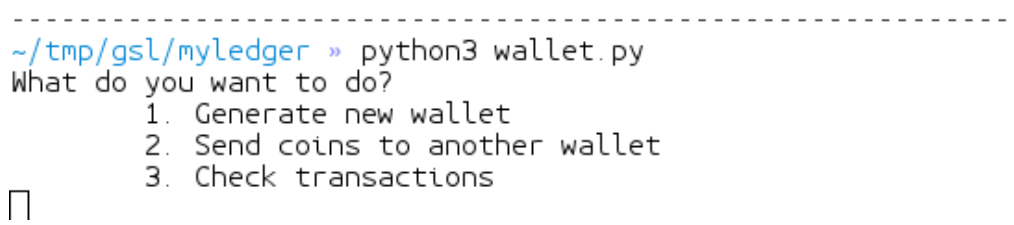

Рис. 11: Возможности кошелька

Генерация пары ключей, а также хэширование записей происходит посредством использования выбранных ранее пользователем алгоритмов.

4.1.5. Работа с данными

Исходный код алгоритмов хранится в директории src/altorithms/hashing и src/altorithms/digital_signature. Он собирается полным проходом в сеть по ссылкам, расположенными в импровизированном key-value хранилище (описано в п. 5.5, а также в настоящем техническом задании — Приложение 1). Данная процедура происходит при автообновлении алгоритмов на сервере каждый день в 21:00 (Рис. 12). После процедуры автообновления, пользователи могут по желанию обновить свою версию программы и использовать более свежий код.

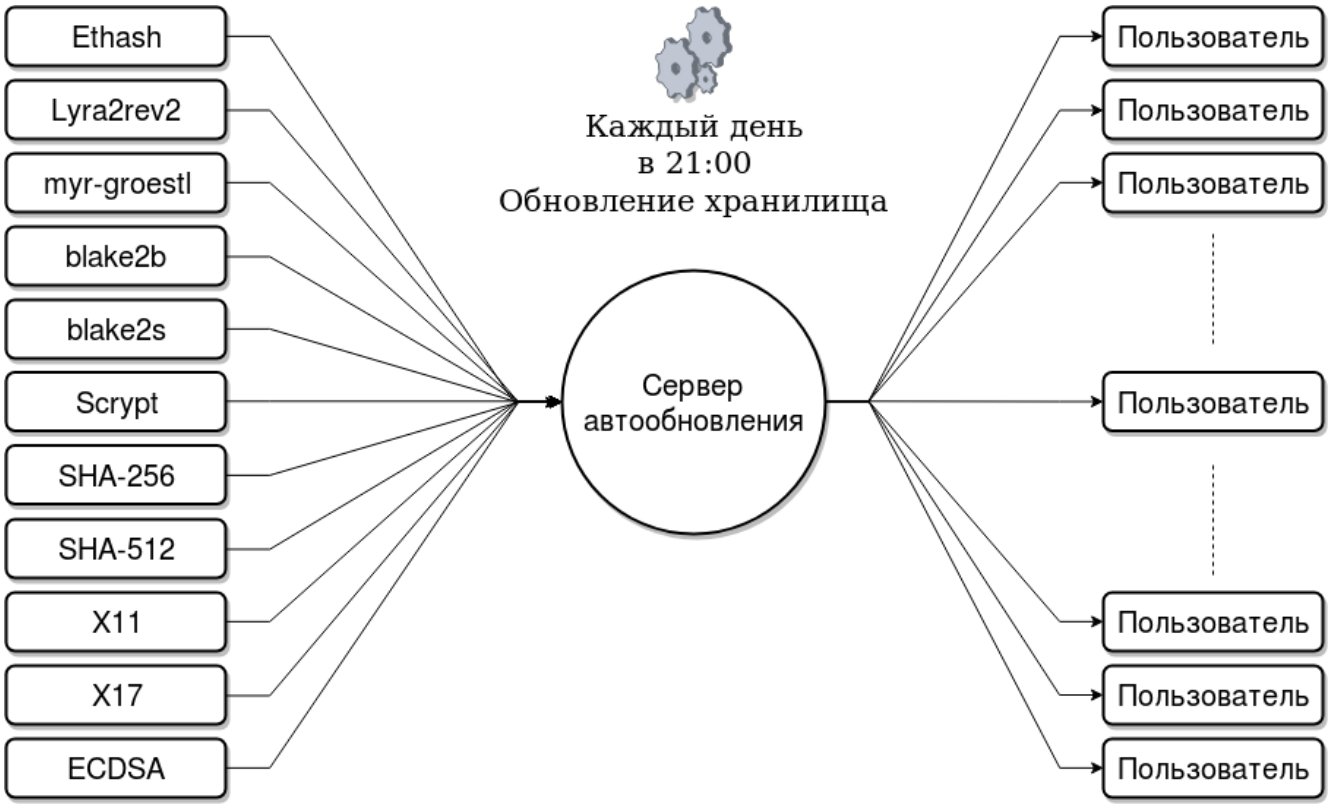

Рис. 12: Процесс работы сервера автообновления

4.2. Выводы

Приведённые решения в области автоматизации программирования позволяют максимально быстро и просто получить готовый год, который можно встроить в своё приложение, либо использовать в изучяющих целях.

В данной главе были описаны архитектурные составляющие проекта, и была затронута последовательность работы с ними. Был обоснован выбор каждой компоненты реализации решения.

5. Глава 3. Программная реализация

5.1. Функциональные требования

К функциональным требованиям приложения компоновщик относятся:

1. Возможность вывода на консоль вариантов выбора по категориям:
   (a) Отображение возможных структур реестра
   (b) Отображение возможных типов открытости реестра
   (c) Отображение возможных алгоритмов консенсуса
   (d) Отображение возможных алгоритмов хэширования
   (e) Отображение возможных алгоритмов генерации случайных чисел
   (f) Отображение возможных алгоритмов цифровой подписи
2. Генерирование значений для выбора “по умолчанию”
3. Возможность записать выбора пользователя
4. Возможность установки загруженных алгоритмов на ФС машины без исключительных root прав
5. Возможность генерировать код по указанной директории
6. Возможность замера времени работы выбранных алгоритмов
7. Возможность просмотра справочной информации по остальным параметрам реестра
8. Вывод информации в цвете, обозначающий степень поддержки программой алгоритма

К функциональным требованиям приложения реализация блокчейна (wallet.py)относятся:

1. Возможность генерации “адреса кошелька” — пары приватный + публичный ключ
2. Использование в качестве алгоритма цифровой подписи выбранный пользователем
3. Использование в качестве алгоритма хэширования выбранный пользователем
4. Возможность записи данных “адреса кошелька” на ФС машины
5. Возможность отправки от одного пользователя другому условное количество условной криптовалюты
6. Возможность получения всей цепочки транзакций, которые были проведены за текущую сессию путём вызова API функции майнера

К функциональным требованиям приложения реализация блокчейна (miner.py)относятся:

1. Возможность принимать json сообщения по http протоколу
2. Использование в качестве алгоритма цифровой подписи выбранный пользователем
3. Использование в качестве алгоритма хэширования выбранный пользователем
4. Возможность выполнения proof-of-work алгоритма
5. Возможность добавления блока в цепочку

5.2. Описание реализации процесса генерации кода компоновщиком

Процесс состоит из нескольких частей:

1. Сбор опций пользователя при помощи интерактивного диалога
2. Поиск опций в хранилище
3. По путям из хранилища программа осуществляет поход в интернет и загрузку кодов алгоритмов
4. Загруженные алгоритмы записываются на ФС машины
5. Загруженные алгоритмы устанавливаются в систему
6. На ФС записываются функции, обеспечивающие совместимость выбранных пользователем алгоритмов и реализации блокчейна
7. На ФС записывается реализация блокчейна

Процессы будут описываться по порядку, начиная со сбора опций.

5.2.1. Сбор опций пользователя

Начинается с отображения приветственного и инструкционного сообщения (Рис. 13)

```
~ » gsl --init --name myledger --path ~/tmp/gsl                                                                    coldmind@coldmind-l
[2019-05-17 11:21:23] [goodsteel_ledger -> 5409 -> 139711636080384] [INFO] >> Start Goodsteel Ledger: a program for generating distributed ledgers
[2019-05-17 11:21:23] [config -> 5409 -> 139711636080384] [INFO] >> Loading config from /etc/gsl/config.yaml
[2019-05-17 11:21:23] [config -> 5409 -> 139711636080384] [INFO] >> Configuration loaded

                ================================================
                ==============INITIALIZE LEDGER==============
                ================================================

                Name: myledger
                Path: /home/coldmind/tmp/gsl

                ================================================
                ==============MAKE YOUR CHOISES==============
                ================================================

THIS color indicates you will be provided with code or documentation for a particular algorithm BUT it will not be included in YOUR ledger code!
THIS color indicates that GSL will generate a working code for your ledger using a particular algorithm

Choose type of concrete algorithm from which your blockchain will consist of:


Choose type of hashing of the ledger
1: SHA-256
2: SHA-512
3: Scrypt
4: KECCAK-256
5: KECCAK-512
6: Ethash
7: X11
8: X17
9: myr-groestl
10: Lyra2rev2
11: blake2s
12: blake2b
Enter num from 1 to 12, default [1]:
```

Рис. 13: Начало работы компоновщика: приветственное окно и первый набор алгоритмов

В листинге 5.2.1 описан процесс сбора опций пользователя. Из хранилища берётся список всех возможных опций, и согласно порядку типов алгоритмов, распечатываются в пронумерованном виде соответствующие опции. Пользователь затем вводит номер, размах значений которого соответствует опциям алгоритмов данного типа. Предусмотрено значение “по умолчанию” для каждой из опций — первое значение. Вывод алгоритмов раскрашивается — из реализованного в рамках данной работы **utils** были импортированы функции и константы для форматирования вывода. В случае ошибки в стандартный выход логгера выводится её сообщение.

```
for k, v in OPTIONS.items():
    print(f'\nChoose type of {k} of the ledger')
    if isinstance(v, list):
        for num, opt in enumerate(v):
            if 'hash' in k or 'digital' in k:
                if opt in TOINSTALL:
                    prefix = ASCIIColors.BACK_BLUE
                else:
                    prefix = ASCIIColors.BACK_LIGHT_BLUE
            else:
                prefix = ASCIIColors.ENDS
            print(f'{prefix}{num+1}: {opt}{ASCIIColors.ENDS}', end='\n')
        try:
            n = input(f'Enter num from 1 to {len(v)}, default [1]: ')
            n = 0 if n == '' else int(n) - 1
            if n < 0 or n >= len(v):
                raise ChooseError
            self.ledger_config[k] = n
        except Exception as e:
            __logger__.exception(str(e))
            return
    else:
        print(v)
```

Листинг 5.2.1: Процесс выбора опций пользователем

5.2.2. Поиск алгоритмов

На данный момент выбранные пользователем опции были получены, и записаны в переменную **options**. Поиском алгоритмов в хранилище, интернете и их установкой на ФС машины занимается класс **ProlificWriter**.

Процесс поиска ссылок для алгоритмов в хранилище происходит путём поиска соответствий переданных классу опций с полем хранилища *TOINSTALL*. Запись на ФС установленных алгоритмов происходит в методе **write**. Сначала пишутся алгоритмы хэширования и электронной подписи, а затем — код реализации блокчейна.

```
def write(self):
    # WRITE ALGORITHMS ITSELF
    #
    self._write_hashing_()
    self._write_digital_signature_()

    self._write_(wallet)
    self._write_(miner)


def _write_(self, script_to_write):
    name = f'{script_to_write.__name__.split(".")[-1]}.py'
    src_code = getsource(script_to_write)
    with open(os.path.join(self.path, name), 'w') as __fd__:
        __fd__.write(src_code)
    if self.timed:
        os.system('sed -ir "0,/def _timed/{s/_timed = .*/_timed = True/}" ' + os.path.join(self.path, name))
    if self.profd:
        os.system('sed -ir "0,/def _profd/{s/_profd = .*/_profd = True/}" ' + os.path.join(self.path, name))
```

```
def _write_hashing_(self):
    # INSTALLING PROCEDURE
    src_path = self._get_src_path_()
    path = os.path.join(src_path, get('TOINSTALL', self.opts['hashing']))
    self._install_(path)

    # WRITING PROCEDURE
    name = 'myhashing.py'
    type_ = self.opts['hashing']
    path = os.path.join(src_path, get('INTERFACES', type_), name)
    if not os.path.exists(self.path):
        os.makedirs(self.path)
    copyfile(path, os.path.join(self.path, name))

def _write_digital_signature_(self):
    # INSTALLING PROCEDURE
    src_path = self._get_src_path_()
    path = os.path.join(src_path, get('TOINSTALL', self.opts['digital signature']))
    self._install_(path)

    # WRITING PROCEDURE
    name = 'mydss.py'
    type_ = self.opts['digital signature']
    path = os.path.join(src_path, get('INTERFACES', type_), name)
    copyfile(path, os.path.join(self.path, name))
```

Листинг 5.2.2: Процесс поиска алгоритмов и записи на ФС машины

5.2.3. Установка

Установка алгоритмов, упомянутая до этого в листинге 5.2.2, происходит посредством запуска метода **_install_**, а при неудачной попытки — **_pip_install_**.

```
def _install_(self, path):
    """
    Installs with `python setup.py install`
    """
    os.chdir(path)
    subprocess.call([sys.executable, f'{path}/setup.py', 'install'])


def _pip_install_(self, package):
    subprocess.call([sys.executable, '-m', 'pip', 'install', package, '--user'])
```

Листинг 5.2.3: Код для установки алгоритмов на ФС машины

5.2.4. Запись реализации блокчейна

Запись на ФС методов-обёрток для совместимости выбранных пользователем алгоритмов и реализации блокчейна, содержится в листинге 5.2.2. За запись методов-обёрток отвечает метод **_write_hashing_** и **_write_digital_signature_**.

5.3. Описание реализации блокчейна

Реализация блокчейна делится на 2 модуля: wallet.py и miner.py. Описание каждого из них представлено в данной секции. Данные модули генерируются компоновщиком, описанным в п. 5.2. Компоновщик сгенерировал данные модули таким образом, что они используют реализации алгоритмов, выбранных пользователем.

5.3.1. wallet.py

Модуль, позволяющий выполнить по запуску со стороны пользователя одну из трёх функциональностей:

1. Создать новый кошелёк
2. Отправить средств
3. Валидировать транзакции

Модуль должен использовать алгоритмы, выбранные пользователем на этапе генерации. Поэтому в начале, алгоритм импортирует их (листинг 5.3.1). В противном случае, при невозможности их импортирования, модулем будут применены стандартные алгоритмы. Подобными для хэширования является **SHA-256**, а для цифровой подписи **ECSDA**.

```
try:
    import mydss
    dss = mydss
    if hasattr(dss, 'name') and hasattr(dss, 'bit'):
        alg_name = dss.name
        alg_bit = dss.bit
        try:
            from mydss import mydss
            dss = mydss
        except:
            dss = mydss
except:
    import ecdsa
    dss = ecdsa
    alg_name = 'ecdsa'
    alg_bit = '256'
```

Листинг 5.3.1: Импортирование выбранных пользователем опций (myhashing и mydss)

5.3.1.1 Создание нового кошелька

При создании нового кошелька, запрашивается имя пользователя, и генерируются публичный и приватный ключи. Далее, они записываются в файл **<name>.txt**. Генерация ключей осуществляется в методе generate_ECDSA_keys(ret=False). При генерации по умолчанию, используется алгоритм **ECSDA**. При успешной попытке использования указанного пользователем алгоритма, используется он. Для компактности записи, длинный публичный ключ кодируется в base64 и в таком формате записыватеся в файл. В дальнейшем, он будет декодирован для корректной процедуры отправки/верификации. Описанная процедура представлена в листинге 5.3.1.1.

```
def generate_keys(ret=False):
    if _timed:
        t1 = time.time()
    singningkey = dss.SigningKey().generate()
    private_key = singningkey.to_string()
    vk = singningkey.getverifyingkey()
    public_key = vk.to_string().hex()
    public_key = singningkey.to_string(pub=True)
    if _timed:
        t2 = time.time()
        _write_time(alg_name, 'Key pair generation', alg_bit, t2-t1)
    public_key = base64.b64encode(bytes.fromhex(public_key))
    if ret:
        return private_key, public_key.decode()
    filename = input('Name of addr: ') + '.txt'
    with open(filename, 'w') as f:
        f.write('PrvK: {0}\nWallet addr / PubK: {1}'.format(private_key, public_key.decode()))
    print('saved{0}'.format(filename))
```

Листинг 5.3.1.1: Генерация пары публичного и приватного ключей

5.3.1.2 Отправка средств

При отправки средств, используются стандартные подходы блокчейна. В модуле miner.py происходит (Листинг 5.3.1.2) основная обработка и добавление самого блока к общей цепочке. В рассматриваемом wallet.py, происходит лишь сбор всей необходимой для совершения транзакции в json payload и отправляется на URL в API модуля miner.py. Дальнейшая обработка отправленного запроса происходит в 5.3.2.1.

```
def _perform_transaction(from_, prv_key, addr_to, amount):
    len_prv = len(prv_key)

    if dss.name == 'gost' or len_prv == 64:
        signature, message = _sign_msg(prv_key)
        url = f'http://localhost:{_port}/mycoin'
        payload = {'from': from_,
                   'to': addr_to,
                   'amount': amount,
                   'signature': signature.decode(),
                   'message': message}
        headers = {'Content-Type': 'application/json'}

        res = requests.post(url, json=payload, headers=headers)
        print(res.text)
    else:
        print('Wrong address; Please try again.')
```

Листинг 5.3.1.2: Отправка сформированного блока в miner.py

5.3.1.3 Валидация трназакций

Валидация транзакций в скрипте wallet.py ограничивается отправкой в miner.py запроса на проверку блоков (Листинг 5.3.1.3).

```
def check_transactions():
    res = requests.get(f'http://localhost:{_port}/blocks')
    print(res.text)
```

Листинг 5.3.1.3: Отправка запроса в miner.py на проверку валидности транзакций

5.3.2. miner.py

Данный модуль запускается и работает как фоновый процесс во время отправки и валидации сообщений первого. Он выполняет такие операции как mine block (майнг блока) — процесс, в котором происходит вызов proof-of-work процедуры, после которой считается возможным добавить в цепочку блок.

Со стороны http api для wallet.py, он выполняет:

1. Принимает **POST** запрос на добавление нового блока в цепочку
2. Принимает **GET** запрос на проверку существующих блоков в цепочке

5.3.2.1 Создание нового блока

Создание нового блока происходит в несколько этапов.

Информация о входящей транзакции регистрируется в очереди NODE_PENDING_TRANSACTIONS и выводится на стандартный вывод (stdout) — Листинг 5.3.2.1. Процесс, запущенный в отдельном потоке исполнения ОС, целевой функцией которого является mine, получает изменения из очереди NODE_PENDING_TRANSACTIONS и запускает процесс майна нового блока — Листинг 5.3.2.1.

```
@app.route('/mycoin', methods=['GET', 'POST'])
def transaction():
    """Each transaction sent to this node gets validated and submitted.
    Then it waits to be added to the blockchain. Transactions only move
    coins, they don't create it.
    """
    if request.method == 'POST':
        new_mycoin = request.get_json()
        if validate_signature(new_mycoin['from'], new_mycoin['signature'], new_mycoin['message']):
            WAITING_TRANSACTIONS.append(new_mycoin)
            print("Got a new transaction")
            print("FROM: {0}".format(new_mycoin['from']))
            print("TO: {0}".format(new_mycoin['to']))
            print("AMOUNT: {0}\n".format(new_mycoin['amount']))
            return "Transaction submission successful\n"
        else:
            return "Transaction submission failed. Wrong signature\n"
    elif request.method == 'GET' and request.args.get("update") == MINER_ADDRESS:
        pending = json.dumps(WAITING_TRANSACTIONS)
        WAITING_TRANSACTIONS[:] = []
        return pending
```

Листинг 5.3.2.1: URL '/txion' в miner.py для создания нового блока

```
def mine(a, blockchain, WAITING_TRANSACTIONS):
    BLOCKCHAIN = blockchain
    WAITING_TRANSACTIONS = WAITING_TRANSACTIONS
    while True:
        if _timed:
            t1 = time.time()
        last_block = BLOCKCHAIN[len(BLOCKCHAIN) - 1]
        last_proof = last_block.data['proof-of-work']
        proof = proof_of_work(last_proof, BLOCKCHAIN)
        if not proof[0]:
            BLOCKCHAIN = proof[1]
            a.send(BLOCKCHAIN)
            continue
        else:
            WAITING_TRANSACTIONS = requests.get(MINER_NODE_URL + "/txion?update=" +
                MINER_ADDRESS).content
            WAITING_TRANSACTIONS = json.loads(WAITING_TRANSACTIONS)
            WAITING_TRANSACTIONS.append({
                "from": "network",
                "to": MINER_ADDRESS,
                "amount": 1})
            new_block_data = {
                "proof-of-work": proof[0],
                "transactions": list (WAITING_TRANSACTIONS)
            }
            WAITING_TRANSACTIONS = []
            new_block_index = last_block.index + 1
            new_block_timestamp = time.time()
            last_block_hash = last_block.hash
            mined_block = Block(new_block_index, new_block_timestamp, new_block_data, last_block_hash)
            BLOCKCHAIN.append(mined_block)
            try:
                print(json.dumps({
                  "index": new_block_index,
                  "timestamp": str(new_block_timestamp),
                  "data": new_block_data,
                  "hash": last_block_hash.decode()
                }) + "\n")
            except:
                print(json.dumps({
                  "index": new_block_index,
                  "timestamp": str(new_block_timestamp),
                  "data": new_block_data,
                  "hash": last_block_hash
                }) + "\n")
            a.send(BLOCKCHAIN)
            requests.get(MINER_NODE_URL + "/blocks?update=" + MINER_ADDRESS)
            if _timed:
                t2 = time.time()
                _write_time(hash_name, 'Mining one block', hash_bit, t2-t1)
```

Листинг 5.3.2.1: Процесс майнинга новых блоков

5.3.2.2 Валидация подписи

Валидация электронной подписи происходит в методе validate_signature. В нём вызываются методы алгоритма цифровой подписи. Алгоритм может быть либо выбранный пользователем на этапе генерации, или, если его не происходило, стандартным, то есть **ECSDA** с кривой SECP256k1 — Листинг 5.3.2.2.

```
def validate_signature(public_key, signature, message):
    if _timed:
        t1 = time.time()
    public_key = (base64.b64decode(public_key)).hex()
    signature = base64.b64decode(signature)
    vk = dss.VerifyingKey().from_string(public_key)
    try:
        res = vk.verify(signature, message.encode())
        if _timed:
            t2 = time.time()
            _write_time(alg_name, 'Verifying signature', alg_bit, t2-t1)
        return res
    except:
        return False
```

Листинг 5.3.2.2: Процесс верификации электронной подписи

5.4. Описание процесса автообновления

Принципиальная схема работы автообновления представлена в п. 4.1.5.
Автообновление настроено на отдельном сервере по расписанию. Конфигурация сервера представляет из себя VPS виртуальную машину, на которой установлена OS *Ubuntu 16.04 LTS* с версией ядра *4.4.0-148-generic*.

Расписание автообновления сконфигурировано при помощи стандартной UNIX утилиты **cron**. Конфигурация **crontab** представлена в листинге 5.4. Происходят там следующие процессы:

- Каждый день в 20:00 запускается скрипт updater.py, который получает актуальные изменения алгоритмов в локальную копию на серверную ФС;
- Каждый день в 21:00 запускается скрипт pusher.py, которые полученные изменения загружает в публичный репозиторий, делая обновлённые и актуальные данные доступными всем;
- Логи обновления и загрузки пишутся в updater.log и pusher.log соответственно.

```
00 20 * * * echo $(date) >> /home/coldmind/gsl/updater.log && cd /home/coldmind/gsl/src && ./updater.py
    >> /home/coldmind/gsl/updater.log 2>&1
00 21 * * * echo $(date) >> /home/coldmind/gsl/pusher.log && cd /home/coldmind/gsl && ./src/pusher.sh
    >> /home/coldmind/gsl/pusher.log 2>&1
```

Листинг 5.4: Конфигурация crontab -l

Код скрипта обновления updater.py представлен в листинге 5.4.0.1. Он использует реализованное key-value хранилище (4.1.5) для получения ссылок, по которым возможно обновление загруженных алгоритмов, и для каждой ссылки и пути записи в ФС (тоже получает из key-value хранилища), запускает скрипт pull_single.sh (Листинг 5.4.0.1).

5.4.0.1 updater

```
#!/usr/bin/env python3.6

import subprocess
from technologies import _kv_, get
toinstall = _kv_.toinstall
update_links = _kv_.update_links

for alg, src in update_links.items():
    p = subprocess.popen(['bash', 'pull_single.sh', f'{get(" toinstall ", alg)}', f'{src}'], stdout=subprocess.pipe)
    (result, error) = p.communicate()
    print(result.decode())
```

листинг 5.4.0.1: Скрипт обновления всех алгоритмов со внешних источников

Скрипт pull_single.sh (Листинг 5.4.0.1) представляет из себя поход по ссылке во внешний источник, выгрузку данных оттуда, и замещение новыми данными старые.

```
#!/usr/bin/env bash

if cd $1;
then
    cd ..
    name1=$(echo $1 | rev | cut -d/ -f1 | rev)
    name2=$(echo $2 | rev | cut -d/ -f1 | rev)
    if [[ $name1 == $name2 ]]
    then
        rm -rf $name1
        git clone $2
        cd $name2
        rm -rf .git*
    fi
fi
```

листинг 5.4.0.1: Скрипт обновления единичного алгоритма

5.4.0.2 pusher

Скрипт pusher.py используется для загрузки полученных изменений в сеть для общего доступа. Представлен в листинге 5.4.0.2. При неизвестной ошибке в процессе обновления или некорректной работе обновлённых алгоритмов, подключенный к репозиторию сервис Continuous Integration не позволит вступить изменениям в силу, вследствие чего, пользователи будут защищены от нерабочего кода.

```
#!/usr/bin/env bash

curr_date=$(date +%d_%m_%Y)

git add .
git commit -m "Update algorithms: $curr_date."
git push


# EOF
```

листинг 5.4.0.2: Скрипт для загрузки обновлённых данных

5.5. Описание реализации хранилища данных

Хранилище представляет собой импровизированное key-value хранилище, выбор среди альтернатив которого, описан в п. 4.1.1.4.

Данные, которыми располагает приложение, подразумевают внесение изменений лишь разработчиком приложения. Этими данными являются ссылки на сетевые адреса хранения исходных кодов алгоритмов, пути записи алгоритмов в ФС компьютера, и пути до функций-обёрток данных алгоритмов. Все представленные данные, при необходимости, правятся разработчиком, в следствие чего, хранилище имеет только метод **get** (Листинг 5.5), и является read-only. Entity-relationship диаграмма представлена на рисунке 14.

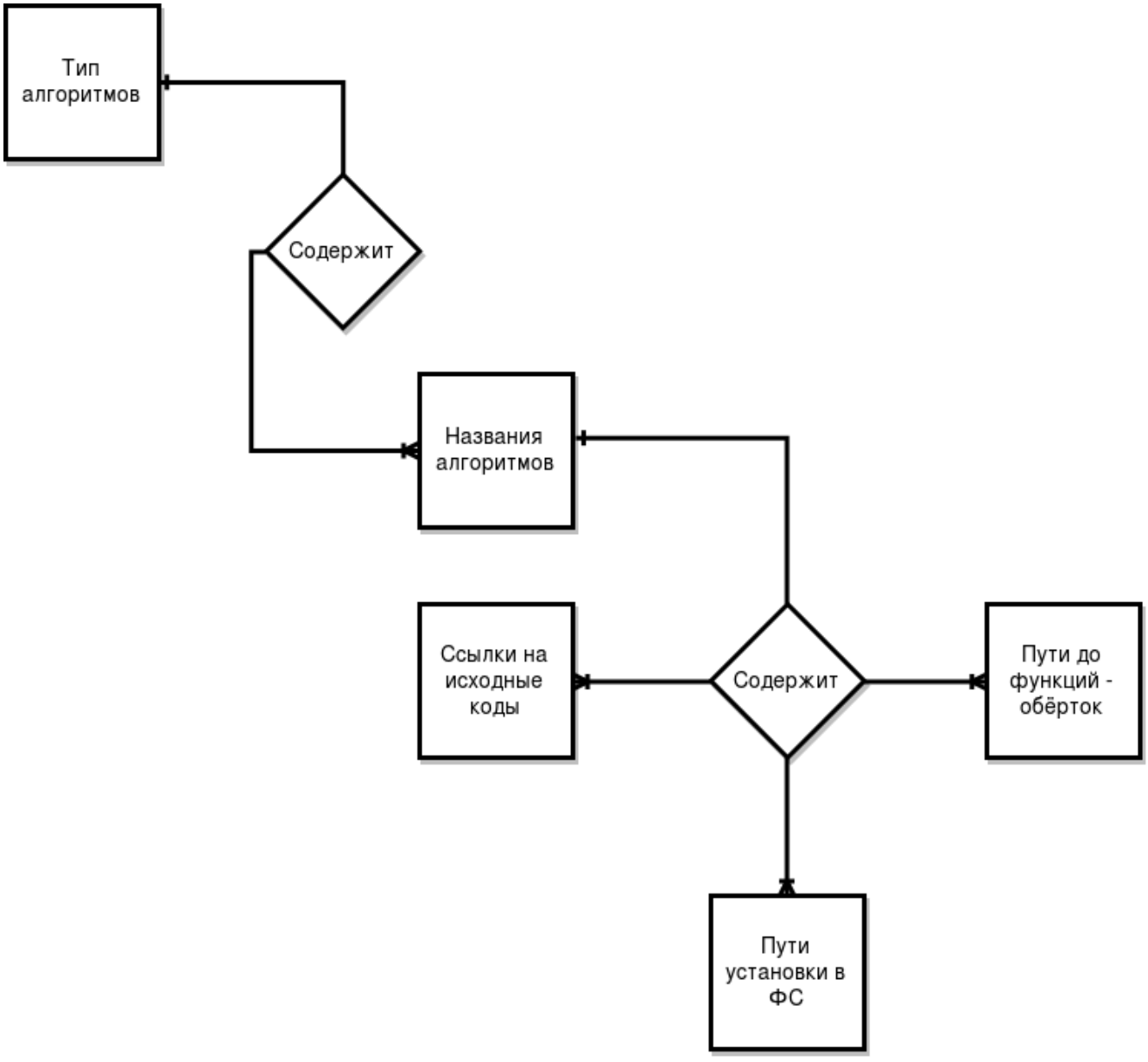

Рис. 14: Диаграмма отношений сущностей в реализованном хранилище

Метод **get** хранилища:

```
def get(name, val, default=None):
    return getattr(_kv_, name).get(val, default)
```

Листинг 5.5: код метода get хранилища

Код класса хранилища:

```
class _kv_(object):
    OPTIONS = {
            'hashing':  ['SHA-256', 'SHA-512', 'Scrypt', 'KECCAK-256',
                         'KECCAK-512', 'Ethash', 'X11', 'X17', 'myr-groestl',
                         'Lyra2rev2', 'blake2s', 'blake2b'],
#...
    }

    LINKS = { } # ....

    UPDATE_LINKS = { } # ....

    TOINSTALL = { } # ....

    INTERFACES = { } # ....
```

Листинг 5.5: Представление кода реализованного key-value хранилища

6. Глава 4. Проведение эксперементов

6.1. Анализ времени исполнения алгоритмов в реализации блокчейнов

Была предпринята попытка замерить время исполнения всех 24 вариаций алгоритмов на следующих операциях:

- Генерация пары ключей (публичный-приватный)
- Вычисления хэш-значения
- Электронная подпись сообщения
- Верификация сообщения
- Время работы Proof of Work
- Время майна одного блока

Были замерены время исполнений участков кода готовых реализаций блокчейнв, сгенерированных при помощи разработанного компоновщика. Замеры проводились при помощи модуля *time* в Python. Дальнейший анализ и построение графиков происходило в среде *Jupyter Notebook* с использованием модуля matplotlib.

Была составлена таблица с записями вида *[название алгоритма; функция; битрейт; время исполнения]* размером более 1500 записей.

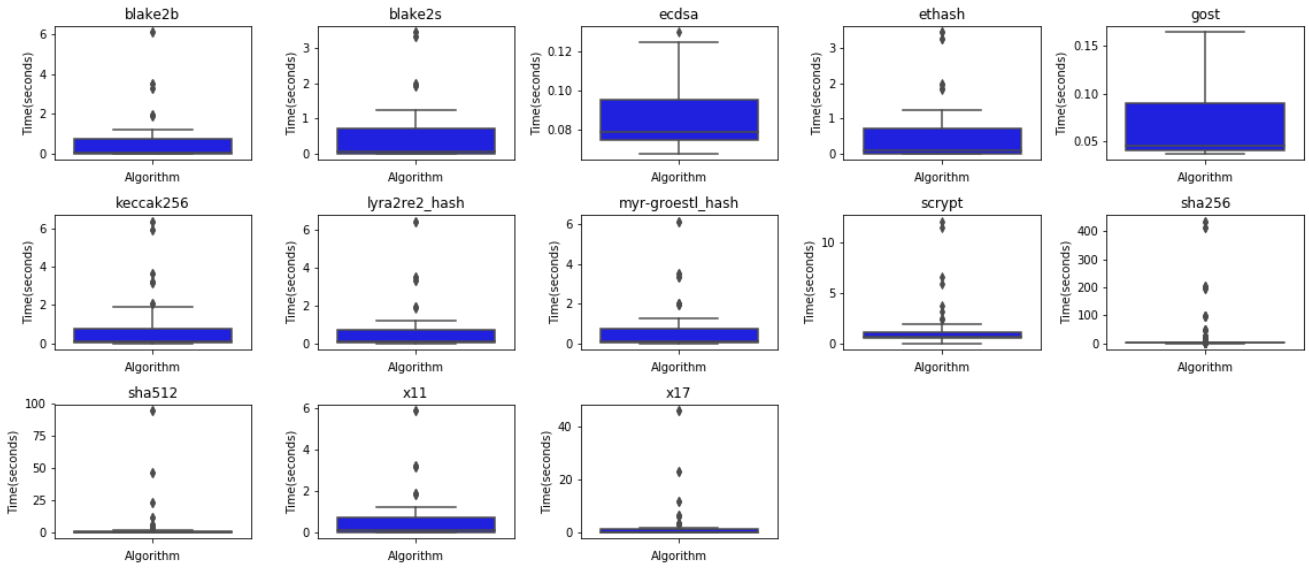

Рис. 15: Box-plot'ы для распределения алгоритмов по времени

Далее были построены распределения общего вида (Рис. 15), а также гистограммы распределения времени выполнения для алгоритмов цифровой подписи, разделяя их на вышеперечисленные процессы (Рис. 16):

И аналогичные данные были построены для алгоритмов хэширования (Рис. 17):

С целью сравнить время исполнения алгоритмов по одинаковым процессам, расположим их на одном графике гистограммы. На Рис. 15 с бокс-плотами были видны выбросы, поэтому

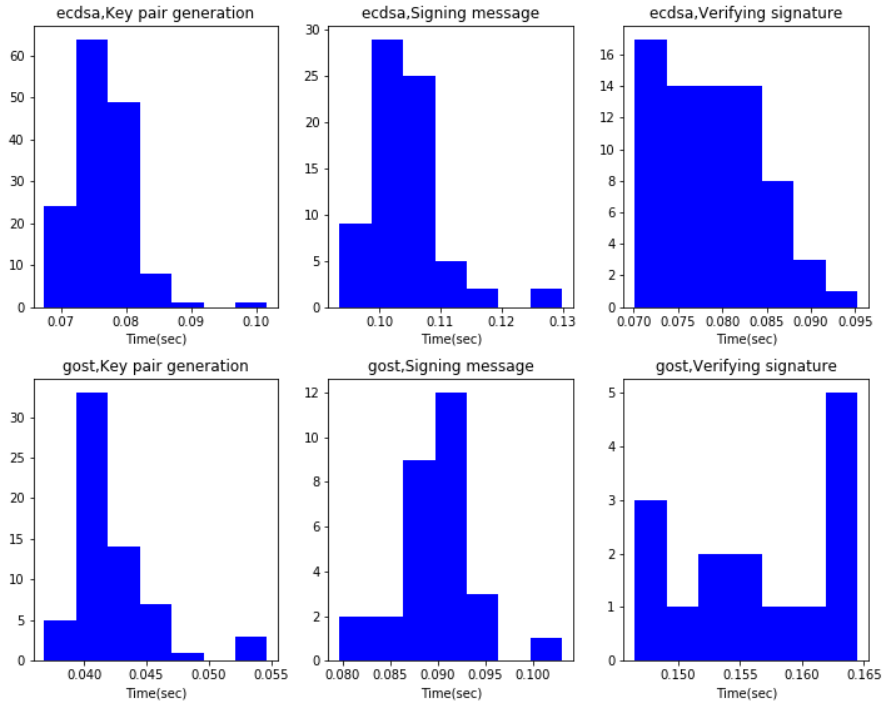

Рис. 16: Распределение времени выполнения среди алгоритмов цифровой подписи

для усреднения значений на последующих графиках, были взяты медианы значений, поскольку данные показатель более устойчив к выбросам, чем обычное среднее.

Самым быстрым алгоритмом хэширования *среди упомянутых*, исходя из проведённого анализа, является, SHA-256. Самым медленным из реализованных на языке СИ — X17. На данных графиках интересно заметить, что оба алгоритма, реализованные на языке Python, имеют гораздо большее время исполнения, по сравнению с реализованными на языке СИ. Алгоритм ГОСТ 34.10-2012 проигрывает международно известной ECSDA лишь во времени исполнения процедуры верификации (Рис. 19).

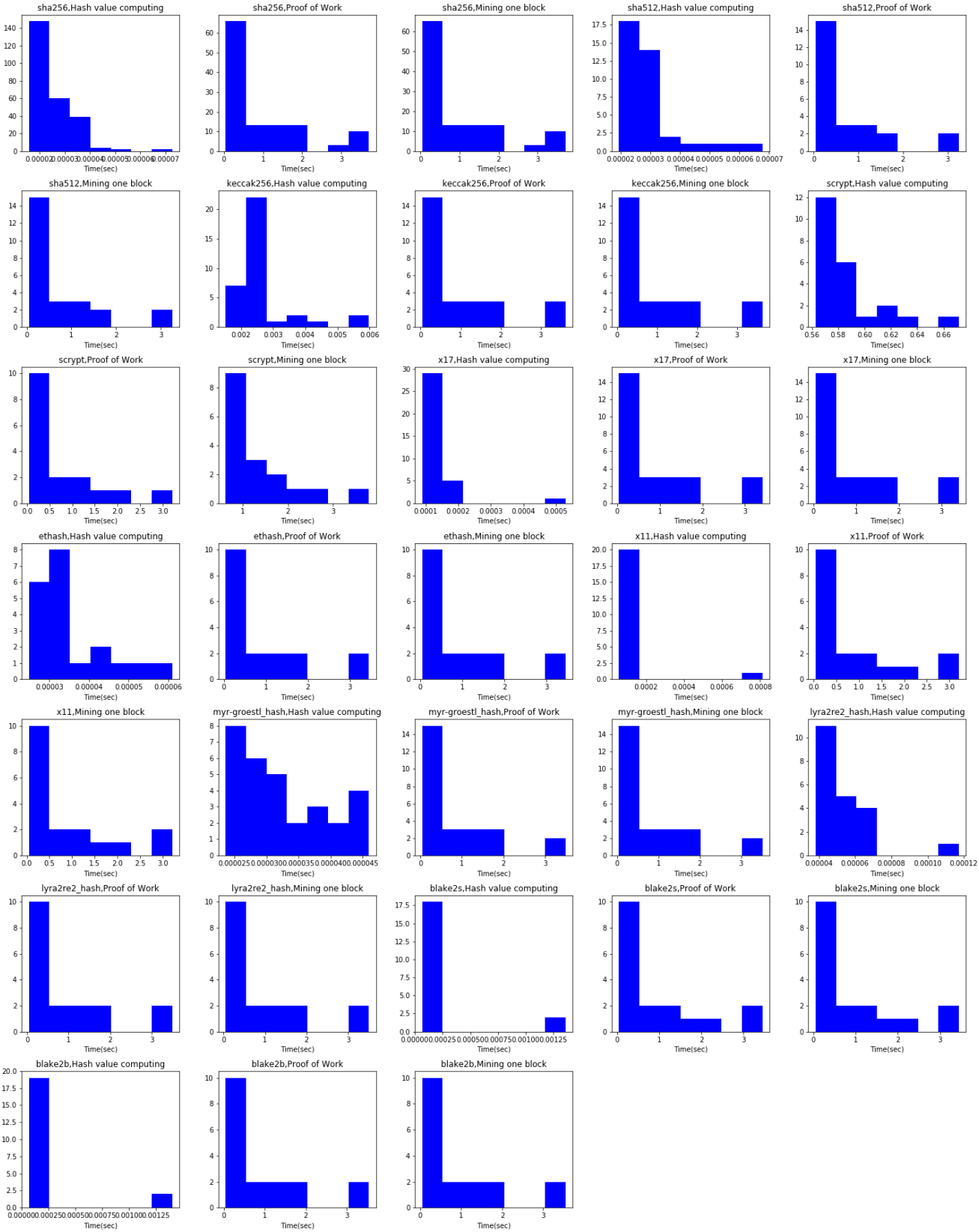

Рис. 17: Распределение времени выполнения среди алгоритмов хэширования

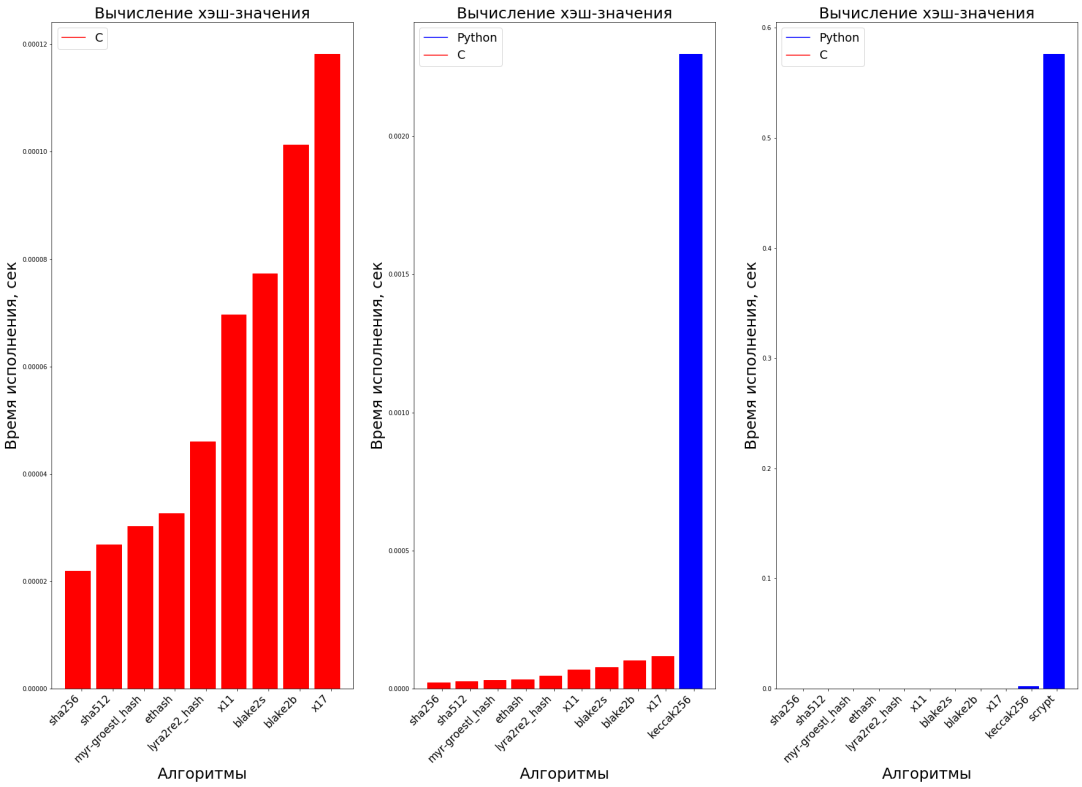

Рис. 18: Сравнение времени исполнения работ различных алгоритмов хэширования на одинаковых функциях

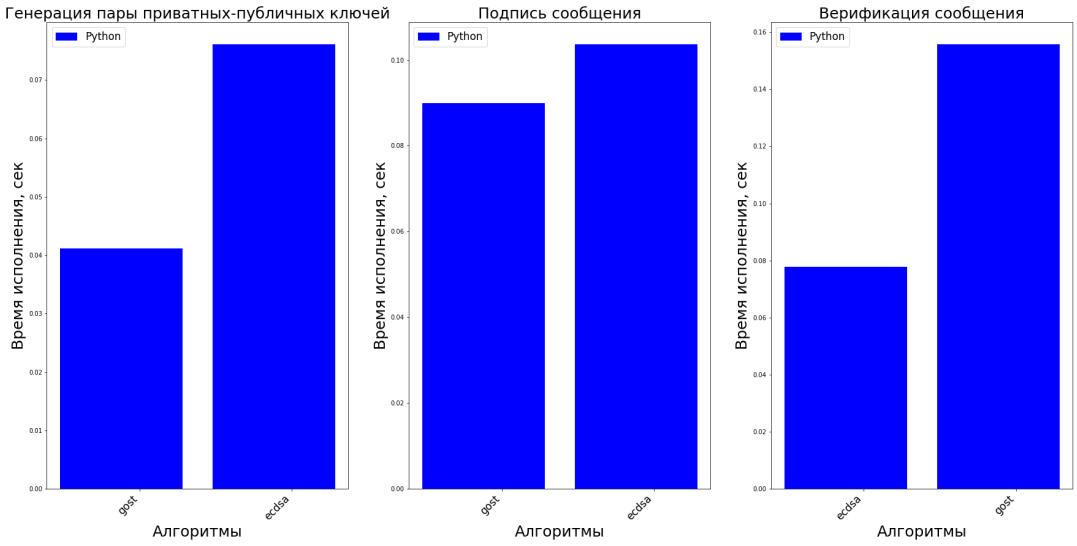

Рис. 19: Сравнение времени исполнения работ различных алгоритмов цифровой подписи на одинаковых функциях

6.2. Выводы

Рассмотренные временные показатели алгоритмов могут дать пользователю представление о том, каким будут временные показатели его блокчейна. Данное сравнение позволит пользователю

ориентироваться при выборе желаемой конфигурации.

7. Заключение

Было проведено исследование использования различных алгоритмов и протоколов в распределённых реестрах. Были найдены новые типы распределённых реестров, такие как Holochain, Hashgraph и Tempo. Это позволило решить проблему устаревшей информации по классификации алгоритмов и протоколов в распределённых реестрах.
Была разработана актуальная классификация использования алгоритмов и протоколов в распределённых реестров, в которой отражено современное многообразие технологичных подходов к решению проблем безопасности.

Было разработано гибкое, масштабируемое программное средство для автоматизации работы программирования. Разработан уникальный процесс по работе с исходными кодами алгоритмов, расположенных удалённо, их использованию и автообновлению. Налажена самоподдерживаемая система, не требующая вмешательства программиста после её первичной настройки. Приложение находится в публичном доступе и доступно к установке.

В то же время, в работу данного приложения может быть внесён ряд усовершенствований, возможными направлениями которых являются:

- Добавление возможности генерации года не только блокчейна, но и других реестров
- Добавление в реализацию блокчейна алгоритмов по защите приватности
- Улучшение временных характеристик алгоритмов, реализованных на Python путём имплементации их на языке СИ

К основным направлениям дальнейшей работы над исследовательской частью работы можно отнести:

- Исследование работы внутренней структуры новых реестров
- Сравненительный анализ блокчейна и новых реестров
- Поддержание разработанной классификации в актуальном состоянии

8. Список использованных источников

Список литературы

1. 279 questions in Blockchain | Science topic. — URL: https://www.researchgate.net/topic/Blockchain (дата обр. 20.05.2019).
2. 97 questions in Cryptocurrency | Science topic. — URL: https://www.researchgate.net/topic/Cryptocurrency (дата обр. 20.05.2019).
3. API — Flask 0.12.4 documentation. — URL: http://flask.pocoo.org/docs/0.12/api/%7B%5C#%7Dflask.Flask (дата обр. 20.05.2019).
4. *Ashish Kotsbtechcdac*. DAG will overcome Blockchain Problems DAG VS. BLOCKCHAIN. — 2018. — URL: https://medium.com/coinmonks/dag-will-overcome-blockchain-problems-dag-vs-blockchain-9ca302651122 (дата обр. 20.05.2019).
5. Bitcoin Is Unsustainable. — URL: https://motherboard.vice.com/en%7B%5C_%7Dus/article/ae3p7e/bitcoin-is-unsustainable (дата обр. 23.04.2019).
6. Blockchain. — URL: https://www.reddit.com/r/BlockChain/ (дата обр. 20.05.2019).
7. *Code Creator*. BlockChain. — URL: https://www.codecreator.com/index.php/blockchain/ (дата обр. 20.05.2019).
8. Continuous Integration and Delivery - CircleCI. — URL: https://circleci.com/ (дата обр. 20.05.2019).
9. Cryptocurrency news and discussions. — URL: https://www.reddit.com/r/CryptoCurrency/ (дата обр. 20.05.2019).
10. Distributed Messaging - zeromq. — URL: http://zeromq.org/ (дата обр. 20.05.2019).
11. *Dwork C., Naor M.* Pricing via Processing or Combatting Junk Mail // Advances in Cryptology — CRYPTO' 92. — 2007. — С. 139—147. — DOI: 10.1007/3-540-48071-4_10.
12. *Etherium*. TESTNET Kovan (KETH) Blockchain Explorer. — 2018. — URL: https://kovan.etherscan.io/ (дата обр. 23.04.2019).
13. *Groth J., Kohlweiss M.* One-out-of-many proofs: Or how to leak a secret and spend a coin // Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). — 2015. — Т. 9057. — С. 253—280. — ISSN 16113349. — DOI: 10.1007/978-3-662-46803-6_9.
14. gRPC. — URL: https://grpc.io/ (дата обр. 20.05.2019).
15. *Jim*. Hash codes are not unique. — 2012. — URL: http://blog.mischel.com/2012/04/13/hash-codes-are-not-unique/ (дата обр. 20.05.2019).
16. *Lamport L., Shostak R., Pease M.* The Byzantine Generals Problem // ACM Transactions on Programming Languages and Systems. — 2002. — Т. 4, № 3. — С. 382—401. — ISSN 01640925. — DOI: 10.1145/357172.357176.
17. Magic Code Generator – Tools for blockchain-based application development. — URL: https://magiccodegenerator.com/ (дата обр. 20.05.2019).
18. *Maurer F. K., Florian M.* Anonymous CoinJoin Transactions with Arbitrary Values. —.
19. *Maxwell G.* Confidential Transactions. — 2015.
20. *Microsoft Azure*. Blockchain Technology and Applications. — URL: https://azure.microsoft.com/en-us/solutions/blockchain/ (дата обр. 20.05.2019).
21. *Nakamoto S.* Bitcoin: A Peer-to-Peer Electronic Cash SyNakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Consulted, 1–9. // Journal for General Philosophy of Science. — 2008. — № 1. — С. 1—9. — ISSN 09254560. — DOI: 10.1007/s10838-008-9062-0. — arXiv: 43543534534v343453.
22. *Noether S., Mackenzie A., Research Lab T. M.* Ring Confidential Transactions // Ledger. — 2016. — Т. 1. — С. 1—18. — DOI: 10.5195/ledger.2016.34.
23. *Poelstra A.* Mimblewimble. — 2016.

24. *Popov S.* IOTA whitepaper v1.4.3. — 2018. — URL: https://assets.ctfassets.net/r1dr6vzfxhev/2t4uxvsIqk0EUau6g2sw0g/45eae33637ca92f85dd9f4a3a218e1ec/iota1%7B%5C_%7D4%7B%5C_%7D3.pdf.
25. Shippable. — URL: https://app.shippable.com/accounts/5aad7d3e76ee0c1700c1d8d2/dashboard (дата обр. 20.05.2019).
26. *Sir Mark Walport*. Distributed Ledger Technology: beyond block chain. — 2018. — DOI: 10.1021/acsaem.8b00240.
27. SQLite. — URL: https://sqlite.org/index.html (дата обр. 20.05.2019).
28. The Official YAML Web Site. — URL: https://yaml.org/ (дата обр. 20.05.2019).
29. *Tim Swanson*. Great Chain of Numbers. — 2014. — URL: https://www.scribd.com/document/210537698/Great-Chain-of-Numbers-a-Guide-to-Smart-Contracts-Smart-Property-and-Trustless-Asset-Management-Tim-Swanson (дата обр. 23.04.2019).
30. Travis CI. — URL: https://travis-ci.org/ (дата обр. 20.05.2019).
31. Using etcd. — URL: https://coreos.com/etcd/ (дата обр. 20.05.2019).
32. *Van Saberhagen N.* CryptoNote v 1.0. — 2012.
33. *Van Saberhagen N.* Monero: CryptoNote v 2.0 // White Paper. — 2013. — С. 1—20. — URL: https://bytecoin.org/old/whitepaper.pdf%7B%5C%%7D0Ahttps://cryptonote.org/whitepaper.pdf.
34. *Vitalik Buterin*. On Public and Private Blockchains. — 2015. — URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/ (дата обр. 22.04.2019).
35. Wikipedia. — URL: https://www.wikipedia.org/ (дата обр. 20.05.2019).
36. *Zerocoin Electric Coin Company*. ZCash. — 2016.
37. ZEXE: Enabling decentralized private computation / S. Bowe [и др.]. — 2019. — URL: https://eprint.iacr.org/2018/962.pdf.
38. Направленный ациклический граф — Википедия. — URL: https://ru.wikipedia.org/wiki/%D0%9D%D0%B0%D0%BF%D1%80%D0%B0%D0%B2%D0%BB%D0%B5%D0%BD%D0%BD%D1%8B%D0%B9%7B%5C_%7D%D0%B0%D1%86%D0%B8%D0%BA%D0%BB%D0%B8%D1%87%D0%B5%D1%81%D0%BA%D0%B8%D0%B9%7B%5C_%7D%D0%B3%D1%80%D0%B0%D1%84 (дата обр. 22.05.2019).
39. *Ольга Скоробогатова*. О российском консорциуме, национальной электронной валюте. — 2016. — URL: https://bankir.ru/publikacii/20160419/olga-skorobogatova-o-rossiiskom-konsortsiume-natsionalnoi-elektronnoi-valyute-10007442/ (дата обр. 23.04.2019).