Санкт-Петербургский государственный университет

Кафедра системного программирования
Программная инженерия

Мишин Никита Матвеевич

Интеграция инструментов разработки для языка Vyper в Intelij Platform

Курсовая работа

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

Консультант:
к. ф.-м. н. Березун Д. А.

Санкт-Петербург
2019

Оглавление

Введение

В 2008 появилась первая криптовалюта — *Bitcoin* [16]. В её основе лежала новая технология блокчейн, которая быстро завоевала популярность.

Блокчейн — это одна из технологий распределенного хранения данных, состоит из цепочки последовательных блоков, которые связаны между собой с помощью хеширования — каждый блок хранит хэш-код предыдущего блока. Любое изменение данных в блоке ведет к изменению хэш-кодов — практически невозможно подделать хранящиеся данные. Можно привести аналогию с бухгалтерской книгой, хранящейся у всех участников сети, в которой нельзя ничего изменять или удалять, можно лишь только добавлять. Блокчейн позволяет безопасно распространять или обрабатывать данные между несколькими лицами через потенциально небезопасную сеть, где существуют недобросвестные узлы. Также засчет того, что у участников сети есть копия цепочки, то в любой момент каждый может проследить историю всех транзакций.

После создания Bitcoin эту технологию начали активно применять, и в блокчейн-среде высказывались идеи о том, что область применения технологии значительно шире, чем использование в криптовалютах [15]. Так появилась идея смарт-контрактов, которая была впервые реализована в *Ethereum* [8].

Смарт-контракт (smart contract) — это программируемый объект, который работает поверх блокчейна. Он обеспечивает передачу информации (цифровых данных) и соблюдение условий контракта обеими сторонами (является аналогом обычных контрактов).

Появление смарт-контрактов расширило сферу применения блокчейн-технологии [4], но вместе с тем появился ряд проблем [4]. Сложный синтаксис и неинтуитивная семантика языков программирования, на которых пишутся смарт-контракты, также не облегчают задачу. В частности, эти проблемы являются актуальными для *Ethereum* — одной из самых популярных [1] платформ для написания смарт-контрактов, в которой присутствует их полноценная поддержка.

В рамках данной платформы для нивелирования указанных проблем был разработан экспериментальный не Тьюринг-полный язык *Vyper* [19]. Он находится в бета-версии и активно развивается[1].

В силу того, что язык совсем новый, находится в бета-версии и часто видоизменяется, его поддержка присутствует лишь в нескольких местах, причем реализована лишь самая базовая функциональность (возможность компиляции, подсветка синтаксиса).

Учитывая идеи, заложенные в язык, поддержку со стороны *Ethereum* сообщества и разработчиков платформы, *Vyper* будет набирать популярность и продвигаться сообществом как основной язык для написания смарт-контрактов для этой платформы.

Таким образом, актуальной является задача реализации полноценной поддержку языка в виде плагина для одной из популярных интегрированных сред разработки (*IDE*), который бы ускорял и облегчал процесс написания, разработки и развертывания смарт-контрактов в рамках *Ethereum*.

[1] текущая версия языка v0.1.0-beta.9

1. Постановка задачи

Целью данной работы является реализация поддержки языка *Vyper* для *Intelij Platfrom*. В рамках курсовой будет осуществлена поддержка инструментов разработки.

Для достижения цели были поставлены следующие задачи:

- провести исследование предметной области:
  - исследовать и проанализировать блокчейн-платформу, для которой создан *Vyper*;
  - провести обзор *Vyper* и инструментов его разработки;
  - исследовать существующие плагины для языка;
- изучить и провести анализ IDE, в рамках которой будет реализована поддержка языка;
- разработать и реализовывать плагин:
  - разработать архитектуру решения;
  - интегрировать компилятор *Vyper*;
  - интегрировать статический анализатор для анализа контрактов;
  - поддержать возможность тестирования смарт-контрактов.

2. Обзор

В этой секции описаны особенности смарт-контрактов, приведено описание известных блокчейн-платформ, в которых присутствуют смарт-контракты. Рассмотрена платформа *Ethereum* и язык *Vyper*.

2.1. Смарт-контракты

Смарт-контракт (smart contract) — это программируемый объект, который работает поверх блокчейна. В нем содержится набор правил и условий, на которых участники согласны взаимодействовать друг с другом. Если условия соблюдены, смарт-контракт автоматически выполняется после соответствующего запроса или транзакции. Смарт-контракты призваны минимизировать доверие третьим сторонам. Они также хранятся в блокчейне, что гарантирует их неизменяемость.

В сравнении с обычными,нецифровыми контрактами, они обладают рядом плюсов:

- независимость и низкая стоимость — нет необходимости поиска специалиста, который проведет сделку;
- безопасность и надежность — условия нельзя изменить, так как все хранится в распределенном реестре в шифрованном виде и продублировано на других узлах сети;
- автоматизация процесса сделок и точность – обычные контракты содержат много форм, в которых человек может допустить ошибку.

Наряду с вышеописанными плюсами присутствуют и минусы:

- неопределенность правового статуса;
- сложность написания;
- отсутствие понимания работы смарт-контрактов со стороны конечного пользователя.

2.2. Блокчейн-платформы

На сегодняшний день существует несколько платформ, поддерживающих смарт-контракты [7]: *Ethereum* [20], *Tezos* [12], *Hyperledger Fabric* [10], *Corda* [13] и др. Самой большой капитализацией из рассмотренных платформ обладает *Ethereum* [1] (технически, можно сказать, что в *Bitcoin* есть примитивные смарт-контракты[2]). По количеству активных пользователей (адреса без нулевого счета) *Ethereum* обогнал [6] *Bitcoin*, что свидетельствует о том, что эта платформа является одной из самых популярных и востребованных. *Ethereum* позволяет создавать поверх блокчейна любые произвольные децентрализованные приложения (Dapps[3]), также присутствует возможность создания децентрализованных автономных организаций (DAO[4]). В *Corda* присутствует схожие по своей структуре и функциональности распределенные приложения, называемые *CorDapps*, но эта платформа нацелена на финансовый сектор и не является публичной[5], что также справедливо и по отношению к *Hyperledger Fabric*. Многообещающей платформой является *Tezos*, в которой используется формальная верификация смарт-контрактов, но она лишь недавно вышла из бета-тестирования и еще не прошла проверку временем.

Суммируя вышесказанное, платформа *Ethereum* обладает самой большой капитализацией, в ней присутствует больше всего активных пользователей и эта платформа является полностью публичной. Стоит отметить, что это первая платформа, в которой появились смарт-контракты. В частности, поэтому в интернете у *Ethereum* большое сообщество и она хорошо документирована.

Таким образом, поддержка языка для данной платформы в рамках интегрированной среды разработки является наиболее актуальной.

[2]Примером может служить мультиподпись

[3]Dapps — программные приложения, которые работают поверх децентрализованной платформы

[4]Отличительной особенностью DAO является то, что они управляются программным кодом (смарт-контракты), не контролируются ни одной властью и не могут быть закрыты

[5]транзакции происходят в свободном порядке и никем не контролируются, нет ограничений на доступ к данным и сети

2.3. Ethereum

Ethereum[20] — это платформа для создания децентрализованных приложений на базе блокчейна, которые используют смарт-контракты. Смарт-контракты, в основном, пишутся на языках высокого уровня (сейчас основным языком служит *Solidity*[6]), комплирующихся в коды операций для виртуальной среды (Ethereum Virtual Machine), в рамках которой они изолированно исполняются. Этот язык Тьюринг-полный, в отличие от *bitcoin-script* [7]. В рамках платформы пользователи могут создавать контракты и вызывать их функции посредством осуществления транзакций. Транзакции валидируются другими узлами сети. И пользователи, и контракты могут хранить внутреннюю валюту (*ether*) и отправлять или принимать ее от других контрактов или пользователей. Контракты дополнительно могут хранить какую-то информацию, необходимую для логики их исполнения. Также существует несколько высокоуровневых языков, которые транслируются в коды виртуальной машины *Ethereum*. *Mutan* (golang-подобный язык, который не поддерживается с 2015 года), *LLL* (lisp-подобный язык, еще поддерживается, но сложен в использовании), *Serpent* (python-подобный язык) и упомянутый выше *Solidity* (javascript-подобный, контрактно-ориентированный язык).

Несмотря на то, что *Solidity* является узкоспециализированным языком (*DSL* для написания смарт-контрактов), он сложен по своей структуре и обладает неинтуитивной семантикой. Также *Solidity* Тьюринг-полный (для многих сценариев смарт-контрактов не нужна Тьюринг-полнота). В частности, и из-за этого возникает большое количество ошибок и уязвимостей [5] при написании смарт-контрактов, которые могут стоить огромных денег. [2, 3] служат лишь их яркими примерами. Стоит, однако, заметить, что существуют фреймворки [14, 11], которые пытаются формально верифицировать смарт-контракты, что позволяет снизить количество уязвимостей в коде.

В частности, этим обуславливается появление экспериментального

[6]https://solidity.readthedocs.io/en/v0.5.8/

[7]Скриптовый язык для описания условий транзакции

Listing 1: Пример смарт-контракта на языке Vyper

```
registry: map(bytes[100], address)

@public
def register(name: bytes[100], owner: address):
    assert self.registry[name] == ZERO_ADDRESS
    self.registry[name] = owner

@public
@constant
def lookup(name: bytes[100]) -> address:
    return self.registry[name]
```

языка *Vyper*[8] [17], создатели которого попытались избавиться от недостатков *Solidity*.

2.4. Экспериментальный язык Vyper

Vyper — это python-подобный, контрактно-ориентированный язык программирования. Начат разрабатываться в 2017 году создателем *Ethereum* платформы Виталием Бутериным как решение проблем с многочисленными уязвимостями в смарт-контрактах.

При разработке и дизайне языка было принято во внимание несколько идей. Во-первых, смарт-контракты, написанные на языке, должны быть легкочитаемыми и простыми для понимания не только специалистами в этой сфере, но и для людей, имеющих малый опыт в данной области. Также синтаксис и семантика языковых конструкций должны быть просты и однозначны. Во-вторых, язык должен быть реализован таким образом, что написание безопасного кода было естественно и не предполагало никаких усилий со стороны конечного пользователя, т.е написание контрактов, которые содержат уязвимости, было бы трудно-реализуемо в рамках языка. В-третьих, для реализации функциональности смарт-контрактов не нужна Тьюринг-полнота[9]

[8] Viper — Изначальное название языка

[9] До сих пор ведутся споры, действительно ли нужны Тьюринг-полные смарт-контракты [9]

В результате идей заложенных в язык, в *Vyper* присутствуют сильная типизация, поддержка чистых функций[10], верхняя оценка границы потребляемого газа[11] при выполнении функций и др. Отсутствуют же, например, рекурсия, бесконечные циклы, наследование классов, перегрузка методов и операторов и др.

Python-подобность языка обуславливается не только легкочитаемостью, простотой синтаксических конструкций и низким порогом вхождения, но и тем, что пользователи, пишущие на python, могут легко начать писать смарт-контракты — имеет стратегическое значение в привлечении новых пользователей к платформе.

Таким образом, в силу принципов и идей, заложенных в *Vyper*, а также его набирающую популярность вместе с поддержкой со стороны *Ethereum* сообщества, становится актуальной задача комплексной поддержки языка в рамках IDE.

[10]Ограниченная, есть возможность пометить функцию декоратором, что запретит изменять внутреннее состояние контракта в результате исполнения метода

[11]Каждая команда, которая исполняется в рамках виртуальной машины *Ethereum*, имеет стоимость, выраженную в газе. Она не только формирует рыночные отношения, но и позволяет ввести ограничение на исполнение программного кода, что позволяет частично обойти проблему останова

Тип	Функциональность
Онлайн компиляторы	Поддержка компиляции программ, в некоторых решениях есть редактор с подсветкой синтаксиса
Плагины к текстовым редакторам	Поддержка компиляции, подсветка синтаксиса, в одном из решений реализована поддержка статического анализатора байт-кода
Онлайн IDE Remix	Поддержка компиляции, базовая подсветка, тестирование и развертывание контрактов через функционал для Solitidy

Таблица 1: Пример классификации существующих решений

3. Существующие решения для Vyper

На сегодняшний день поддержка *Vyper* осуществлена лишь некоторых редакторах и онлайн инструментах. Это обусловлено тем, что язык появился недавно, находится в бета-версии и, соответственно, часто подвергается изменениям и, например, не имеет собственной официальной формальной грамматики[12]. Существующие решения можно разделить на несколько видов. Классификация представлена в таблице 1.

3.1. Онлайн компиляторы

- Vyper in browser[13]

  Присутствует возможность компиляции, используется устаревшая версия компилятора (0.1.0b5), минималистичный UI.

- Ethrescan Vyper compiler[14]

  Присутствует возможность компиляции контрактов с возможностью выбора версии компилятора.

- Vyper online compiler[15]

  Предоставляет возможность компиляции программ. Есть простое API с возможность GET/POST запроса с исходным кодом для

[12]https://github.com/ethereum/vyper/issues/1363
[13]https://jacqueswww.github.io/vyper-in-browser/
[14]https://etherscan.io/vyper
[15]https://vyper.online/

получения результатов компиляции. Присутствует базовая подсветка синтаксиса. По сути, компилятор в онлайне с подсветкой синтаксиса с использованием грамматики питона.

3.2. Плагины к текстовым редакторам

- Vim[16], Emacs[17] и Atom[18]

  Предоставляют подсветку синтаксиса. Используют ключевые слова, регулярные выражения и грамматику питона.

- Visual Studio Code[19]

  Присутствует подсветка синтаксиса с использованием грамматики питона. Интегрирован компилятор *Vyper* — пользователю необходимо установить компилятор и прописать путь к нему в настройках плагина.

- Visual Studio Code[20]

  Интегрирован компилятор, ошибки при результате компиляции подсвечиваются в коде. Также присутствует подсветка синтаксиса с использованием грамматики питона. Интегрирован статический анализатор байт кода (не *Vyper* кода) посредством онлайн взаимодействия через API MythX[21]

3.3. Remix IDE

Remix[22] —это онлайн *IDE*, специально разработанная для написания, тестирования, анализа и развертывания в сети смарт-контрактов, написанных на *Solidity*.

[16]https://github.com/jacqueswww/vim-vyper
[17]https://github.com/ralexstokes/vyper-mode
[18]https://github.com/wschwab/language-vyper
[19]https://github.com/p-/vscode-vyper
[20]https://github.com/tintinweb/vscode-vyper
[21]https://mythx.io/, статический анализатов Ethereum байт кода
[22]https://remix.ethereum.org/

В *IDE* присутствует поддержка *Vyper* — интегрирован компилятор, присутствует подсветка синтаксиса в окне редактора. В рамках *IDE* возможно быстро проверить работоспособность контрактов, а также разворачивать их в сети (функциональность доступна в силу того, что *Remix* оперирует байт кодом виртуальной машины).

Из всего вышесказанного следует, что существующую поддержку можно значительно улучшить, реализовав комплексную языковую поддержку с интеграцией инструментов разработки.

4. Реализация поддержки языка Vyper

В этой секции описывается архитектура решения и техническая реализация плагина.

4.1. Intelij Platfrom

С самого начала была выбрана платформа, в которой будет реализована поддержка языка. *Intelij Platfrom*[23] была выбрана по ряду причин:

- Она позволяет писать не под конкретную платформу (*Clion*, *Pycharm*, *Intelij Idea* и др.), а под целое семейство платформ, которые базируются на *Intelij Platfrom*;
- предоставляет не только SDK, но и широкий набор инструментов разработки в виде плагинов, позволяющих облегчить и ускорить разработки;
- невзирая на возможную неполноту документации, сам код исходной платформы и плагинов, которые написаны под нее, являются открытыми и могут служить как документацией, так и примерами реализаций;
- плагин для *Solidity*, написанный для платформы, может служить референсной реализацией;
- процесс разработки параллелится и позволяет вести модульную разработку — отдельная часть функциональности регистрируется в xml файле, который служит точкой входа в плагин;
- реализация плагина под целое семейство платформ охватывает широкий круг пользователей, что актуально в контексте привлечения аудитории к платформе *Ethereum*.

[23]https://www.jetbrains.com/opensource/idea/

4.2. Стек технологий и инструменты разработки Vyper

Основным языком разработки был выбран *kotlin*, так как он позволяет писать более компактный код по сравнению с *java*, более безопасен, в нем больше возможности написания участков кода в функциональном стиле и др. Также стоит отметить поддержку механизма сопроцедур *coroutines*, которые облегчают написание асинхронных программ и являются более производительными, чем обычные потоки.

4.2.1. Компилятор

Во всех рассмотренных плагинах, пользователю необходимо совершать ряд дополнительных усилий, чтобы иметь возможность компилировать контракты: вручную клонировать репозиторий, создавать виртуальное питоновское окружение (разработчики компилятора настоятельно советуют использовать его, так как компилятор написан на *python*), прописывать в настройках плагина путь к компилятору.

Предложенное решение заключается в использовании *docker* внутри плагина. Это избавляет пользователя от описанных сложностей. Для реализации данного решения необходимо создать образ компилятора и распространить его через *dockerhub* или же воспользоваться официальным образом.

4.2.2. Тестирование смарт-контрактов

Для тестирования смарт-контрактов был выбран фреймворк *vyper-debug*[24], специально разработанный для языка. В нем присутствует утилита *vyper-run*, которая предоставляет возможность тестирования отдельно взятых методов контракта[25]. Утилита использует компилятор, поэтому было решено создать *docker*-образ, внутри которого находился бы сам компилятор и фреймворк (в дальнейшем планируется ис-

[24]https://github.com/status-im/vyper-debug

[25]Позволяет запускать несколько разных методов(или одних и тех же) контракта с сохранением изменений состояния блокчейна между последовательными вызовами, т.е на каждый вызов утилиты эмулируется тестовая среда

пользовать официальный образ[26]). Также пришлось решить проблему с некорректной работой vyper-run[27].

4.2.3. Статический анализатор смарт-контрактов

На сегодняшней день статический анализ *vyper*-кода присутствует только в SmartCheck[28], поэтому было решено интегрировать его в плагин. Несмотря на то, что он написан на *java* (т.е существует возможность напрямую использовать методы), было решено использовать *docker*. Данное решение обусловлено в первую очередь проблемой с интеграцией (перекрывающиеся зависимости статического анализатора и самой платформы — XSLT-процессор). Преимуществом может служить то, что обновления для пользователя будут происходить незаметно (будет обновляться лишь образ).

Для взаимодействия с docker была выбрана библиотека *spotify-docker-client*, которая предоставляет всю необходимую функциональность. Существуют и другие библиотеки, но функциональность везде идентичная.

Недостатками данных решений служат накладные расходы, связанные с запуском *docker*-контейнеров. Также пользователю необходимо иметь установленный *docker*.

4.3. Архитектура решения и детали реализации

На рисунке 1 представлена диаграмма части архитектуры плагина, в которой реализована заявленная функциональность.

Модуль *gui* представляет собой набор UI компонент, которые отображаются перед пользователем при различных сценариях использования плагина. Более конкретно, он содержит в себе набор форм, которые появляются в результате взаимодействия пользователя с функциональностью статического анализатора, компилятора и утилитой *vyper-run*.

[26]https://github.com/ethereum/vyper/issues/1417

[27]https://github.com/status-im/vyper-debug/pull/19

[28]Статический анализатор для *Solidity* кода с помощью нахождения паттернов в контрактах; недавно появилась поддержка *Vyper* [18]

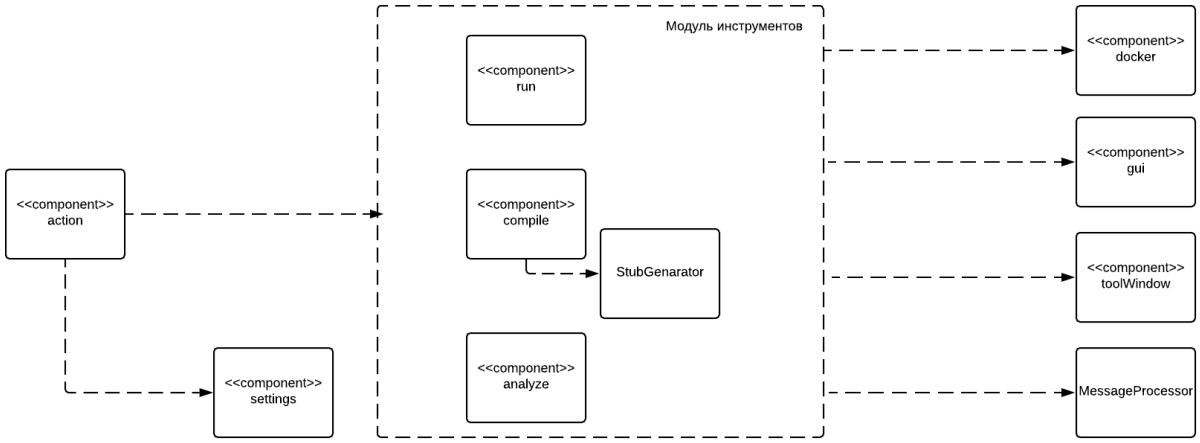

Рис. 1: Архитектура части плагина

Модуль *docker* содержит в себе набор классов, реализующих абстрактный класс IToolDocker. Они — обертки, предоставляющие функциональность соответствующих программ в *docker*-контейнерах. Классы, реализующие абстрактный класс IToolDocker, должны предоставить метод *exec()*, который возвращает ToolResult. Данный модуль создает дополнительный уровень абстракции, позволяя использующим их классам не зависеть от конкретных реализаций (*docker*-контейнеров и, соответственно, программ) и клиента-обертки над *docker*.

Модуль *toolWindow* отвечает за создание окон плагина, в которых отображаются результаты компиляции и запусков методов соответствующих контрактов. Он предполагает расширения засчет добавления постоянного окна, которое будет предоставлять функциональность для взаимодействия с *Ethreum* сетью.

StubGenerator отвечает за возможную (зависит от настроек пользователя) генерацию директории с результатами компиляции в виде файлов с учетом ключей компиляции (точнее, типов данных).

Модуль *settings* позволяет пользователю настроить ряд параметров для плагина (ключи компиляции, необходима ли генерации папки с созданием файлом с результатами компиляции и др.) посредством взаимодействия с UI формой, которая находится в настройках IDE. Настройки персистентные, сохраняются между запусками *IDE*.

MessageProccesor отвечает за отображения различного рода инфор-

мации пользователю через средства Intelij Platform. Отображаемые сообщения могут быть уведомлениями, например, об успешной компиляции или запуске метода, подсказками для возможного решения проблемы и др.

Модули *compile*, *run* и *analyze* отвечают за компиляцию и тестирование контрактов, используя классы из модуля *docker*, и перенаправляют результаты исполнения и уведомления пользователю к *StubGenerator*, *toolWindow* и *MessageProccesor*. Кроме того *compile*,*run* и *analyze*, могут оповещать средства языковой поддержки [29] о местах ошибок, для их последующего отображения в окне редактора.

Стоит отметить, что *run* предоставляет возможность запуска как одного метода контракта, так и их произвольной последовательности.

Модуль *analyze* может быть расширен добавлением других анализаторов, например, *MythX*, который анализирует байт-код.

Модуль *action* отвечает за обработку действий пользователя (вызов анализатора, запуск компиляции, тестирование контракта). Стоит отметить, что вызов методов контракта может осуществляться в окне редактора при взаимодействии пользователя с сигнатурой функции (требует взаимодействия с PSI)[28] — реализован *LineMarkerProvider* для выделения функций, которые можно запустить.

Конкретную реализацию можно найти здесь[30]. На текущей стадии плагин находится в альфа-версии. Ближайшая цель в рамках плагина — публикация в *JetBrains Plugins Repository* и расширение его функциональности.

[29]Это единственные точки прямого взаимодействия функционала языковой и инструментальной поддержки

[30]https://github.com/NikitaMishin/vyper-plugin

Заключение

В рамках курсовой работой были выполнены следующие задачи:

- изучена предметная область;
- проведен анализ существующих решений;
- интегрирован компилятор *Vyper*;
- реализована поддержка статического анализатора смарт-контрактов *SmartCheck*;
- поддержана возможность быстрого тестирования контрактов через *vyper-debug*;
- реализована альфа-версия плагина.

Дополнительным артефактом курсовой может служить совместная обзорная статья *Survey on blockchain technology, consensus algorithms, and alternative distributed technologies*, которая была принята для публикации в материалах конференции *SEIM 2019* (РИНЦ). Также было найдено несколько ошибок в реализации компилятора и предложены возможные улучшения [31], исправлены ошибки с некорректной работой утилиты *vyper-run* в *vyper-debug*[27].

Ближайшей целью в рамках разработки плагина является поддержка возможности взаимодействия с *Ethereum* сетью и его публикация в *JetBrains Plugins Repository*.

[31] https://github.com/ethereum/vyper/issues/1359, https://github.com/ethereum/vyper/issues/1417

Список литературы

[1] Access mode: https://cryptolization.com/ (online; accessed: 2018-12-14).

[2] 300m in cryptocurrency accidentally lost forever due to bug. — Access mode: https://www.theguardian.com/technology/2017/nov/08/cryptocurrency-300m-dollars-stolen-bug-ether (online; accessed: 2018-12-14).

[3] A 50 million hack just showed that the dao was all too human. — Access mode: https://www.wired.com/2016/06/50-million-hack-just-showed-dao-human/ (online; accessed: 2018-12-14).

[4] Alharby Maher, Moorsel Aad. Blockchain-based Smart Contracts: A Systematic Mapping Study. — Fourth International Conference on Computer Science and Information Technology (CSIT-2017). — 2017. — Accessed:2018-12-14. Access mode: https://arxiv.org/pdf/1710.06372.pdf.

[5] Atzei Nicola, Bartoletti Massimo, Cimoli Tiziana. A survey of attacks on ethereum smart contracts (sok) // International Conference on Principles of Security and Trust / Springer. — 2017. — P. 164–186.

[6] Avan-Nomayo Osato. Ethereum Surpasses Bitcoin in Number of Active Addresses. — Access mode: https://ethereumworldnews.com/ethereum-surpasses-bitcoin-in-number-of-active-addresses/ (online; accessed: 2018-05-31).

[7] Bartoletti Massimo, Pompianu Livio. An Empirical Analysis of Smart Contracts: Platforms, Applications, and Design Patterns // Financial Cryptography Workshops. — Vol. 10323 of Lecture Notes in Computer Science. — Springer, 2017. — P. 494–509.

[8] Buterin Vitalik. Ethereum: A next-generation smart contract and decentralized application platform. — 2014. — Accessed: 2018-

12-14. Access mode: https://github.com/ethereum/wiki/wiki/White-Paper.

[9] Do Smart Contract Languages Need to be Turing Complete? / Marc Jansen, Farouk Hdhili, Ramy Gouiaa, Ziyaad Qasem. — 2019. — 03.

[10] Elli Androulaki Artem Barger Vita Bortnikov, other. Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains. — 2018. — Accessed:2018-12-14. Access mode: https://arxiv.org/pdf/1801.10228.pdf.

[11] Formal Verification of Smart Contracts: Short Paper / Karthikeyan Bhargavan, Antoine Delignat-Lavaud, Cédric Fournet et al. // PLAS@CCS. — ACM, 2016. — P. 91–96.

[12] Goodman L.M. Tezos — a self-amending crypto-ledger White paper. — 2014. — Accessed:2018-12-14. Access mode: https://tezos.com/static/papers/white_paper.pdf.

[13] Hearn Mike. Corda: A distributed ledger. — 2016. — Accessed:2018-12-14. Access mode: https://www.corda.net/content/corda-technical-whitepaper.pdf.

[14] K framework evm-semantics. — Access mode: https://github.com/kframework/evm-semantics.

[15] Morris David Z. Bitcoin is not just digital currency. It's Napster for finance. — 2014. — http://fortune.com/2014/01/21/bitcoin-is-not-just-digital-currency-its-napster-for-finance/.

[16] Nakamoto Satoshi. Bitcoin: A Peer-to-Peer Electronic Cash System. — 2008. — Dec. — Accessed: 2018-12-14. Access mode: https://bitcoin.org/bitcoin.pdf.

[17] The New Viper Smart Contract Programming Language. — Access mode: https://ethereumclassic.github.io/blog/2017-03-13-viper/.

[18] SmartCheck — the First Vyper Security Tool. — Access mode: https://blog.smartdec.net/smartcheck-the-first-vyper-security-tool-a6a501904436.

[19] Vyper. — Access mode: https://github.com/ethereum/vyper.

[20] Wood Gavin. Ethereum: A secure decentralised generalised transaction ledger EIP-150 REVISION (759dccd - 2017-08-07). — 2017. — Accessed: 2018-12-14. Access mode: https://ethereum.github.io/yellowpaper/paper.pdf.