Санкт-Петербургский государственный университет

Программная инженерия
Кафедра системного программирования

Тюляндин Иван Владимирович

Проектирование и реализация среды исполнения смарт-контрактов для блокчейна Hyperledger Iroha

Выпускная квалификационная работа

Научный руководитель:
ст. преп. Я. А. Кириленко

Консультант:
программист ООО "Интеллиджей Лабс"
к. ф.-м. н. Д. А. Березун

Рецензент:
Генеральный директор ООО "Сорамитсу Лабс"
К. Р. Салахиев

Санкт-Петербург
2019

SAINT PETERSBURG STATE UNIVERSITY

Software Engineering

Ivan Tyulyandin

Design and implementation of smart contracts execution environment for Hyperledger Iroha blockchain

Graduation Thesis

Scientific supervisor:
Senior lecturer Iakov Kirilenko

Consultant:
IntelliJ Labs Co. Ltd developer
PhD Daniil Berezun

Reviewer:
Soramitsu Labs CEO Kamil Salakhiev

Saint-Petersburg
2019

Оглавление

Введение

Блокчейн — это распределенное, неизменяемое хранилище данных, которое обеспечивает процесс записи транзакций и отслеживания различных активов в бизнес-сетях. У каждого участника сети хранится копия истории проведенных операций, на основе которой проверяется валидность последующих транзакций в сети. Данная технология ускоряет процесс обмена, уменьшает риски и снижает стоимость для всех вовлечённых сторон. Впервые блокчейн был применен в 2008 году для реализации криптовалюты Bitcoin [17]. Структура блокчейна выглядит следующим образом: транзакции объединяются в *блоки*, которые хранятся в истории в виде односвязного списка. В каждом блоке содержится хеш от предыдущего блока. Таким способом хранения информации достигается иммутабельность — одно из ключевых свойств блокчейна.

Существуют блокчейны двух видов: *публичные* (permissionless) и *приватные* (permissioned). В приватных блокчейнах все стороны известны, и их можно идентифицировать. Каждому пользователю выдаются права доступа, выполнение которых можно обеспечить с помощью цифровых сертификатов. Наличие прав доступа позволяет хранить больше деталей бизнес-сделки. В противовес этому в публичных сетях пользователи анонимны, при этом любой участник может увидеть содержимое любой транзакции. Примеры публичных платформ: Bitcoin, Ethereum [26], приватных: Hyperledger Fabric [6].

Для корректной работы блокчейна необходимо, чтобы у всех участников сети была одинаковая история транзакций. Существуют различные *алгоритмы консенсуса*, которые позволяют синхронизировать историю. В ходе работы такого алгоритма выбирается участник сети, который предложит следующую транзакцию (или блок), а остальные участники договариваются, принимать эту транзакцию или нет. Самые известные алгоритмы консенсуса: Proof-of-Work [11] (соревнование в решении вычислительно-сложной математической задачи, используется в Bitcoin, Ethereum), Proof-of-Stake [20] (участник,

предлагающий транзакцию, выбирается по доле активов, пример — NavCoin [18]), решение задачи о византийских генералах, применяется в Hyperledger Fabric (Practical Byzantine Fault Tolerance [3]).

В некоторых блокчейнах есть возможность заключать *смарт-контракты* (smart contracts). Они позволяют точно зафиксировать условия бизнес-сделки в виде программы. Все стороны выполняют ровно то, что описано в коде. Результаты выполнения смарт-контракта либо невозможно, либо очень сложно подделать, это зависит от конкретного блокчейна и алгоритма консенсуса.

Существуют различные языки и среды исполнения смарт-контрактов. Для платформы Ethereum используются язык Solidity и байткод Ethereum Virtual Machine (далее EVM), для Bitcoin — Bitcoin-script, в Hyperledger Fabric смарт-контракты могут быть реализованы на языках Go, Node.js и Java.

Hyperledger Iroha [8] — приватный блокчейн консорциума Hyperledger (часть Linux Foundation) с открытым кодом. Hyperledger Iroha позиционируется как простая и производительная система с алгоритмом консенсуса YAC [28]. У блокчейна Hyperledger Iroha на текущий момент нет среды исполнения смарт-контрактов, что сильно уменьшает область его использования и ограничивает функциональность. При наличии смарт-контрактов, данный блокчейн можно будет использовать в различных областях, таких как: торговля, медицина, документооборот и так далее. Смарт-контракты повысят доверие участников внутри сетей на блокчейне Hyperledger Iroha и ускорят взаимодействие за счет автоматизации процедуры заключения сделок.

1. Постановка задачи

Цель данной работы — реализовать инфраструктуру поддержки среды исполения смарт-контрактов для блокчейна Hyperledger Iroha.

Были поставлены следующие задачи:

- выполнить обзор существующих языков и сред исполнения смарт-контрактов;
- разработать архитектуру и программный интерфейс для взаимодействия среды исполнения смарт-контрактов с Hyperledger Iroha;
- реализовать взаимодействие одной из существующих сред исполнения с Hyperledger Iroha;
- провести тестирование добавленной среды исполнения смарт-контрактов.

2. Обзор

В этой главе будут рассмотрены среды исполнения и языки смарт-контрактов, а так же некоторые особенности Hyperledger Iroha.

2.1. Языки и среды исполнения смарт-контрактов

В 1997 году Ник Сабо (Nick Szabo) предложил концепцию смарт-контрактов [24]. Смарт-контракт — это программа, которая описывает взаимодействие участников блокчейн-сети. При сравнении с традиционным бумажным контрактом, смарт-контракт имеет однозначную семантику и выполняется автоматически при достижении определенных условий. Результат выполнения смарт-контракта легко подтверждается, так как он будет записан в историю транзакций блокчейна. На сегодняшний день существует множество различных языков смарт-контрактов и блокчейнов, которые могут исполнять программы на этих языках.

Смарт-контракт всегда должен завершаться для продолжения работы блокчейн-сети. Если язык смарт-контрактов Тьюринг-полный, то среда исполнения должна предоставлять механизм, который будет ограничивать тем или иным способом количество операций смарт-контракта. В случае Ethereum каждая инструкция стоит определенное количество *газа*, цена которого выражена во внутренней криптовалюте Ethereum. В Hyperledger Fabric имеется ограничение на время выполнения кода смарт-контракта. Для языка Rholang [19], основанном на Rho-calculus [15], выставляется лимит по количеству применений правил редукции.

Далее будут рассмотрены несколько сред исполнения смарт-контрактов и языки, которые данные среды поддерживают.

2.1.1. Языки смарт-контрактов

В этом параграфе рассмотрены языки смарт-контрактов с учетом их парадигм и свойств, таких как Тьюринг-полнота, механизм огра-

ничения выполнения смарт-контракта платформой, на которой смарт-контракт исполняется, и системы типов.

На данный момент существуют языки смарт-контрактов с различным уровнем абстракции. *Низкоуровневые языки* (low-level) предназначены для непосредственного выполнения средой исполнения. Многие концепции, такие как семантика, вычислительная модель, система ограничения выполнения и типизация часто описываются на этом уровне. Примеры таких языков — EVM [26], Bitcoin Script [2] и Michelson [16]. *Высокоуровневые языки* (high-level), такие как Solidity [22], Flint [4] и Liquidity [13], упрощают процесс разработки смарт-контрактов за счет повышенной читаемости, наличия более абстрактных синтаксических конструкций и системы типов. *Промежуточные языки* (intermediate-level) смарт-контрактов являются своего рода компромиссом между высокоуровневыми и низкоуровневыми языками по степени абстракции. Как правило, они спроектированы для упрощения формальной верификации или статического анализа исходного кода, с учетом вычислительной модели, системы типов, семантики и других формализмов. Scilla [21] является промежуточным языком смарт-контрактов.

В ходе обзора языков смарт-контрактов на конференцию SYRCoSE 2019 в соавторстве была написана обзорная статья *A Survey of Smart Contract Safety and Programming Languages*, которая принята к публикации в сборнике трудов ИСП РАН. В приложении А приведена сводная таблица по языкам смарт-контрактов и их свойствам из данной статьи. В ней приведены следующие характеристики языков смарт-контрактов: название, уровень абстракции, текущее состояние разработки, проект, для которого язык предназначен, парадигма, способ ограничения выполнения смарт-контракта целевой платформой и Тьюринг-полнота соответственно.

Было выявлено, что Ethereum является наиболее популярной платформой для работы со смарт-контрактами. Экосистема данного блокчейна развита: существует множество языков с различными подходами, сред разработки и статических анализаторов. Аналогичной экосистемы нет ни у одного из всех рассмотренных блокчейнов.

2.1.2. Ethereum Virtual Machine

Ethereum Virtual Machine (сокращенно EVM) — Тьюринг-полная виртуальная стековая машина блокчейна Ethereum. Смарт-контракты для этой платформы написаны на байткоде [26]. Под эту среду исполнения смарт-контрактов существует множество языков [1, 4, 10, 14, 22, 23, 25, 27, 29], которые компилируются в байткод EVM.

Есть два участка памяти, куда EVM может записывать значения во время выполнения кода смарт-контракта — *memory* и *storage*. Memory является временным хранилищем данных, необходимым для записи промежуточных значений. Размер машинного слова EVM 256 битов. Можно провести аналогию, что memory для EVM — это как оперативная память для компьютера. Ячейки memory адресуются от 0 до $2^{256}-1$ и содержат байт информации. Storage представляет из себя хранилище пар вида ключ-значение, которые описывают текущее состояние переменных смарт-контракта. В отличие от memory, данные storage записываются в блокчейн. Размер storage равен 2^{256} ячеек, каждая из них хранит машинное слово.

Аккаунты в блокчейн сети Ethereum бывают двух видов: *пользовательские* и *контракты*. Аккаунты-контракты содержат код, который может быть вызван пользовательским аккаунтом или кодом другого контракта.

Существует множество реализаций клиентов для работы с распределённой блокчейн сетью Ethereum, например Geth[1] и Aleth[2]. Также есть проекты, которые используют только виртуальную машину Ethereum и её байт-код, таким проектом является Hyperledger Burrow [5].

2.1.3. Выбор среды исполнения

Для интеграции в Hyperledger Iroha была выбрана среда исполнения смарт-контрактов из проекта Hyperledger Burrow. Этот про-

[1]https://geth.ethereum.org/

[2]http://www.ethdocs.org/en/latest/ethereum-clients/cpp-ethereum/

ект реализует приватный блокчейн с возможностью выполнения смарт-контрактов и написан на языке Go. Ключевыми особенностями реализации являются алгоритм консенсуса Tendermint [12], наличие программного интерфейса для удаленного вызова процедур, возможность выставлять права доступа к данным и на выполнение операций внутри сети, а также виртуальная машина для смарт-контрактов.

Эта среда исполнения обладает следующими преимуществами: 1) выполнение смарт-контрактов, написанных на байт-коде EVM; 2) наличие программного интерфейса для взаимодействия; 3) проект поддерживается Hyperledger; 4) есть примеры интеграции с проектами Hyperledger Fabric [7] и Hyperledger Sawtooth [9]. Так как Ethereum де-факто является самой популярной платформой для работы со смарт-контрактами, программистам будет проще адаптироваться к разработке на Hyperledger Iroha. Важную роль играет принадлежность к Hyperledger — при возникновении проблем и вопросов на этапе интеграции виртуальной машины можно обратиться к сообществу разработчиков указанных ранее проектов.

У виртуальной машины внутри есть доступ к участку памяти memory, а для реализации операций над участком памяти storage необходимо обращаться к истории блокчейна. Программный интерфейс для взаимодействия описывает все методы, которые нужны виртуальной машине, чтобы получать, добавлять и обновлять данные блокчейна.

2.2. Hyperledger Iroha

Hyperledger Iroha [8] — это приватный блокчейн, обладающий функциональностью для управления активами и сущностями, вводимые пользователями. В нем используется алгоритм консенсуса YAC [28], который основан на решении задачи о византийских генералах. Данный фреймворк для распределенных хранилищ ориентирован на использование на мобильных устройствах. Проект написан на языке C++ с

использованием библиотек Boost[3], Protobuf[4], GTest[5] и множества других. В качестве хранилища истории блокчейна сети используется локальная для каждого участника база данных PostgreSQL[6].

Как и во многих блокчейн сетях, пользователи Hyperledger Iroha выполняют различные действия с помощью транзакций, которые формируются из набора специальных команд. Используя их, участники могут пересылать активы и управлять правами доступа, например, создать новую сущность или передать право владением активом другому участнику.

Прежде чем принять транзакцию, всем участникам сети нужно подтвердить её корректность. Есть два этапа валидации — *stateless* и *stateful*. Команды должны быть сформированы согласно определённым правилам. Во время stateless валидации контролируется соответствие этим правилам. Далее проводится stateful валидация. На этом этапе проверяется выполнимость отправленной транзакции, например, права доступа или наличие активов.

[3]https://www.boost.org/
[4]https://developers.google.com/protocol-buffers/
[5]https://github.com/google/googletest
[6]https://www.postgresql.org/

3. Взаимодействие среды исполнения смарт-контрактов с Hyperledger Iroha

В этой главе будет рассмотрена схема взаимодействия среды исполнения смарт-контрактов и программный интерфейс, который эту схему реализует.

3.1. Схема взаимодействия

Для создания инфраструктуры для среды исполнения смарт-контрактов необходимо разработать схему взаимодействия этой среды и блокчейна Hyperledger Iroha. Среда исполнения должна уметь получать и записывать данные в блокчейн.

На рисунке 1 показано взаимодействие среды исполнения с Hyperledger Iroha и локальной базой данных PostgreSQL, в которой хранится история транзакций, права доступа и информация о пользователях. Среда исполнения принимает на вход код смарт-контракта и начинает его выполнение. Для того, чтобы получать и записывать актуальное состояние переменных смарт-контракта, среда исполнения об-

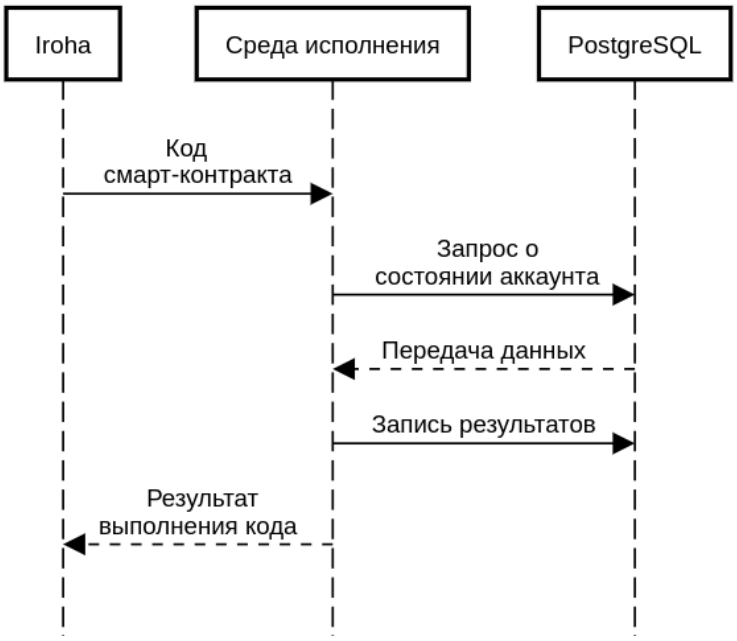

Рис. 1: Диаграмма последовательности взаимодействия

ращается к базе данных. Запросы выполняются по необходимости.

3.2. Реализация интерфейса взаимодействия

Пользователи формируют транзакции с помощью специального набора команд. Для работы с командами в коде используется паттерн «фабричный метод». Чтобы участники могли сохранять и выполнять код смарт-контракта, была добавлена новая команда *AddSmartContract*. Она включает в себя данные, необходимые для обращения к среде исполнения. Содержание команды зависит от конкретной среды исполнения. Для передачи транзакций и записи в блокчейн используются библиотеки Protobuf и RapidJSON[7], поэтому нужно уметь переводить данные, содержащиеся в AddSmartContract, как и в любой другой команде, в форматы Protocol Buffers и JSON и обратно во внутреннее представление.

Перед тем, как транзакция будет записана в истории блокчейна, все участники сети должны провести stateless и stateful валидацию всех команд внутри этой транзакции. Для AddSmartContract stateless валидация может содержать различные проверки кода, например на синтаксическую корректность. Во время stateful валидации участник сети обязан выполнить код и получить новое состояние блокчейна, чтобы в дальнейшем во время работы алгоритма консенсуса все участники договорились, принимать это новое состояние или нет.

Таким образом, был создан программный интерфейс в виде новой команды AddSmartContract. Для обращения к среде исполнения смарт-контрактов нужно указать формат входных данных в реализации AddSmartContract, а также сделать запрос во время stateful валидации к среде исполнения и получить от неё результат выполнения кода смарт-контракта.

[7] http://rapidjson.org/

4. Внедрение среды исполнения смарт-контрактов

В этой главе описаны аргументы для выбора среды исполнения и реализация взаимодействия этой среды с блокчейном Hyperledger Iroha.

4.1. Взаимодействие среды исполнения смарт-контрактов с Hyperledger Iroha

В данном параграфе описано взаимодействие виртуальной машины Hyperledger Burrow, написанной на языке Go, с блокчейном Hyperledger Iroha в реализованной системе на примере жизненного цикла смарт-контракта.

Сначала пользователь добавляет в транзакцию команду AddSmartContract (см. параграф 3.2), передавая ей следующие данные: адрес вызывающего, адрес вызываемого, количество газа, код смарт-контракта и, если нужно вызвать уже существующую функцию — её сигнатуру и аргументы. Код передается в виде шестнадцатеричной последовательности, в которой закодированы операции виртуальной машины. Вызов функции кодируется следующим образом: на строковое представление сигнатуры функции применяется хэш keccak256, берутся первые четыре байта результата, а входные аргументы приписываются в конец. После того, как пользователь завершил создание транзакции, она отправляется остальным участникам сети для проверки и последующего принятия или отклонения.

Далее транзакция должна пройти валидацию у каждого участника сети. При stateful валидации команды AddSmartContract происходит обращение к виртуальной машине Hyperledger Burrow с заданными параметрами. Для этого код на C++ должен передавать данные в среду исполнения языка Go и получать из неё результат после соответствующего запроса к виртуальной машине. Связывание сред исполнения реализовано следующим образом. На языке Go написана обёртка,

внутри которой находится код, делающий запрос к виртуальной машине, и вспомогательные функции, реализующие интерфейс, необходимый для работы виртуальной машины. Обёртка имеет доступ к базе данных PostgreSQL, в которой локально хранится история транзакций блокчейн сети. Обёртка транслируется компилятором Go с использованием опции *buildmode*, которая создает разделяемую библиотеку (shared object) и заголовочный файл на языке C для работы с ней.

Таким образом, все участники независимо друг от друга запускают код и получают новое состояние блокчейна. Далее, в соответствии с алгоритмом консенсуса, достигается соглашение о принятии или отклонении транзакции, которое сообщается пользователю.

5. Тестирование

В ходе работы в проект были добавлены два компонента: интерфейс для взаимодействия со средой исполнения смарт-контрактов (команда AddSmartContract) и виртуальная машина из проекта Hyperledger Burrow. В главе описано тестирование внедрённых команды AddSmartContract и реализации программного интерфейса, необходимого для корректной работы виртуальной машины. В проекте Hyperledger Iroha существует инструмент тестирования ITF (Integration Test Framework), основанный на библиотеке GTest. Он позволяет настроить блокчейн и его историю: сформировать аккаунты пользователей, выставить параметры для алгоритма консенсуса, создать заглушку, содержащую данные для формирования транзакций, провести stateless и stateful валидацию заранее заготовленной транзакции.

Для проверки команды AddSmartContract были добавлены модульные тесты, проверяющие операции над данными внутри команды, а именно: сериализация данных в форматы Protocol Buffers и JSON и десериализация из них — а также интеграционные тесты на stateless и stateful валидацию.

Для тестирования реализации программного интерфейса, которая необходима виртуальной машине, были разработаны модульные тесты, содержащие вызов различных смарт-контрактов, в том числе и некорректных с точки зрения соответствия байт-коду EVM. Внутри них выполняются базовые операции, такие как создание аккаунта-контракта, присваивание и чтение переменных, вызов функции с параметрами. Критерием успеха являлась корректность состояния блокчейна после выполнения одного или нескольких смарт-контрактов в рамках одной транзакции.

6. Результаты

Была реализована инфраструктура поддержки среды исполнения смарт-контрактов для блокчейна Hyperledger Iroha.

В ходе работы были выполнены следующие задачи:

- выполнен обзор существующих сред исполнения и языков смарт-контрактов, обзорная статья принята для публикации в сборнике трудов ИСП РАН;
- разработаны архитектура и программный интерфейс для взаимодействия среды исполнения смарт-контрактов с Hyperledger Iroha
- реализовано взаимодействие Hyperledger Iroha и среды исполнения смарт-контрактов проекта Hyperledger Burrow;
- проведено модульное и интеграционное тестирование.

A. Обзорная таблица языков смарт-контрактов

Language	Level	Current state	Project	Paradigm / influence	Metering	Turing completeness
Bamboo	high-level	alpha (experimental)	Ethereum	functional	gas system	yes
Bitcoin Script	low-level	under development	Bitcoin	stack-based, reverse-polish	script size	no
Chain-code	high-level	stable	Hyperledger Fabric	general purpose languages	timeout	yes
EOSIO	high-level	stable	EOS.IO	object-oriented, statically typed	bound system	yes
EVM bytecode	low-level	stable	Ethereum	stack-based	gas system	yes
Flint	high-level	alpha	Ethereum	type safe, contract-oriented	gas system	yes
IELE	low-level	prototype	Ethereum	register-based	gas system	yes
Ivy	high-level	prototype (experimental)	Bitcoin	imperative	gas system	no
Liquidity	high-level	under development	Tezos	fully-typed, functional	gas system	yes
LLL	intermediate-level	under development	Ethereum	stack-based	gas system	yes
Logikon	high-level	experimental	Ethereum	logical-functional	gas system	yes
Michelson	low-level	under development	Tezos	stack-based, strongly typed	gas system	yes
Plutus (PlutusCore)	high-level (low-level)	under development	Cardano	functional	gas system	yes
Rholang	high-level	under development	RChain	functional	rule reduction system	yes
Scilla	intermediate-level	under development	Zilliqa	functional	gas system	no
Simplicity	low-level	under	Bitcoin	functional, typed,	Bit Machine	no

продолжение

Language	Level	Current state	Project	Paradigm / influence	Metering	Turing completeness
		development		combinator-based		
Solidity	high-level	stable	Ethereum	statically typed, object-oriented	gas system	yes
SolidityX	high-level	beta	Ethereum	secure-oriented	gas system	yes
Vyper	high-level	beta	Ethereum	imperative	gas system	no
Yul	intermediate-level	under development	Ethereum	object-oriented	gas system	yes

Список литературы

[1] Bamboo. — Access mode: https://github.com/pirapira/bamboo (online; accessed: 2019-05-24).

[2] Bitcoin script. — Access mode: https://en.bitcoin.it/wiki/Script (online; accessed: 2019-05-16).

[3] Castro Miguel, Liskov Barbara. Practical Byzantine Fault Tolerance and Proactive Recovery // ACM Trans. Comput. Syst. — Vol. 20, no. 4. — P. 398–461. — Access mode: http://doi.acm.org/10.1145/571637.571640.

[4] Flint. — Access mode: https://github.com/flintlang/flint (online; accessed: 2019-05-24).

[5] Hyperledger Burrow. — Access mode: https://www.hyperledger.org/projects/hyperledger-burrow (online; accessed: 2019-05-26).

[6] Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains / Elli Androulaki, Artem Barger, Vita Bortnikov et al. // Proceedings of the Thirteenth EuroSys Conference. — EuroSys '18. — New York, NY, USA : ACM, 2018. — P. 30:1–30:15. — Access mode: http://doi.acm.org/10.1145/3190508.3190538.

[7] Hyperledger Fabric EVM Chaincode. — Access mode: https://github.com/hyperledger/fabric-chaincode-evm (online; accessed: 2019-05-26).

[8] Hyperledger Iroha. — Access mode: https://soramitsu.co.jp/iroha (online; accessed: 2019-05-12).

[9] Hyperledger Sawtooth - Seth. — Access mode: https://github.com/hyperledger/sawtooth-seth/ (online; accessed: 2019-05-26).

[10] IELE: An Intermediate-Level Blockchain Language Designed and Implemented Using Formal Semantics : Rep. :

http://hdl.handle.net/2142/100320 / University of Illinois ; Executor: Theodoros Kasampalis, Dwight Guth, Brandon Moore et al. : 2018. — July.

[11] Jakobsson Markus, Juels Ari. Proofs of Work and Bread Pudding Protocols // Proceedings of the IFIP TC6/TC11 Joint Working Conference on Secure Information Networks: Communications and Multimedia Security. — CMS '99. — Deventer, The Netherlands, The Netherlands : Kluwer, B.V., 1999. — P. 258–272. — Access mode: http://dl.acm.org/citation.cfm?id=647800.757199.

[12] Kwon Jae Kyun. Tendermint : Consensus without Mining. — 2014.

[13] Liquidity. — Access mode: https://github.com/OCamlPro/liquidity (online; accessed: 2019-05-16).

[14] Logikon. — Access mode: https://github.com/logikon-lang/logikon (online; accessed: 2019-05-24).

[15] Meredith L. Gregory, Radestock Matthias. A Reflective Higher-order Calculus // Electr. Notes Theor. Comput. Sci. — 2005. — Vol. 141. — P. 49–67.

[16] Michelson language. — Access mode: https://www.michelson-lang.com/ (online; accessed: 2019-01-31).

[17] Nakamoto Satoshi. Bitcoin: A peer-to-peer electronic cash system // http://www.bitcoin.org/bitcoin.pdf. — 2008.

[18] NavCoin. — Access mode: https://navcoin.org/en/ (online; accessed: 2019.04.14).

[19] RChain and Rholang. — Access mode: https://www.rchain.coop/platform (online; accessed: 2019-05-24).

[20] Saleh Fahad. Blockchain Without Waste: Proof-of-Stake // SSRN Electronic Journal. — 2018. — 01.

[21] Sergey Ilya, Kumar Amrit, Hobor Aquinas. Scilla: a Smart Contract Intermediate-Level LAnguage // CoRR. — 2018. — Vol. abs/1801.00687.

[22] Solidity. — Access mode: `https://github.com/ethereum/solidity` (online; accessed: 2019-05-24).

[23] SolidityX. — Access mode: `https://solidityx.org/` (online; accessed: 2019-0-24).

[24] Szabo Nick. Formalizing and Securing Relationships on Public Networks // First Monday. — 1997. — Vol. 2, no. 9.

[25] Vyper. — Access mode: `https://github.com/ethereum/vyper` (online; accessed: 2019-05-24).

[26] Wood Gavin. Ethereum: A secure decentralised generalised transaction ledger. — Access mode: `https://ethereum.github.io/yellowpaper/paper.pdf` (online; accessed: 2019-04-14).

[27] Wood Gavin. LLL. — Access mode: `https://lll-docs.readthedocs.io/en/latest/index.html` (online; accessed: 2019-05-24).

[28] YAC: BFT Consensus Algorithm for Blockchain / Fedor Muratov, Andrei Lebedev, Nikolai Iushkevich et al. // CoRR. — 2018. — Vol. abs/1809.00554. — 1809.00554.

[29] Yul. — Access mode: `https://solidity.readthedocs.io/en/latest/yul.html` (online; accessed: 2019-05-24).