Mina: Децентрализованная криптовалюта в масштабе

Жозеф Бонно[1], Исаак Меклер[2], Ванишри Рао, и Эван[2] Шапиро[2]

[1]Нью-Йоркский Университет

[2]O(1) Labs

Март 2020 г.

Аннотация

Мы вводим понятие краткой цепочки блоков, реплицируемого конечного автомата, в котором каждый переход состояния (блок) может быть эффективно проверен за постоянное время независимо от количества предыдущих переходов в системе. Традиционные блокчейны требуют времени проверки, пропорционального количеству переходов. Мы покажем, как построить краткий блокчейн, используя рекурсивно составленные краткие интерактивные аргументы знания (SNARK). Наконец, мы создаем экземпляр этой конструкции для реализации Mina, платежной системы (криптовалюты), использующей краткий блокчейн. Mina предлагает платежную функциональность, аналогичную биткоину, со значительно более быстрым временем проверки 200 мс, что позволяет легковесным клиентам и мобильным устройствам выполнять полную проверку истории системы.

1 Введение

Bitcoin и другие распределенные платежные системы (также называемые криптовалютами или просто блокчейнами) призваны обеспечить децентрализованную систему для совершения и проверки платежей. Однако для традиционных криптовалют, включая Bitcoin, децентрализация достигается за счет масштабируемости, поскольку каждый узел должен обрабатывать всю историю системы при присоединении к сети. Асимптотически проверка блокчейна, содержащего t транзакции, требует $\Omega(t)$ времени (обычно больше, чем линейное, поскольку для разрешения ссылок на транзакции во время проверки требуется бухгалтерский учет). На момент написания этой статьи размер блокчейна Bitcoin превышает 250 ГБ и содержит более 500 млн транзакций (см. рис. 1). Загрузка и проверка этой истории занимает несколько дней на обычном ноутбуке.

Эти требования к ресурсам удерживают большинство пользователей от запуска полного узла, который хранит и проверяет блокчейн. Как видно на рисунке 2, количество полных узлов в Bitcoin не растет, несмотря на его растущую популярность с течением времени. Вместо этого большинство пользователей запускают легкий узел, проверяющий только заголовки блоков, но не транзакции, или сверхлегкий узел, ничего не проверяющий и полагающийся на доверенный сервер.

Это подрывает децентрализацию, поскольку большинство клиентов полагаются на доверие, а не на независимую проверку. Это также снижает производительность: размер блока (и, следовательно, пропускная способность транзакций) частично искусственно ограничивается, чтобы уменьшить нагрузку на проверку.)

Рисунок 1: Рост блокчейна Bitcoin с течением времени, в ГБ. Источник: www.blockchain.com.

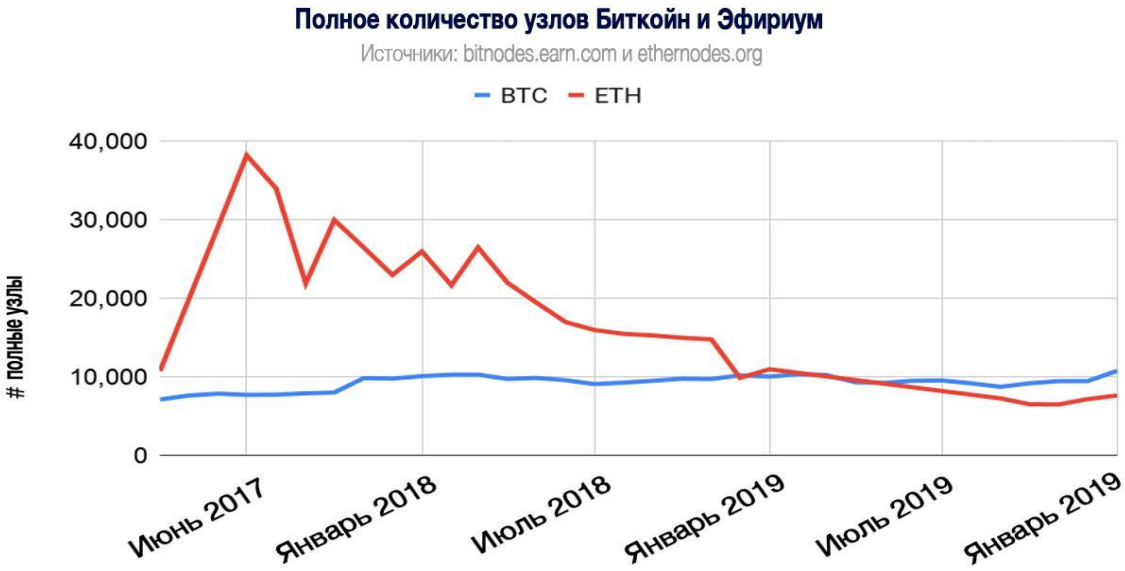


Рисунок 2: Расчетное количество полных узлов, участвующих в сетях Биткойн и Эфириум, с течением времени. Источник: www.bitnodes.earn.com и www. Ethernodes.org.

В этой работе наша цель — разработать децентрализованную платежную систему, которая предлагает эффективную проверку истории системы с момента возникновения, не полагаясь на какие-либо внешние советы. В частности, мы стремимся обеспечить постоянную времени проверки (O (1)) в количестве транзакций; мы называем такую цепочку блоков **короткой цепочкой блоков**.

Мы достигаем этой цели, включая краткие доказательства действительности состояния в каждом блоке. Как правило, можно вычислить краткий неинтерактивный аргумент знания (SNARK) любого утверждения NP, включая, например, то, что система, заявленная как зафиксированная текущим блоком в цепочке блоков, может быть достигнута из исходного состояния с помощью серия действительных транзакций в системе. Этот (большой) список транзакций свидетельствует о том, что текущий блок действителен. Однако вычисление нового доказательства достоверности всей системной истории для каждого блока было бы непомерно дорогим. Вместо этого мы используем методы постепенно вычисляемых SNARK, чтобы гарантировать, что стоимость вычисления доказательства для каждого блока пропорциональна только количеству транзакций, добавленных после предыдущего блока.

Мы реализуем понятие краткой цепочки блоков и представляем **протокол Mina**. Mina — это блокчейн, ориентированный на платежи, предлагающий функциональность, аналогичную Bitcoin, но с другой семантикой транзакций. В частности, Mina использует модель на основе учетных записей (как в Ethereum [24]) (вместо модели UTXO, как в Bitcoin[18] и других [19]), в которой текущее состояние блокчейна представляет собой список всех учетных записей. балансы, а не список неизрасходованных монет (UTXO).

Каждый блок содержит обязательство по этому состоянию (в дереве Меркла), а не по всему состоянию. Поэтому полному узлу не обязательно хранить все состояние, но он может эффективно проверять балансы счетов, учитывая только обязательство состояния в последнем заголовке блока. Однако прувер в нашей системе (примерно эквивалентный майнеру в Bitcoin) должен хранить полное состояние, поскольку он является частью свидетеля при проверке достоверности новых блоков.

Для протокола консенсуса Mina мы представляем первый доказуемо безопасный протокол консенсуса Proof-of-Stake (PoS) для кратких блокчейнов под названием Ouroboros Samasika. Обратите внимание, что готовый механизм консенсуса не обязательно совместим с краткой структурой блокчейна, поскольку способ достижения консенсуса при наличии нескольких конкурирующих цепочек может полагаться на произвольную историю транзакций, заставляя узлы хранить всю историю транзакций. На самом деле, это естественный подход к механизмам консенсуса, поскольку информация, необходимая для того, чтобы отличить честную цепочку от нечестной, скорее всего, будет включать в себя детали в точке разветвления; поскольку сторона может узнать о форке спустя долгое время после того, как он произошел, ей может потребоваться сохранить всю историю, чтобы помочь в процессе выбора цепочки. Это действительно имеет место в известных механизмах консенсуса PoS [16, 12, 3]. Кроме того, другие механизмы консенсуса PoS полагаются на доверенный внешний совет для начальной загрузки [11].

Конкретно, в нашей текущей реализации размер доказательства состояния составляет всего 864 байта, и его проверка занимает около 200 мс. Таким образом, любое устройство, которое может поддерживать этот уровень вычислений, например, современные смартфоны, может проверить текущее состояние системы без надежного совета.

Помимо постепенно вычисляемых SNARK, мы используем несколько оптимизаций, наиболее важной из которых является состояние параллельного сканирования. На высоком уровне это повышает пропускную способность транзакций за пределами последовательно вычисляемых доказательств. Грубо говоря, идея состоит в том, чтобы поставить в очередь все блоки, которые еще нужно включить в доказательство, и распределить их доказательство между параллельными пруверами. Мы также вводим специальную очередь последних транзакций, чтобы уменьшить задержку подтверждения транзакций ниже пределов, налагаемых минимальным временем проверки. Кроме того, мы вводим специальную структуру поощрения для максимального участия доказывающего в сети.

1.1 Наш вклад

Подводя итоги, наш вклад:

- Мы формализуем понятие *краткого блокчейна*. Это понятие может представлять самостоятельный интерес для альтернативных конструкций емких блокчейнов.

- Мы представляем подход к построению краткой цепочки блоков для общих функций, смоделированных как реплицированные конечные автоматы с использованием последовательно вычисляемых SNARK.

- Мы представляем конкретную реализацию нашего подхода для конкретной функциональности платежной системы под названием Mina.

- Мы представляем Ouroboros Samasika, доказуемо безопасный согласованный протокол PoS, который является адаптивно безопасным и предлагает начальную загрузку с самого начала.

- Мы вводим понятие *состояния параллельного сканирования*, чтобы сократить время подтверждения транзакции сверх ограничений, налагаемых конструкцией доказательства.

- Мы представляем отчет об оценке эффективности выполнения протокола с участием публичного сообщества.

2. Краткий блокчейн

В этом разделе мы вводим понятие кратких блокчейнов.

Основные концепции блокчейна. Мы начнем с напоминания определений некоторых базовых концепций блокчейна [12]. Это поможет в определении кратких блокчейнов.

Определение 2.1 (состояние, блок-доказательство, блок-производитель, блок, блокчейн, генезис-блок). Состояние — это строка $st \in \{0, 1\}^{\lambda}$. Доказательство блока — это значение (или набор значений) $\boldsymbol{\pi_i^B}$, содержащее информацию для проверки правильности блока. Каждый блок связан с уникальной стороной, называемой производителем блока. Блок $B_i = (sn_i, st_i, \boldsymbol{\pi_i^B}, d_i, \text{b-pk}_i, \text{b-sig}_i)$,, сгенерированный с порядковым номером $sn_i \in \mathbb{N}$, содержит текущее состояние st_i, доказательство блока $\boldsymbol{\pi_i^B}$, data $d_i \in \{0, 1\}*$, открытый ключ производителя блока b-pki и подпись b-sigi on $(sn_i, st_i, \boldsymbol{\pi_i^B}, d_i)$, по отношению к b-pki.
Блокчейн — это последовательность блоков $C = (B_1, \ldots, B_n)$, связанных строго возрастающей последовательностью порядковых номеров. Первый блок B_1 называется блоком генезиса. Длина $\text{len}(C) = n$ — это количество блоков в нем.

Краткие блокчейны. Теперь мы готовы представить определение краткого протокола блокчейна. В определении также будет введено понятие сводки цепочки блоков, которая на высоком уровне представляет собой некоторую сводку цепочки блоков, такую, что сводка действительна тогда и только тогда, когда действительна цепочка блоков. Концепция блокчейна, лежащая в основе резюме блокчейна, не будет очевидна из определения самого краткого протокола блокчейна. Однако это будет отражено в понятии извлекаемости цепи в определении 2.4.

Определение 2.2 (Краткий блокчейн-протокол). Краткий протокол блокчейна M характеризуется набором из пяти алгоритмов PPT (VerifyConsensus, UpdateConsensus, VerifyBlock, UpdateChain, VerifyChain), синтаксически определенных следующим образом.

- VerifyConsensus (consensusState, consensusProof) → T/⊥: этот алгоритм
  rithm принимает в качестве входных данных консенсусное состояние и консенсусное доказательство, проверяет в соответствии с некоторым понятием правильности и выводит t или ⊥ соответственно.

- UpdateConsensus(consensusState, consensusProof) → nextConsensusState : этот алгоритм также принимает в качестве входных данных консенсусное состояние и консенсусное доказательство и выводит обновленное состояние консенсуса.

- VerifyChainSummary(S_i) → T/⊥: этот алгоритм проверяет, данная сводка блокчейна S_i действительна или нет.

- VerifyBlock(S_{i-1}, B_i) → T/⊥: этот алгоритм проверяет, является ли данный блок B_i действительным по отношению к данной сводке блокчейна S_{i-1}. . В рамках проверки проверяется, что VerifyConsensus (consensusStatei-1, консенсусProofi → t, где S_{i-1} содержит консенсусStatei-1 и $\boldsymbol{\pi_i^B}$ содержит consensusProof, где B_i = (˙,˙, $\boldsymbol{\pi_i^B}$, ˙, ˙, ˙)

- UpdateChainSummary(S_{i-1}, B_i) → S_i : этот алгоритм принимает сводку блокчейна S_{i-1} и новый блок Bi и выводит обновленную сводку блокчейна S_i..

Протокол удовлетворяет следующему свойству краткости.

Краткость. Каждый из алгоритмов VerifyBlock, VerifyChainSummary и VerifyConsensus выполняется за время poly(λ). Кроме того, размер сводки по блокчейну S_i в любой момент времени t_i имеет размер poly(λ) (т. е. постоянен в числе сводных обновлений цепочки).

Замечание 2.1 (механизм консенсуса). Говорят, что пара алгоритмов (VerifyConsensus, UpdateConsensus) составляет механизм консенсуса. Ниже приведены некоторые примеры того, как понятие может быть конкретизировано. Для протоколов проверки работоспособности (например, Bitcoin) состояние консенсуса будет содержать несколько предыдущих целевых показателей сложности и времени блока (из которых можно вычислить текущую целевую сложность), а доказательство консенсуса будет содержать само доказательство выполнения работы вместе с новое время для обновления состояния. Для механизма подтверждения ставок в стиле Ouroboros Praos [12] состояние консенсуса будет содержать текущее случайное начальное число, (корень Merkle) ставок текущей эпохи и некоторую информацию о предыдущих блоках и времени блоков. Доказательство консенсуса будет содержать открытый ключ и доказательство оценки проверяемой случайной функции (VRF), соответствующее пороговому целевому значению, соответствующему этому открытому ключу, и ставкам, указанным в состоянии консенсуса.

Типы ролей. Согласно приведенному выше определению, в кратком блокчейне есть три вида ролей. (В зависимости от реализации могут быть дополнительные роли.)

1. **Полный узел**: в этой роли сторона отслеживает сводку блокчейна и проверяет ее.

2. **Производитель блоков**: в этой роли сторона создает блок.

3. **Производитель резюме блокчейна**: в этой роли сторона генерирует сводки блокчейна.

Обратите внимание, что существенным преимуществом краткого блокчейна является то, что любая сторона с разумными ресурсами может быть полным узлом благодаря свойству краткости. То есть лаконичный блокчейн не требует роли легких клиентов, чтобы справиться с растущими размерами блокчейна.

Взаимосвязь между сводкой блокчейна и базовым блокчейном. Определив краткую цепочку блоков с точки зрения сводок цепочки блоков, мы теперь покажем, как сводки связаны с лежащими в основе цепочками блоков. Грубо говоря, мы хотели бы, чтобы сводные данные по блокчейну наследовали достоверность базовых блокчейнов. То есть сводка действительна тогда и только тогда, когда действительна базовая цепочка блоков.

Кроме того, учитывая сводку блокчейна, мы приходим к его базовому блокчейну через понятие извлекаемости. В частности, мы определяем извлечение рекурсивно; то есть, имея сводку цепочки блоков с порядковым номером i, экстрактор (используя некоторую дополнительную информацию) извлекает сводку цепочки блоков с серийным номером i − 1 и блок B_i, в котором все компоненты удовлетворяют необходимым проверочным тестам. Дополнительная информация, которую использует экстрактор, — это стенограмма выполнения, которую мы называем трассировкой выполнения, формально определенной ниже.

Определение 2.3 (трассировка выполнения, сводка по блокчейну в трассировке выполнения). Для (адаптивного) противника $\mathcal{A}$ и среды $\mathcal{Z}$ трассировка $\mathcal{E}$ протокола блокчейна Π набором сторон $\mathcal{U}$ с параметром безопасности λ представляет собой расшифровку, включающую входные данные, предоставленные $\mathcal{Z}$, случайные монеты сторон и случайные монеты противника. Эти данные определяют всю динамику протокола: отправленные и доставленные сообщения, внутренние состояния сторон на каждом шаге и набор коррумпированных сторон на каждом шаге. Обозначим след через $\mathcal{E} \leftarrow \Pi(1^{\lambda}, \mathcal{U})$ или просто $\mathcal{E} \leftarrow \Pi(\mathcal{U})$.

Для каждого протокола блокчейна существует алгоритм CurrChain, такой, что для каждого набора сторон PPT $\mathcal{U}$, $\mathcal{E} \leftarrow \Pi(\mathcal{U})$, времени t, честной стороны $P \in \mathcal{U}$ мы имеем, что CurrChain выводит действительную сводку блокчейна; т. е. CurrChain($\mathcal{E}$, P, t) → $\mathcal{S}$ и

VerifyChainSummary($\mathcal{S}$) → t. Говорят, что S является сводкой блокчейна в представлении $P \in \mathcal{U}$ в момент времени t. Сводка блокчейна в трассировке выполнения $\mathcal{E}$ — это сводка блокчейна $\mathcal{C}$ в представлении любой добросовестной стороны в любой момент времени t; мы обозначаем это через $\mathcal{S} \in \mathcal{E}$.

Теперь определим понятие извлекаемости цепи. В этом определении используется понятие «порядковый номер сводки блокчейна». Интуитивно понятно, что это просто натуральное число j, которое представляет количество блоков в базовой цепочке блоков. Обозначается в нижнем индексе как S_j.

Определение 2.4 (Извлекаемость цепочки). Говорят, что краткий протокол блокчейна Π = (VerifyConsensus, UpdateConsensus, VerifyBlock, UpdateChain, VerifyChain) удовлетворяет извлекаемости цепочки, если следующая вероятность $\mathrm{Adv}_{\Pi, \mathcal{U}}(1^{\lambda})$ пренебрежимо близка к 1 для каждого $\mathcal{U} = \{\mathcal{A}_i\}_i$, множества Алгоритмы PPT. Для каждого $\mathcal{A}_i$, существует алгоритм PPT $\mathrm{Ext}_{\mathcal{A}_i}$, называемый экстрактором, и $\mathrm{Adv}_{\Pi, \mathcal{U}}$ определяется следующим образом.

$$\mathsf{Adv}_{\Pi,\mathcal{U}}(1^\lambda) := \Pr\left[\begin{array}{l} \mathrm{VerifyChainSummary}(\mathcal{S}_{j-1}) = \top \\ \wedge \\ \mathrm{VerifyBlock}(\mathcal{S}_{j-1}, B_j) = \top \\ \wedge \\ B_1 \text{ является блоком Genesis} \\ \wedge \\ \mathcal{S}_0 \text{ является пустой строкой} \end{array} : \begin{array}{c} \mathcal{E} \leftarrow \Pi(\mathcal{U}) \\ \forall \mathcal{S}_j \in \mathcal{E}, \exists \mathcal{A}_i \in \mathcal{U} \\ (\mathcal{S}_{j-1}, B_j) \leftarrow \mathsf{Ext}_{\mathcal{A}_i}(\mathcal{E}, \mathcal{S}_j, r) \end{array}\right]$$

где r — случайные монеты $\mathcal{A}_i$.

Определение 2.5 (Блокчейн, лежащий в основе резюме блокчейна). Пусть Π будет протоколом блокчейна, который удовлетворяет извлекаемости цепочки. Пусть $\mathcal{E}$ — выполнение протокола множеством сторон $\mathcal{U}$. Пусть $P \in \mathcal{U}$ — честная сторона, активная с начала протокола. Пусть $\mathcal{S}_\ell$ будет блокчейном в $\mathcal{E}$.. Для каждого $1 \le i \le \ell$ пусть B_i — блок, гарантированный свойством цепной извлекаемости. Последовательность $(B_1, \ldots, B_\ell)$ называется блокчейном, лежащим в основе $\mathcal{S}_\ell$.

2.1 Свойства безопасности краткого блокчейна

Теперь мы задействуем свойства безопасности краткого блокчейна. Вместо сводок блокчейна свойства относятся к базовому блокчейну, гарантированному свойством извлекаемости цепочки.

Рассмотрим блокчейн-протокол Π и выполнение $\mathcal{E}$. Пусть C будет лежащим в основе блокчейном сводки блокчейна $\mathcal{S}$ в $\mathcal{E}$. Напомним следующие свойства, впервые строго сформулированные в [14]. Предположим, что время разделено на предопределенные слоты.

Общий префикс (CP); с параметрами $k \in \mathbb{N}$. Блокчейны C_1, C_2, соответствующие двум тревожным сторонам в начале слотов $sl_1 \le sl_2$, таковы, что $C_1^{\lceil k} \preccurlyeq C_2$, где $C_1^{\lceil k}$ обозначает блокчейн, полученный удалением последних k блоков из $C_1 \preccurlyeq$, а “обозначает префиксное отношение.

Цепной рост (CG); с параметрами $\tau \in (0, 1]$ и $s \in \mathbb{N}$. Рассмотрим C, цепочку блоков, которой владеет тревожная сторона в начале слота sl. Пусть sl_1 и sl_2 — два предыдущих слота, для которых $sl_1 + s \le sl_2 \le sl$, поэтому sl_1 предшествует sl_2 не менее чем на s. Тогда $|C[sl_1, sl_2]| \ge \tau \cdot s$. Мы называем τ коэффициентом скорости.

Цепное качество (CQ); с параметрами $\mu \in (0, 1]$ i $k \in \mathbb{N}$. Рассмотрим любую часть длины по крайней мере блокчейну, соответствующую стороне предупреждения в начале слота; отношение блоков, происходящих от сторон предупреждения, в этой части составляет по меньшей мере μ, называемый коэффициентом качества цепи.

3 Предварительные

В этом разделе мы даем несколько необходимых определений систем SNARK, которые мы будем использовать для создания краткой цепочки блоков.

Обозначения. Мы используем аббревиатуру PPT для обозначения вероятностного полиномиального времени. Мы используем λ для обозначения параметра безопасности.

Определение 3.1 (SNARK). Пусть $R = \{(\phi, w)\}$ — полиномиальное отношение утверждений ϕ и свидетелей w. Краткий неинтерактивный AR-аргумент знаний для R — это четверка алгоритмов (sSetup, sProve, sVerify, sSim), которая является полной, краткой и достоверной (определение приведено ниже) и работает следующим образом:

- (srs, r) ← sSetup (R): Алгоритм установки генерирует структурированную случайную строку srs и лазейку r.
- π ← sProve(srs, ϕ, w): алгоритм доказательства генерирует доказательство u.
- T/⊥ ← sVerify(srs, ϕ, π): алгоритм верификатора проверяет данное доказательство.
- π ← sSim(srs, ϕ, τ):симулятор PPT имитирует доказательство без свидетеля, но с использованием люка.

Полнота. В нем просто говорится, что при наличии истинного утверждения доказывающий со свидетелем может убедить проверяющего. То есть для каждого кожного (srs, τ) ← **sSetup**(R) і π ← **sProve**(srs, ϕ, w) мы имеем T ← sVerify(srs, ϕ, π).

Лаконичность. В нем говорится, что размер доказательства | π | является poly(λ).

Здравость знаний. В нем говорится, что всякий раз, когда кто-то приводит действительный аргумент, можно извлечь действительного свидетеля из его внутренних данных. Формально для каждого противника PPT $\mathcal{A}$ существует экстрактор PPT $\mathcal{X}_{\mathcal{A}}$, такой что следующая вероятность пренебрежимо мала в h:

$$\Pr\left[\begin{array}{ccc} (\mathrm{srs}, \tau) \leftarrow \mathsf{sSetup}(R) & & (\phi, w) \notin R \\ (\phi, \pi) \leftarrow \mathcal{A}(\mathrm{srs}) & : & \wedge \\ w \leftarrow \chi_{\mathcal{A}}(\mathrm{trans}_{\mathcal{A}}) & & \mathsf{sVerify}(\mathrm{srs}, \phi, \pi) \rightarrow \top \end{array}\right]$$

Извлекаемая симуляция SNARK, извлекаемые с помощью моделирования. Извлекаемый с помощью моделирования SNARK — это SNARK, который обеспечивает более высокий уровень безопасности, а именно возможность извлечения с помощью моделирования. Понятие извлекаемости с помощью симуляции аналогично понятию достоверности знаний, за исключением того, что противник также может видеть смоделированные доказательства.

Подписи знаний (SoK). SoK — это обобщение цифровых подписей путем замены открытого ключа экземпляром на языке NP. Формальное определение см. в [15]. Понятие SoK связано с понятием извлекаемых с помощью моделирования не интерактивных аргументов с нулевым разглашением, таких как SNARK. Фактически в [15] показано, что первое может быть построено на основе второго. В этой работе мы полагаемся на SoK, построенные с использованием SNARK, что позволяет использовать краткость таких SoK.

4. Mina: краткий блокчейн на основе рекурсивных SNARK

В этом разделе мы представляем краткую конструкцию блокчейна под названием Mina, основанную на SNARK. На высоком уровне достоверность последовательности переходов в блокчейне подтверждается с помощью SNARK. Затем доказательство блокчейна состоит из этого SNARK и опускает подробный список блоков, поскольку проверка SNARK проверяет встроенные блоки. Краткость SNARK обеспечивает лаконичность блокчейна.

Обратите внимание, что блокчейн динамичен, и в него постоянно добавляются новые блоки. Тем не менее, мы хотели бы обеспечить краткость в любой момент времени. Поэтому по мере «*роста*» блокчейна мы вычисляем новое доказательство SNARK, которое проверяет не только новые блоки, но и само существующее доказательство SNARK. Понятие SNARK-доказательства, свидетельствующего о верифицируемости другого SNARK-доказательства, — это понятие «инкрементально-вычислимого SNARK» [23, 7, 5].

Сначала мы уточним конструкцию SNARK, а затем продемонстрируем, как ее можно использовать для создания краткой цепочки блоков.

4.1 Инкрементально вычисляемые SNARK

Теперь мы вспомним понятие инкрементно-вычисляемых SNARK, описанное в различных работах [23], [7] и [5]. Вместо формулировки конструкции на языке инкрементально проверяемых вычислений, как в [23], или на языке систем PCD (доказательных данных), как в [7] и [5], мы предпочитаем описывать ее в терминах состояний. переходных систем, поскольку он более четко отражает применение создания краткой цепочки блоков.

Сначала напомним определение системы переходов состояний.

Определение 4.1 (Система перехода состояний). Система переходов состояний представляет собой кортеж (Σ, T, Update), где Σ — набор состояний, T — набор переходов, а Update — (недетерминированная) поливременная вычислимая функция Update: $\mathrm{T}\times\Sigma \to \Sigma$. Обновление также может «сгенерировать исключение» (т. е. не создать новое состояние для определенных входных данных). Более того, элементы Σ и T должны быть представлены битовыми строками длины $\mathrm{poly}(\lambda)$.

Теперь мы определим SNARK для систем перехода состояний. На высоком уровне нам нужны доказательства $\mathrm{poly}(\lambda)$-размера (которые проверяемы за $\mathrm{poly}(\lambda)$-время), подтверждающие утверждения вида «существуют состояние 1 и последовательность переходов $t_1, \ldots, t_k \in \mathrm{T}$ такие, что $\mathrm{Update}(t_k, \mathrm{Update}(t_{k-1}, \ldots, \mathrm{Update}(t_1, \sigma_1))) = \sigma 2$”. Другими словами, нам нужны краткие свидетельства существования последовательностей перехода состояний, соединяющих два состояния. Применение к цепочкам блоков следующее: мы возьмем наше состояние как базу данных учетных записей (вместе с некоторыми метаданными, необходимыми для правильной проверки новых блоков), а переходы — как блоки.

Определение 4.2 (Инкрементально вычисляемые SNARK). Инкрементально вычисляемый SNARK для системы переходов состояний (Σ, T, Update) представляет собой набор алгоритмов

(sSetup, sProve, sVerify, sSim) такие, что выполняется следующее. Подавление генерации параметров и передача параметров в sProve и sVerify,

1. (sSetup, sProve, sVerify, sSim) є SNARK.
   (sSetup, sProve, sVerify, sSim) є SNARK для соотношения R = {(σ_{i+k}), σ_i, t_{i+1}, . . . , t_{i+k})}, где σ_{i+k} = Update(t_{i+k}, Update(t_{i+k-1}, . . . , Update (t_{i+1}, σ_i))) для какого-либо *k*.

2. (sSetup, sProve, sVerify, sSim) краток.
   Каждое честно сгенерированное доказательство имеет размер poly(*ʎ*), и для любых $\boldsymbol{\pi}$, $\boldsymbol{\sigma}$ мы имеем, что **sVerify**($\boldsymbol{\sigma}$, $\boldsymbol{\pi}$) выполняется за время poly(*ʎ*).

4.1.1 Инкрементно-вычисляемые SNARK с использованием рекурсивной композиции доказательства

Наивная рекурсивная композиция теоретически жизнеспособна, поскольку для SNARK проверка доказательства асимптотически дешевле, чем простая проверка соответствующего утверждения NP. Однако это чрезвычайно дорого. Хотя верификаторы SNARK выполняются довольно быстро — всего за несколько миллисекунд на настольном компьютере, создание доказательства SNARK для подтверждения принимающей схемы верификатора обходится дорого. Это связано с тем, что выполнение верификаторов по-прежнему требует миллионов шагов вычислений, что нецелесообразно даже для одного уровня рекурсии, как объяснено в [5].
Чтобы решить эту проблему, мы используем технику «цикла эллиптических кривых» (как описано в [5]), в которой две конструкции SNARK — классически называемые

Tick и Tock — разработаны таким образом, что каждая может эффективно проверять доказательства друг друга. Затем мы определяем SNARK Tick и Tock для получения «бинарного дерева доказательств» следующим образом. Tick SNARK используется для сертификации переходов состояний в «основе» дерева. Затем, чтобы обеспечить эффективное слияние этих доказательств, каждое из них «оборачивается» с помощью Tock SNARK. Затем два доказательства Tock объединяются с помощью Tick SNARK.

Поэтому обратите внимание, что нам понадобятся два Tick SNARK — один для доказательства переходов состояний, а другой для слияния двух доказательств Tock. И нам понадобится один Tock SNARK, чтобы обернуть доказательство Tick в доказательство Tock. Более формально:

1. **База SNARK**. SNARK на основе тиков для сертификации переходов одного состояния, который мы будем называть «базовым» SNARK.

Утверждение: (σ_1, σ_2) ∈ Z^2.

Свидетель: $t \in \mathrm{T}$.

Вычисление: существует t ∈ T такое, что Update(t, σ_1) = σ_2.

Обозначим доказательство через $\sigma_1 \rightarrow$Tick σ_2.

2. **Слияние SNARK**. SNARK на основе тиков для слияния двух доказательств Tock, который мы будем называть «слиянием» SNARK.

Утверждение: (σ_1, σ_3) ∈ Z^2

Свидетель: $\sigma_2 \in \Sigma$ и доказательства Тока π_1, π_2.
Вычисление: существуют $\sigma_2 \in \Sigma$ и доказательства Tock π_1, π_2 такие, что
$\text{Verify}_{\text{Tock}}((\sigma_1, \sigma_2), \pi_1)$ и $\text{Verify}_{\text{Tock}}((\sigma_2, \sigma_3), \pi_2)$
Обозначим доказательство через $\sigma_1 \rightarrow$ Отметьте σ_3. σ_1 до σ_2 и доказательство SNARK, подтверждающее существование переходов из σ_2 в σ_3.

3. **Пленка SNARK**. SNARK на основе Tock для обертывания доказательства Tick, который мы будем называть «оберткой» SNARK.

   Утверждение: $(\sigma_1, \sigma_2) \in \Sigma^2$.

   Свидетель: доказательство Tick π.

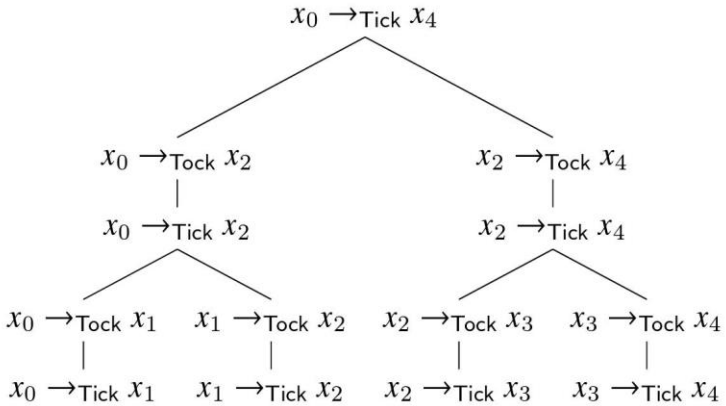
   Вычисление: существует тиковое доказательство u такое, что VerifyTick $((\sigma_1, \sigma_2), u)$.
   Мы будем обозначать доказательство как $\sigma_1 \rightarrow$Tock σ_2. Этот SNARK просто оборачивает Tick SNARK в Tock SNARK, чтобы другой Tick SNARK мог проверить
   это эффективно.

4.1.2 Пример системы перехода

Чтобы проиллюстрировать это, мы покажем, как это применимо для подтверждения утверждений в очень простой системе переходов, где каждое состояние является просто хэшем H предыдущего состояния. Предположим, что текущее состояние $\underbrace{x' = \mathrm{H}(\mathrm{H}(\ldots \mathrm{H}(\mathrm{x}) \ldots))}_{k}$ для некоторого k, начиная с

из исходного состояния x. Мы используем вышеприведенную технику, когда наше состояние Σ является объединением домена и диапазона H, T является единственным набором, содержащим пустую строку, и функцией обновления является Update(t,x) = H(x).

Это дает нам SNARK для подтверждения того, что существует последовательность переходов (t_1, . . . , t_k), такая что Update(t_k, Update(t_k-1, . . . , Update(t_1, x), ...)) = x',, поскольку Update(t, y) = H(y), дает нам именно то, что мы хотим. Для строк x_0, x_4 з $H(H(H(H(x_0)))) = x_4$ дерево доказательств SNARK выглядит следующим образом:



4.2 Mina: краткий блокчейн с использованием последовательно вычисляемых SNARK

В этом разделе мы представляем протокол Mina, сжатый блокчейн, основанный на SNARK с поэтапным вычислением. Интуитивно понятно, что обновление блокчейн

рассматривать как систему перехода между состояниями, и, следовательно, SNARK с поэтапным вычислением (которые просто SNARK для систем перехода между состояниями) могут сделать создание коротких блокчейнов.

4.2.1 Наша конструкция

В этом разделе мы представляем протокол Mina. В частности, мы обсудим детали общих функций, полных по Тьюрингу, которые преобразуют базу данных. Затем, в разделе 5, мы создадим экземпляр протокола с функциями платежей.

На высоком уровне мы будем рассматривать блокчейн как функцию перехода состояния. Рассмотрим, например, модель UTXO (вывод неоплаченных транзакций), в которой каждая сторона имеет «счет» с некоторым «*балансом*», как в Bitcoin. Состояние блокчейна — это база данных (например, дерево Merkle) всех остатков на счетах. Переход – это переход части баланса с одного счета на другой. Хотя это всего лишь пример, наш протокол является общим и рассматривает любое множество состояний Σ' и полную по Тьюрингу функцию перехода Update′ с некоторым множеством переходов T′; то есть мы начинаем с (Σ', T′, Update′).

Тогда протокол Mina для (Σ', T′, Update′) строится следующим образом. Мы использовать наш согласованный протокол, а именно Ouroboros Samasika, который мы представляем в разделе 7. Мы объединим (Σ', T′, Update′) и согласованный протокол для создания новой системы перехода состояний (Σ, T, Update), в основном для включения консенсуса проверка в функции обновления. Используется инкрементально вычисляемый SNARK для (Σ, T, Update), и доказательства подтверждают, что текущее состояние вычисляется правильно. Сводка блокчейна состоит просто из состояния в Σ и доказательства. Производитель резюме блокчейна будет просто доказывающим, а полному узлу нужно будет только выполнить доказательную проверку, чтобы проверить правильность блокчейна.

Описав протокол на интуитивном уровне, мы теперь обсудим детали. Учитывая (Σ', T′, Update′) и устойчивую к коллизиям хеш-функцию H, компоненты протокола следующие.

The Mina Protocol

Протокол Mina состоит из следующих компонентов.

- **Механизм консенсуса:** (UpdateConsensus, VerifyConsensus): Механизм консенсуса — это протокол Ouroboros Samasika, который мы представляем в разделе 7. В Ouroboros Samasika VerifyConsensus запускается во времени poly (h), как требуется.
- **Блоки:** рассмотрим переход $t'_i \in Tau'$ соответствующего блока σ'_{i-1}. Который действует на состояние $B_i = (\mathrm{sn}_i, \mathrm{st}_i, \pi_i^B, d_i, \text{b-pk}_i, b\text{-sig}_i)$ построен с $\mathrm{st}_i = \sigma'_{i-1}$, $\pi_i^B = \mathrm{consensusProot}_i$ and $d_i = t'_i$.
- **Система перехода состояний для SNARK:** рассмотрим систему перехода состояний $(\Sigma, \mathrm{T}, \mathrm{Update})$ определяется следующим образом: $\Sigma = \{\mathrm{H}(\sigma'), \mathrm{consensusState}\}_{\sigma' \in \Sigma', \mathrm{consensusState}}$. Переход — это блок.

Функция $\mathrm{Update}(B_i, \sigma_{i-1})$ проверяет, если $\mathrm{H}(\mathrm{st}_i) = \sigma_{i-1}$, подпись проверяется в блоке и что $\mathrm{VerifyConsensus}(\mathrm{consensusState}_{i-1}, \mathrm{consensusProof}_i)$, где $\mathrm{consensusState}_{i-1}$ это часть σ_{i-1} а также π_i^B содержит $\mathrm{consensusProof}_i$.

- **Сводка блокчейна:** Сводка блокчейна состоит из состояния Σ и доказательство$_i$.
- $\mathrm{VerifyChainSummary}(\mathcal{S}_{i-1} = (\sigma_i, \mathrm{snark}_i))$: Как упоминалось ранее, этот алгоритм просто проверяет доказательство snark_i против утверждения σ_i.
- $\mathrm{VerifyBlock}(\mathcal{S}_{i_1}, B_i)$: Этот алгоритм просто проверяет согласованность, а именно проверку консенсуса, проверку подписи и проверку того, что состояние в $\mathcal{S}_{i_1}$ это хэш состояния в блоке.
- $\mathrm{UpdateChainSummary}(\mathcal{S}_{i-1}, B_i)$: Let $\mathcal{S}_{i-1} = (\sigma_{i-1}, \mathrm{snark}_{i-1})$. Этот алгоритм запускает функцию обновления как обновление (B_i, σ_{i-1}) чтобы получить σ_i. Затем проверяется доказательство snark_{i-1}. Наконец, он запускает доказательство, чтобы получить новое доказательство snark_i.

Рисунок 3: Протокол Мины.

Замечание 4.1. Мы предполагаем, что длина цепочки блоков не влияет на извлекаемость цепочки, потому что нет доказательств, указывающих на обратное для конструкций SNARK, которые мы используем, как также отмечено в [6].

5 Протокол Mina для платежей

В этом разделе мы сосредоточимся на платежном приложении, где каждая сторона имеет учетную запись с некоторым балансом, и транзакция перемещает часть своего баланса на счет другой стороны.

Далее мы сначала укажем структуру платежного приложения, затем базовую конструкцию SNARK, а затем представим протокол Mina для платежей.

5.0.1 Платежная система Mina

Пусть $\mathcal{U}$ — множество сторон. Структура состоит из следующих понятий.

- **Учетные записи**. Говорят, что каждая сторона имеет учетную запись, характеризуемую (pk, баланс, nonce), где pk является открытым ключом стороны для авторизации платежей, балансом ∈ N и одноразовым номером ∈ I, который предотвращает повтор транзакции.

- **Леджер**. Мы определяем бухгалтерскую книгу как список всех учетных записей. Мы будем ссылаться на учетную запись стороны как L(pk), а поля учетной записи, такие как баланс, — как L(pk).balance.

- **Сделка**. Транзакция — это перевод суммы со счета Отправителя L(pk*r*).balance на счет Получателя L(pk*r*).balance. Он представлен в виде txn = (pkB*B*, pk*r* , amt, nonce*B*, sig*B*), где pk*B*, pk*r* — открытые ключи Отправителя и Получателя соответственно, nonce*B* — одноразовый номер в учетной записи Отправителя, а sig*B* — подпись на (pk*r* , amt, nonce*B*) Отправителем.

  Проверка сделки. При наличии транзакции txn и реестра L следующий алгоритм проверяет действительность транзакции.

VerifyTransaction(txn = $(pk_B, pk_r, amt, nonce_B, sig_B)$, L):

assert $(L(pk_s).balance \geq amt)$;
assert $(L(pk_s).nonce = nonce_B)$;
assert $(\text{VerifySig}(pk_s, (pk_r, amt, nonce_s), sig_s) = T)$;

Вращение T

• **Обновление реестра**. Учитывая регистр L и набор транзакций {txn = (pk*r* ,pk*r* ,amt*r* ,nonce,sig*r*)}, следующий алгоритм обрабатывает набор, обновляя регистр.

UpdateLedger(L, {txn}):

∀txn ∈ {txn}
- $L(pk_B).balance \leftarrow L(pk_B).balance - amt$;
- $L(pk_r).balance \leftarrow L(pk_r).balance + amt$;
- $L(pk_B).nonce \leftarrow L(pk_B).nonce + 1$;

Вращение L

5.0.2 SNARK

Теперь мы опишем инкрементально вычисляемый SNARK S, используемый для построения нашего краткого блокчейна. Напомним, что система переходов между состояниями характеризует последовательно вычисляемый SNARK (см. определение 4.2). S указано на рисунке 4.

SNARK $\mathcal{S}$

Пусть H будет устойчивой к коллизиям хеш-функцией. $\mathcal{S}$ определяется как последовательно вычисляемый SNARK для следующей системы перехода состояний $(\Sigma, \mathrm{T}, \mathrm{Update})$:

- Σ представляет собой набор $\{\sigma\}_i$ пар хэш-значений реестра и состояний консенсуса $\{(\mathrm{ledgerHash}_i, \mathrm{consensusState}_i)\}$.
- T – набор блоков B_i имеет вид $(i, \mathsf{L}_{i-1}, \mathrm{consensusProof}_i, \{\mathrm{txn}_j\}_j, \mathsf{b\text{-}pk}_i, b\text{-}\mathrm{sig}_i)$, $\{\mathrm{txn}_j\}_j$ – набор транзакций, a b-pk – открытый ключ для проверки подписи $b\text{-}\mathrm{sig}_i$.
- Функция Update $(t_i, \sigma_{i-1}) \rightarrow \sigma_i$ определяется следующим образом:

$\underline{\text{Update}(t_i, \sigma_{i-1}) \rightarrow \sigma_i:}$

assert $(\text{VerifyTransaction}(\text{txn}_j, \mathsf{L}_{i-1}) = \top),\ \forall \text{txn}_j;$
assert $(\text{ledgerHash}_{i-1} = \mathsf{H}(\mathsf{L}_{i-1}));$
assert $(\text{VerifyConsensus}(\text{consensusState}_{i-1},$
$\text{consensusProof}_i) = \top);$
assert $(\text{VerifySig}(\text{b-pk}_i, m, b\text{-sig}_i) = \top),$ where $m =$
$(i, \mathsf{L}_{i-1}, \text{consensusProof}_i, \{\text{txn}_j\}_j);$
$\mathsf{L}_i \leftarrow \text{UpdateLedger}(\mathsf{L}_{i-1}, \{\text{txn}_j\}_j);$
$\text{ledgerHash}_i \leftarrow \mathsf{H}(\mathsf{L}_i);$
$\text{consensusState}_i \leftarrow \text{UpdateConsensus}($
$\text{consensusState}_{i-1}, \text{consensusProof}_i);$
return $\sigma_i \leftarrow (\text{ledgerHash}_i, \text{consensusState}_i)$

Рисунок 4: Базовые SNARKS.

5.0.3 Наша конструкция

Пусть (VerifyConsensus,UpdateConsensus) будет механизмом консенсуса Ouroboros Samasika. Пусть H будет устойчивой к коллизиям хэш-функцией. Протокол Mina определяется следующим образом.

Протокол Mina для платежей

Протокол Mina для платежей, определенный в структуре из Раздела 5.0.1,основан наSNARK на рисунке 4. Он состоит из следующих компонентов.

- **Блокчейн C_{i-1}.** Блокчейн состоит из хэштекущей книги, текущего состояния консенсуса и доказательства SNARK. То есть,

$$C_{i-1} = (\text{ledgerHash}_{i-1}, \text{consensusState}_{i-1}, \text{snark}_{i-1})$$

.

- **Блок B_i.** Напомним, что основная блокировка имеет формы $B_i = (\text{sn}_i, \text{st}_i, \pi_i^B, d_i, \text{b-pk}_i, b\text{-sig}_i)$, где $\text{sn}_i, \text{b-pk}_i, b\text{-sig}_i$ серийный номер, открытый ключ и подпись отправителя блока $(\text{sn}_i, \text{st}_i, \pi_i^B, d_i)$ автором блока соответственно. $\text{st}_i, \pi_i^B, d_i$ указаны следующим образом.
  - $\text{st}_i = \mathsf{L}_{i-1}$;
  - $\pi_i^B = \text{consensusProof}_i$;
  - $d_i = \{\text{txn}_j\}_j$.

Алгоритмы VerifyBlock, UpdateChain, VerifyChain определяются следующим образом.

- $\underline{\text{VerifyBlock}(\mathcal{S}_{i-1}, B_i) \rightarrow \top/\bot:}$

$$
\begin{aligned}
&\text{assert } (\mathsf{H}(\mathsf{L}_{i-1}) = \text{ledgerHash}_{i-1});\\
&\text{assert } (\text{VerifyConsensus}(\text{consensusState}_{i-1},\\
&\quad \text{consensusProof}_i) = \top);\\
&\forall j \text{assert } (\text{VerifyTransaction}(\text{txn}_j, \mathsf{L}_{i-1}) = \top);\\
&\text{assert } (\text{VerifySig}(\mathsf{b\text{-}pk}_i, m, \text{sig}_i) = \top),\\
&\quad \text{where } m = (\text{sn}_i, \mathsf{L}_{i-1}, \text{consensusProof}_i, \{\text{txn}_j\}_j)\\
&\text{return } \top
\end{aligned}
$$

- $\underline{\text{UpdateChainSummary}(\mathcal{S}_{i-1}, B_i) \rightarrow C_i:}$

$$
\begin{aligned}
&\text{assign } \sigma_{i-1} \leftarrow (\text{ledgerHash}_{i-1}, \text{consensusState}_{i-1});\\
&\sigma_i \leftarrow \text{Update}(B_i, \sigma_{i-1});\\
&\text{snark}_i \leftarrow \mathsf{sProve}(\sigma_i, (\sigma_{i-1}, t_i));\\
&\text{assign } \mathcal{S}_i \leftarrow (\sigma_i, \text{snark}_i);\\
&\text{return } \mathcal{S}_i
\end{aligned}
$$

- $\underline{\text{VerifyChainSummary}(\mathcal{S}_i) \rightarrow \top/\bot:}$

$$
\begin{aligned}
&\text{assign } \sigma_i \leftarrow (\text{ledgerHash}_i, \text{consensusState}_i);\\
&\text{return } \mathsf{sVerify}(\sigma_i, \text{snark}_i)
\end{aligned}
$$

Рисунок 5: Протокол Mina для платежей.

6 Snark работники

В этом разделе мы представим два метода оптимизации, а именно «состояние параллельного сканирования» и «стимулирование прувера». Они оба нацелены на следующую проблему.

Проблема. Обратите внимание на рис. 3, что для вычисления S_i нам нужно S_{i-1}. Таким образом, существует последовательная зависимость для вычисления доказательства SNARK. В результате наивная реализация страдает от времени блока, которое по крайней мере равно времени, необходимому для вычисления доказательства. Кроме того, он страдает от высоких требований к памяти для разработчиков блоков из-за высокой задержки транзакций (где задержка транзакции — это время, необходимое для суммирования транзакции в доказательстве SNARK).

Решение. Цель состоит в том, чтобы разработать методы, которые максимизируют пропускную способность. В частности, наша цель — максимально увеличить скорость обработки и проверки транзакций в сети протокола Mina. Это позволяет большему количеству одновременных пользователей в сети.

6.1 Состояние параллельного сканирования

Напомним, что необработанная цепочка блоков по своей сути является последовательной (т. е. вообще не может быть распараллелена). Однако, благодаря инкрементной вычислимости SNARK, работу SNARK можно распараллелить. Это ключевое наблюдение, которое приводит к понятию «состояние параллельного сканирования», когда мы отделяем создание блока от вычисления доказательств SNARK.

Мы поддерживаем специальную очередь, называемую рабочей очередью, куда мы ставим в очередь новые блоки по мере их поступления. Другими словами, это очередь «работы SNARK», которую должна выполнить сеть.

Затем сеть параллельно вычисляет доказательства SNARK; вычисляется дерево доказательств, где листья соответствуют доказательствам, доказывающим правильность отдельных блоков, а другие доказательства просто подтверждают правильность их дочерних доказательств. Наконец, доказательство корней свидетельствует о правильности всех блоков, соответствующих листьям дерева. Рассмотрим, например, последовательность блоков B_i, B_{i+1}, . . . , B_j в очереди на работу. Корневое доказательство подтверждает достоверность каждого из блоков. Это корневое доказательство может быть объединено с SNARK-доказательством достоверности S_{i-1}, чтобы получить SNARK-доказательство, подтверждающее достоверность S_j. Это показано на рис. 6. Обратите внимание, что это дерево внутри работает аналогично системе переходов, описанной в разделе 4.1.2.

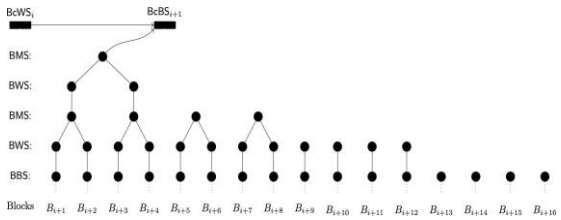


Рисунок 6: Моментальный снимок состояния параллельного сканирования.

Обратите внимание, что благодаря тщательной разработке параллелизма мы можем гарантировать, что пропускная способность полностью соответствует скорости добавления транзакций, что является оптимальным. В то время как задержка транзакции в простом подходе была O (*R*), где *R* — скорость, с которой создаются блоки, в предлагаемом подходе задержка транзакции равна O (log(*R*)). При тщательном проектировании структуры данных требования к памяти могут быть снижены с O (*R*) в наивном подходе до 2*R* − 1 + O (1) в предлагаемом подходе; мы опускаем детали структуры данных [?].

6.2 Стимулирование испытателей

Сторона, которая генерирует доказательства SNARK, называется *Snarker*. Мы опишем стимулы для проверки с целью достижения минимально возможной задержки транзакций.

(т. е. *минимизировать* временной разрыв между созданием блока и его поглощением в SNARK-доказательстве блокчейна).

Предлагаемая структура стимулирования выглядит следующим образом. Каждый производитель блоков, который помещает блок в рабочую очередь, должен извлекать блок, создавая доказательство, подтверждающее блок. Он отправляет запрос на комиссию вместе с генерируемым доказательством SNARK. Он также включает в себя транзакцию в том же блоке, которая платит комиссию доказывающему, который будет вычислять snark для этого блока. Как правило, комиссия выплачивается из комиссии за транзакцию, которую в противном случае получил бы производитель блока.

По сути, для каждой работы SNARK проводится аукцион по самой низкой цене. Производители блоков хотели бы платить snarkers как можно меньше за их доказательства, а snarkers хотели бы получать как можно более высокую плату за свои доказательства. Таким образом, принудительное выполнение производителем блоков функции доказывающего, но для другого блока обеспечивает стабильность системы.

Обратите внимание, что это понятие похоже на вышеупомянутый наивный подход в том, что производитель блоков также вычисляет доказательство SNARK, но с той разницей, что производитель блоков вычисляет доказательство для блока в начале очереди.

Замечание 6.1. Учитывая доказательство и соответствующий запрос на комиссию, мы требуем, чтобы злоумышленник не мог взломать запрос на комиссию. В противном случае злоумышленник может выдать доказательство другой стороны за свое (путем замены открытого ключа) или изменить чей-то запрос на комиссию.

Подписи знаний являются криптографическим примитивом, который позволяет нам достичь именно этого, как упоминалось в разделе 3. В Mina мы используем конструкцию, основанную на извлекаемом с помощью моделирования SNARK Боуи-Габизона [8].

7. Ouroboros Samasika — PoS-консенсус для кратких блокчейнов

Одним из основных технических вкладов этой статьи является первый доказуемо безопасный протокол консенсуса Proof-of-Stake для краткого блокчейна. Существующие протоколы либо не являются адаптивно безопасными, либо полагаются на централизованную доверенную третью сторону для получения рекомендаций по контрольным точкам для узлов, пытающихся загрузиться, либо полагаются на произвольную информацию в истории протокола для выбора цепочки, что немедленно делает протокол несовместимым для кратких настроек [16]. , 12, 3, 11].

Мы создаем согласованный протокол, защищенный от адаптивного искажения и не требующий доверенной службы контрольных точек для начальной загрузки, адаптивно безопасный и, что наиболее важно, лаконичный. То есть требуется лишь краткая информация, чтобы отличить честные цепочки от нечестных, независимо от того, как далеко в истории они разветвились. Нашей отправной точкой является консенсусный протокол PoS Ouroboros Genesis [3] (иногда называемый Genesis в этой статье).

Ouroboros Genesis уже использует адаптивную защиту и предлагает ускорение от Genesis. Однако правила выбора цепочки требуют информации о произвольном расстоянии в истории цепочки; это естественно, так как злоумышленник может изменить произвольные аспекты блокчейна, чтобы создать форк

и время разветвление может быть изучено сторонами спустя долгое время после того, как оно произошло. По этой причине Ouroboros Genesis сам по себе непригоден для использования в сжатой обстановке. Задача состоит в том, чтобы каким-то образом создать сводку истории постоянного размера, которой достаточно для правильного выбора одной цепи из нескольких кандидатов.
В этом разделе мы представляем *Protocolboros*, в котором мы решаем вышеуказанную проблему. В частности, мы демонстрируем подход к краткому обобщению информации, необходимой для правильного арбитража цепочек с вилками дальнего действия.

7.1 Интуитивное описание

В этой части раздела мы сосредоточим особое внимание на ядре консенсусного протокола, а именно на правилах выбора цепочки. Весь протокол консенсуса подробно описан вместе с доказательствами безопасности в Разделе 7.3.

Мы начнем с того, что вспомним правила Ouroboros Genesis.

Цепные правила отбора Ouroboros Genesis. В Ouroboros Genesis есть два правила выбора цепочки. Один - когда вилка ближнего действия; в этом случае правило состоит в том, чтобы просто выбрать самую длинную цепочку. Другой - когда вилка дальняя; в этом случае нельзя просто выбрать самую длинную цепочку, поскольку противник мог со временем проводить различные атаки, чтобы, возможно, исказить распределение выбора лидера, и ему удалось создать более длинную цепочку. Таким образом, для длинных вилок правило состоит в том, чтобы ограничить сравнение цепочек несколькими слотами сразу после вилки.

Обратите внимание, что правило для длинных разветвлений требует информации в произвольное время в истории. Ясно, что кратко изложить информацию не так-то просто.

Теперь мы представляем правила выбора цепи
Protocolboros.

Правила выбора сети
Protocolboros. Подобно Ouroboros Genesis,
Protocolboros также имеет два правила консенсуса, применимые в зависимости от того, как далеко в истории произошел форк.

Правило вилки ближнего действия. Грубо говоря, это правило срабатывает всякий раз, когда разветвление происходит таким образом, что противник еще не может изменить распределение плотности блоков, и правило состоит в том, чтобы просто выбрать самую длинную цепочку. Хотя действие правила такое же, как и в Ouroboros Genesis, предикат для принятия решения о том, является ли данная вилка ближней или нет, немного отличается. Поскольку основной вклад вносит правило дальнего разветвления, мы отложим точное описание предиката до Приложения 7.7.

Правило дальней вилки. Это правило применяется к разветвлениям, произошедшим более k блоков назад. Прежде чем мы опишем само правило, следует интуиция. Во-первых, напомним, почему просто правило самой длинной цепочки может не работать.

выбрать правильную цепочку в этом случае; после того, как противник создаст разветвление, со временем это может исказить распределение выбора лидера, что приведет к более длинной цепочке противников. Из-за этого мы можем полагаться только на разницу в плотности в первых нескольких слотах после форка (что и используется в Ouroboros Genesis). *Задача состоит в том, чтобы каким-то образом «перенести вперед» сводную информацию об этой разнице в плотности, даже если положение вилки — и, следовательно, рассматриваемый диапазон слотов — заранее неизвестно.*

Идея. Идея состоит в том, чтобы рассмотреть движущееся окно слотов и сохранить только *минимум* всех плотностей, наблюдаемых до сих пор в этом окне. Обратите внимание, что эта идея почти решает проблему: для нечестной цепочки, даже если противнику удастся увеличить плотность цепи, минимальное значение плотности указывает на окно, следующее за разветвлением, предоставляя требуемую сводку. С другой стороны, в честной цепочке нет больших колебаний в плотности, и из-за того, что в цепочке мажоритарная доля, минимальное значение плотности в цепочке, вероятно, будет выше, чем в нечестной цепочке.

Хотя это почти полностью решает проблему, есть еще одна часть проблемы, которую нам еще предстоит решить. А именно, какой длины должно быть окно и как оно должно скользить. Это очень важно, так как на него распространяются два противоречащих друг другу требования: (1) максимальная длина окна, поскольку после определенной точки после разветвления нельзя дать никаких гарантий относительно плотности блоков в состязательной цепочке и (2) Минимальная длина окна, поскольку требуется некоторое минимальное количество выборок, чтобы отличить данные два распределения. Окно критической длины после разветвления, удовлетворяющее обоим ограничениям, назовем «*критическим окном*». Таким образом, задача состоит в том, чтобы спроектировать длину окна и его движение таким образом, чтобы независимо от того, где расположена вилка, движущееся окно захватывало все критическое окно одним кадром. Идея состоит в том, чтобы длина окна была немного больше критической длины окна. Что касается перемещения окна, во-первых, обратите внимание, что простое смещающееся окно (где следующее смещение перемещает начало окна после его текущего конца) может не захватить все критическое окно в любой заданной позиции, поскольку форк может происходить в произвольных слотах. Идея состоит в том, чтобы сдвинуть окно на долю своего размера. Получившееся окно назовем «ν -сдвигающим ω-окном», где ω — длина окна, а ν — длина, на которую оно смещается (см. определение 1). При тщательной калибровке ν и ω мы можем гарантировать, что окно захватывает критическое окно.

Эта интуиция формально доказывается далее в теореме 2.

Ниже мы формально опишем правила выбора цепи. Наше новое правило выбора цепочки, формально обозначенное как алгоритм **maxvalid-sc(·)** (см. рис. 7), хирургически адаптирует правило выбора цепочки из Ouroboros Genesis, а именно **maxvalid-bg(·)** (см. рис. 20), но путем замены длинного правило вилки диапазона с новым. Мы продолжим все обсуждения, ссылаясь на

базовые блокчейны цепочек резюме. Отметим, что единственная информация о базовых блокчейнах, необходимая в любой части механизма консенсуса, — это фиксированное количество блоков в непосредственной истории. Таким образом, проверка консенсуса является краткой.

Здесь C_{loc} — локальная цепь, $\boldsymbol{\mathcal{N}} = \{C_1, \ldots, C_M\}$ — список цепочек на выбор. Функция isShortRange(C, C') выводит независимо от того, разветвляются ли цепочки в «ближнем диапазоне» или нет. Функция **getMinDen(C**) выводит минимум всех плотностей окон, наблюдаемых до сих пор в C; формально он определен на рисунке 16.

Алгоритм maxvalid-sc$(C_{\text{loc}}, \mathcal{N} = \{C_1, \ldots, C_M\}, k)$

// Сравнивать C_{loc} с каждой цепью-кандидатом в $\mathcal{N}$

1. Настроить $C_{\text{max}} \leftarrow C_{\text{loc}}$
2. **для** $i = 1, \ldots, M$ **делать**
   если isShortRange(C_i, C_{max}) **тогда** // Вилка ближнего действия
      если $|C_i| > |C_{\text{max}}|$ **тогда**
      Настроить $C_{\text{max}} \leftarrow C_i$
      и если
      еще // Вилка дальнего действия
      если getMinDen$(C_i) >$ getMinDen(C_{max}) **тогда**
      Настроить $C_{\text{max}} \leftarrow C_i$
      и если
      и если
   и для
3. **возврат** C_{max}

Рисунок 7: Новое правило выбора цепочки.

7.2 Предыстория

Мы вводим множество важных концепций из Ouroboros Genesis, которые остаются неизменными и в нашем сеттинге. Начнем с краткого обзора семейства протоколов Ouroboros.

Семейство консенсусных протоколов Ouroboros. Kiayias и др. предложил первый протокол Proof-of-Stake (PoS) **Ouroboros** [16] со строгой безопасностью

гарантии. Однако состязательная модель рассматривала только синхронные сети. Здесь выполнение протокола разделено на временные единицы, называемые слотами, а группы *слотов* образуют *эпохи*.

Состязательная модель была усилена в последующей работе **Ouroboros Praos** [12] за счет рассмотрения полусинхронной настройки (где выполнение протокола по-прежнему делится на слоты и эпохи, но сеть может испытывать максимальную задержку Δ слотов при доставке сообщений). .) В этой работе введено понятие «*пустых слотов*» как искусственный способ предоставления коротких периодов тишины для сторон, чтобы наверстать упущенное в случаях задержек доставки сообщений. Важным моментом, который следует отметить в отношении Ouroboros Praos, является правило выбора цепи. Всякий раз, когда есть две цепочки, которые разветвились «недавно» (т. е. когда есть так называемая «*короткая развилка*»), она выбирает более длинную цепочку. С другой стороны, т. е. если разветвление происходит далеко в прошлом (или когда существует так называемое «*долговременное разветвление*»), то он просто полагается на доверенную внешнюю службу (называемую службой контрольных точек) для предоставления совет по честной цепи. Ясно, что это вносит сильный элемент централизации, который рассматривается в последующем проекте.

Наконец, **Ouroboros Genesis** преодолел недостаток Ouroboros Praos, заключающийся в том, что он полагался на внешний сервис для разрешения вилок дальнего действия, предоставив дополнительное правило выбора цепочки [3]. В частности, правило рассматривает короткий диапазон s слотов вскоре после дальнего разветвления и выбирает цепь с более высокой плотностью в этих слотах.

Время моделирования: слоты, эпохи и пустые слоты. Время выполнения протокола делится на эпохи, которые далее делятся на слоты. В эпохе есть слоты R. Как и в Ouroboros Praos и Ouroboros Genesis, некоторые слоты могут быть «*пустыми*» (то есть без какого-либо связанного с ними блока). В частности, параметр консенсуса, f, обозначает вероятность того, что для любого данного слота назначен производитель блоков. Если для слота не назначен ни один производитель блоков, то слот пуст (т. е. без блока).

Виды сторон. Мы классифицируем стороны таким образом, чтобы моделировать различные сценарии реальной жизни, такие как новые присоединяющиеся стороны и стороны с временными проблемами подключения/доступности, как подробно описано в [3]. Модель определяет три вида партий следующим образом. *Стороны оповещения* — это стороны, которые имеют доступ ко всем требуемым ресурсам, а также синхронизируются. Эти стороны пользуются полными гарантиями безопасности, и нам потребуется нижняя граница их доли (см. ниже) для обеспечения безопасности. *Потенциально активные стороны* (или для краткости активные) — это все стороны, которые, вообще говоря, потенциально могут действовать в текущем временном интервале выполнения протокола. Сюда входят как честные стороны, имеющие доступ ко всем необходимым ресурсам, так и враждебные стороны. *Неактивные стороны* — это все другие стороны, такие как честные стороны, которые не могут получить доступ к некоторым необходимым ресурсам для взаимодействия с протоколом, например, к их сетевому соединению.

Распределение ставок. Предполагается, что протокол поддерживает следующие соотношения:

- $\alpha \geq 1/2$ - это отношение ставки оповещения к активной ставке.

- β - отношение активной ставки ко всей ставке.

Распределения случайности эпох и выбора лидеров. Распределение ставок, рассматриваемое в эпоху **ep**, на самом деле является распределением ставок в последнем слоте **ep**-2. Помимо распределения ставок, понятие случайности эпохи влияет на распределение выбора лидера в любую эпоху. В частности, случайность эпохи выводится как функция информации об определенных блоках из первых двух третей ep −1.

Важным замечанием, которое мы будем использовать в наших доказательствах, является следующий факт: рассмотрим разветвление в точке $sl*$, где нечестная цепь разветвляется от честной цепи. Для слотов $R/3$, следующих за $sl*$, распределения случайности эпохи и выбора лидера гарантированно не будут искажены. Однако в слотах, следующих за слотами $R/3$, такая гарантия не может быть дана в отношении распределения в нечестной цепочке.

Обозначения. Для цепи C и интервала слотов $I \triangleq a\ [sl_i, sl_j] = \{sl_i, \ldots, sl_j\}$, обозначим через $C[I] = C[sl_i, sl_j]$ последовательность блоков в C, номера слотов которых попадают в интервал . Мы заменяем скобки в этих обозначениях скобками для обозначения интервалов, не включающих конечные точки; например, $sl_i, sl_j] = \{sl_{i+1}, \ldots, sl_j\}$. Наконец, мы обозначаем через $|C[I]|$ количество блоков в $C[I]$. Обычно мы обозначаем партию ***P***.

7.3 Новые правила выбора сети

В этом разделе мы опишем предлагаемое правило выбора цепочки для краткой настройки. Описание структурировано таким образом, чтобы четко выделить (синим цветом) отличия от соответствующих протоколов и алгоритмов Ouroboros Genesis. Алгоритмы/протоколы, совершенно новые для Ouroboros Samasika, выделены только своими именами; описание не выделено для удобства чтения. Более того, в описаниях протоколов упоминаются детали, общие для Ouroboros Samasika и Ouroboros Genesis, но не углубляются в них, чтобы (а) сосредоточиться на основном вкладе и (б) поддерживать уровень описания псевдокода. Мы отсылаем читателя к [3] за деталями, общими для Ouroboros Samasika и Ouroboros Genesis. Формальный алгоритм выбора цепи **maxvalid-sc** представлен на рис. 7.

7.3.1 Правило выбора цепи ближнего действия

Напомним, что форк является кратким, когда можно гарантировать, что противник еще не изменил распределение плотности блоков. **maxvalid-sc** вызывает предикат **isShortRange**, который выводит, является ли заданный диапазон ближним или нет в этом смысле. Предикат описывается следующим образом.

Алгоритм isShortRange(C_1, C_2)

1. Позволить $\text{prevLock}_1^{\text{cp}}$ а также $\text{prevLock}_2^{\text{cp}}$ быть $\text{prevLock}^{\text{cp}}$ компоненты в последних блоках C_1, C_2, соответственно.
2. **если** $\text{prevLock}_1^{\text{cp}} = \text{prevLock}_2^{\text{cp}}$ **тогда**
3. **возврат** $\top$
4. **еще**
5. **возврат** $\bot$

Рисунок 8: Алгоритм определения того, является ли данный форк ближним или нет.

Ниже на рисунке 12 показан протокол, выполняемый для выбора новой цепочки, обозначенный как SelectChain.

Протокол SelectChain$(P, \text{sid}, C_{\text{loc}}, \mathcal{N} = \{C_1, \ldots, C_M\}, k)$

// Шаг 1: Отменить недопустимые цепочки

1. Инициализировать $\mathcal{N}_{\text{valid}} \leftarrow \emptyset$
2. **для** $i = 1, \ldots, M$ **делать**
   Вызов протокола IsValidChain$(P, \text{sid}, C_{\text{loc}}, C_i, k)$; если вернется $\top$ затем обновлять $\mathcal{N}_{\text{valid}} \leftarrow \mathcal{N}_{\text{valid}} \cup \{C_i\}$
   конец для

// Шаг 2: Применить правило выбора цепочки к действительным цепочкам

3. Выполнить алгоритм maxvalid-sc$(C_{\text{loc}}, \mathcal{N}_{\text{valid}}, k)$. Обозначить выход цепь по C_{max}.
4. Набор $C_{\text{loc}} \leftarrow C_{\text{max}}$. Выход C_{loc}.

ВЫХОД: Выход C_{loc}.

Рисунок 9: Протокол для сторон, чтобы выбрать цепочку, когда их больше одной.

7.4 Минимальная плотность окна

Основной концепцией нового правила выбора цепочки является понятие плотности окна. Как упоминалось ранее в этом разделе, идея состоит в том, чтобы рассмотреть смещение

окна и всегда поддерживать минимальную плотность во всех окнах до сих пор.

Более формально мы используем ν сдвигающееся окно (см. определение 1), в котором длинное окно смещается за раз на ν слоты.

Определение 1 (u-сдвигающее ω-окно). Для ν, ω N и $0 < \nu < \omega$ ν -сдвигающее ω-окно над последовательностью слотов sl_1, sl_2, . . . характеризуется алгоритмом shiftWindow, который принимает на вход переменную (или набор переменных), являющуюся функцией слотов в интервале $[sl_{i+1}, sl_{i+\omega}]$ и присваивает/обновляет ее функцией слотов $[sl_{i+1+\nu}, sl_{i+\omega+\nu}]$.. Мы называем u параметром сдвига, а ω — параметром длины окна.

Параметр сдвига ν и параметр длины окна ω устанавливаются следующим образом. Пусть s_{CG} — параметр роста цепи, обеспечиваемый правилом короткодействующей цепной селекции (эквивалентным цепному правилу селекции Ouroboros Genesis) (см. теорему 2 в [3]).
Отправной точкой для Ouroboros Samasika является Ouroboros Genesis. В Ouroboros Genesis рассматривалось (неизменяющееся) окно размером порядка s_{CG} сразу после форка. В Ouroboros Samasika из-за краткости мы считаем позиционирование окна независимым от положения вилки. Однако нам нужно рассмотреть окно почти близко к развилке. По этой причине мы рассматриваем окно размером немного больше, чем s_{CG} (см. (1)) и слегка сдвигаем его на ν слотов с течением времени (см. (2)), чтобы захватить окно, близкое к разветвлению. В целях реализации можно представить себе n_s подокон, каждое из которых имеет длину ν слотов, составляющих окно (см. (3)).

$$\omega = (1 + s_B)s_{CG}, \quad (1)$$

$$u = s_B\, s_{CG}, \quad (2)$$

$$n_B = (1/s_B) + 1, \quad (3)$$

при этом гарантируя, что ν и $1/\epsilon_s$ являются целыми числами и что $\epsilon_s > 0$. Для наглядности приведем пример: пусть s_{CG} = 6 слотов. Пусть ϵ_s = 1/3. Тогда окно имеет размер ω = 8 интервалов со сдвигом на ν = 2 интервала (см. рис. 10).

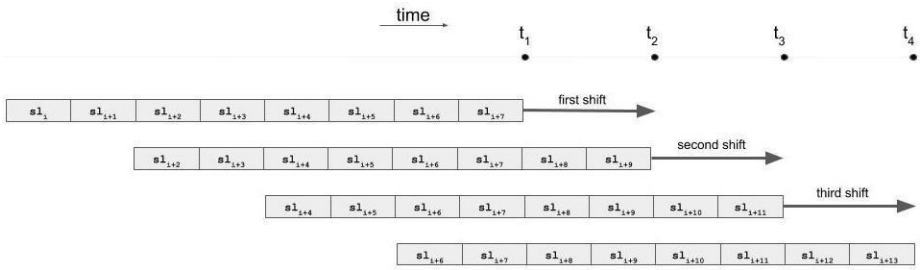


Рисунок 10: Пример окна сдвига с s_{CG} = 6 слотов и ϵ_s = 1/3. Следовательно, размер окна ω = 8 слотов и размер сдвига ν = 2 слота. Первый сдвиг происходит, когда текущее время равно t_2, второй — t_3 и так далее.

Мы представляем пример алгоритма реализации на рисунках 11 и 13. Мы обозначаем минимальную плотность окна через **minDen**. Как упоминалось ранее, мы поддерживаем n_s подокон, каждое из которых имеет размер ν слотов; а именно, $\text{pDen}_1, \ldots, \text{pDen}_{ns}$. Мы также поддерживаем дополнительное подокно $\text{pDen}_{\text{curr}}$ также размером ν слотов, чтобы отслеживать плотность в текущем подокне до того, как произойдет сдвиг. Последовательность всех эти параметры обозначаются $\overrightarrow{\text{Den}} = (\text{pDen}_1, \ldots, \text{pDen}_{ns}, \text{pDen}_{\text{curr}}, \text{minDen})$.

Алгоритм isWindowStop(sl, ν)

1. **если** $(\text{sl} \ \% \ \nu) = 0$
2. **возвращаться** $\top$
3. **иначе**
4. **возвращаться** $\bot$

Рисунок 11: Алгоритм проверки достижения конца окна.

Протокол SelectChain$(P, \text{sid}, C_{\text{loc}}, \mathcal{N} = \{C_1, \ldots, C_M\}, k)$

// Шаг 1: Отменить недопустимые цепочки

1. Инициализировать $\mathcal{N}_{\text{valid}} \leftarrow \emptyset$
2. **для** $i = 1, \ldots, M$ **делать**

   Вызов протокола IsValidChain$(P, \text{sid}, C_{\text{loc}}, C_i, k)$; если вернется $\top$ затем обновлять $\mathcal{N}_{\text{valid}} \leftarrow \mathcal{N}_{\text{valid}} \cup \{C_i\}$

   конец для

// Шаг 2: Применить правило выбора цепочки к действительным цепочкам

3. Выполнить алгоритм maxvalid-sc$(C_{\text{loc}}, \mathcal{N}_{\text{valid}}, k)$. Обозначить выход цепь по C_{max}.
4. Набор $C_{\text{loc}} \leftarrow C_{\text{max}}$. Выход C_{loc}.

ВЫХОД: Выход C_{loc}.

Рисунок 12: Протокол, который выбирает цепочку с помощью **maxvalid-sc**.

Алгоритм shiftWindow($\overrightarrow{\text{Den}}$)

Позволять $\overrightarrow{\text{Den}} = (\text{pDen}_1, \ldots, \text{pDen}_{n_s}, \text{pDen}_{\text{curr}}, \text{minDen})$.

1. набор minDen $\leftarrow \min(\text{minDen}, \text{minDen} - \text{pDen}_1 + \text{pDen}_{\text{curr}})$
2. для $i = 1$ to $n_s - 1$
3. набор $\text{pDen}_i \leftarrow \text{pDen}_{i+1}$
4. **конец для**
5. набор $\text{pDen}_{n_s} \leftarrow \text{pDen}_{\text{curr}}$ and $\text{pDen}_{\text{curr}} \leftarrow 0$
6. **возврат**$(\text{pDen}_1, \ldots, \text{pDen}_{n_s}, \text{pDen}_{\text{curr}}, \text{minDen})$

Рисунок 13: Алгоритм смещения окна.

Замечание 7.1 (О делимости длины окна на сдвиг). Требование длины сдвига $\mathcal{V}$, полностью делящей длину окна ω, состоит только в том, чтобы иметь чистое описание алгоритма. Все аргументы в силе, даже если это не так.

7.5 Принятие нового правила выбора цепочки

Теперь мы обсудим, как модификации, введенные в алгоритм maxvalid-sc из Ouroboros Genesis, проникают в другие подпротоколы консенсуса.

Сначала устанавливаются параметры $\overrightarrow{\text{Den}} = (\text{pDen}_1, \ldots, \text{pDen}_{ns}, \text{pDen}_{\text{curr}}, \text{minDen})$.
когда блок Genesis генерируется путем запуска прототипа Initialization-Genesis.
кол. Сторона, которая была зарегистрирована со всеми своими ресурсами, становится работоспособной, вызывая этот протокол.

Алгоритм Инициализации-Генезис $(P, \text{sid}, R, n_s, \omega)$

1. Отправлять $(\text{KeyGen}, \text{sid}, P)$ к $\mathcal{F}_{\text{VRF}}$ and $\mathcal{F}_{\text{KES}}$; получение $(\text{VerificationKey}, \text{sid}, v_p^{\text{vrf}})$ и $(\text{VerificationKey}, \text{sid}, v_p^{\text{kes}})$, соответственно
2. **если** $\tau = 0$ **тогда**
3. Отправлять $(\text{ver\_keys}, \text{sid}, P, v_p^{\text{vrf}}, v_p^{\text{kes}})$ к $\mathcal{F}_{\text{INIT}}$ претензии берутся из генезисблока.
4. Призвать $\text{FinishRound}(P)$ и вызывать $\text{UpdateTime}(P)$ обновить τ, ep, sl.
5. **тогда как** $\tau = 0$ **выполнять**
6. Вызов $\text{UpdateTime}(P)$ апдейт, и отказ от активации
7. **конец, пока**
8. **конец, если** // Следующее выполняется, если это раунд on-genesis.
9. **если** $\tau > 0$ **тогда**
10. Установите каждую из переменных $\{\text{pDen}_1, \ldots, \text{pDen}_{n_s}, \text{pDen}_{\text{curr}}\}$ to $\varnothing$. Также набор $\text{minDen} \leftarrow \omega$.
11. набор $\overrightarrow{\text{Den}} = (\text{pDen}_1, \ldots, \text{pDen}_{n_s}, \text{pDen}_{\text{curr}}, \text{minDen})$.
12. если $\mathcal{F}_{\text{INIT}}$ сигнализирует об ошибке тогда
13. Остановить казнь.
14. **конец, если**
15. Отправлять $(\text{genblock\_req}, \text{sid}, P)$ к $\mathcal{F}_{\text{INIT}}$.
16. Получить от $\mathcal{F}_{\text{INIT}}$ ответ $(\text{genblock}, \text{sid}, \mathbf{G}' = (\mathbb{S}_1, \eta_1, \overrightarrow{\text{Den}}))$, где

$$\mathbb{S}_1 = ((U_1, v_1^{\text{vrf}}, v_1^{\text{kes}}, s_1), \ldots, (U_n, v_n^{\text{vrf}}, v_n^{\text{kes}}, s_n)).$$

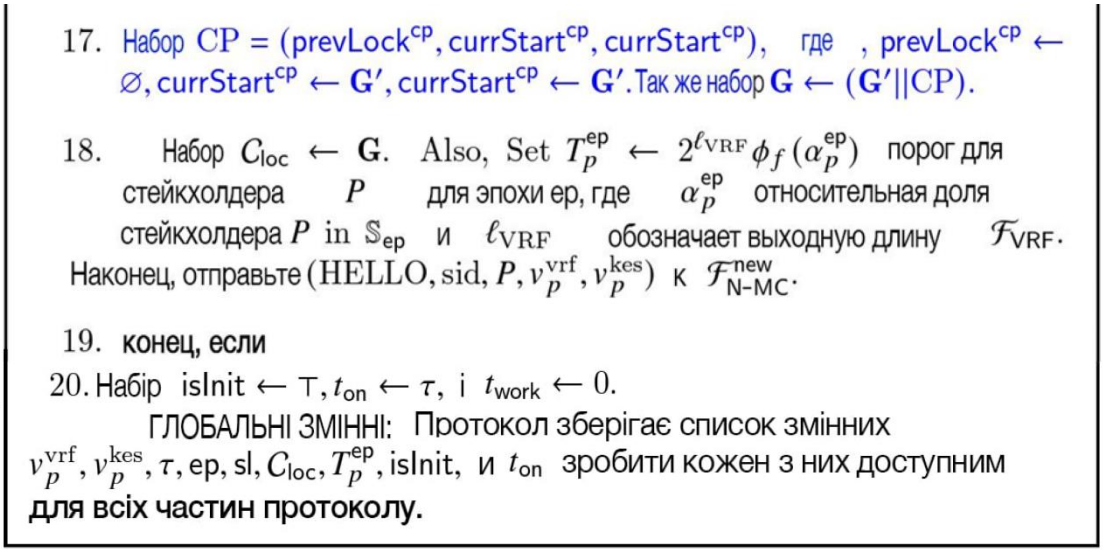


Рисунок 14: Протокол инициализации Ouroboros Samasika (запускается только при первом присоединении группы).

ПРОТОКОЛ StakingProcedure(P, sid, k, ep, sl, buffer, C_{loc})

Следующие шаги выполняются в (MAINTAIN-LEDGER, sid, minerID)-прерываемом режиме:

1. Отправлять (EvalProve, sid, nonce_j||sl||NONCE) к $\mathcal{F}_{\text{VRF}}$, обозначают ответ от $\mathcal{F}_{\text{VRF}}$ по (Evaluated, sid, y_ρ, π_ρ).Кроме того, отправить(EvalProve, sid, nonce_j ||sl||TEST)) к $\mathcal{F}_{\text{VRF}}$, обозначают ответ от $\mathcal{F}_{\text{VRF}}$ по (Evaluated, sid, y, π).

2. если $y < T_p^{\text{ep}}$ тогда // Генерируем новый блок

3. Набор buffer′ ← buffer, $\vec{N} \leftarrow \text{txn}p^{\text{base-tx}}$ and st ← $\text{blockify}_{\text{OG}}(\vec{N})$

4. Возврат

5. Анализировать buffer как последовательность $(\text{txn}1, \ldots, \text{txn}n)$

6. для $i = 1$ к n выполнять

7. если $\text{ValidTx}_{\text{OG}}(\text{txn}i, \vec{\text{st}}||\text{st}) = 1$ тогда

8. Набор $\vec{N} \leftarrow \vec{N}||\text{txn}i$, удалить $\text{txn}i$ от buffer′ и ɴналаш-тувати $\text{st} \leftarrow \text{blockify}_{\text{OG}}(\vec{N})$

9. и если

10. и тогда

11. до того как $\vec{N}$ больше не увеличивается

12. Настроить $\text{crt} = (P, y, \pi)$, $\rho = (y_\rho, \pi_\rho)$ и $h \leftarrow \text{H}(\text{head}(C_{\text{loc}}))$ и отправить (USign, sid, P, $(h, \text{st}, \text{sl}, \text{crt}, \rho)$, sl) к $\mathcal{F}_{\text{KES}}$: обозначить ответ через by (Signature, sid, $(h, \text{st}, \text{sl}, \text{crt}, \rho)$, sl, σ).

13. Обновите переменные минимальной плотности следующим образом:

- Выполнять $\overrightarrow{\text{Den}} \leftarrow \text{get}\overrightarrow{\text{Den}}(C_{\text{loc}})$, where $\overrightarrow{\text{Den}} = (\text{pDen}_1, \ldots, \text{pDen}_{n_s}, \text{pDen}_{\text{curr}}, \text{minDen})$.
- $\text{pDen}_{\text{curr}} \leftarrow \text{pDen}_{\text{curr}} + 1$

- если $\mathsf{isWindowStop}(\mathsf{sl}, \nu)$ тогда
- $\overrightarrow{\mathsf{Den}} \leftarrow \mathsf{shiftWindow}(\overrightarrow{\mathsf{Den}})$
- и если

14. Настроить $B' \leftarrow (h, \mathrm{st}, \mathsf{sl}, \mathrm{crt}, \rho, \sigma, \overrightarrow{\mathsf{Den}})$
15. Обновите контрольные точки CP следующим образом:
16. если слот $\mathsf{sl} = 1$, i.e., первый в текущую эпоху, тонда
    - Настроить $\mathsf{currStart}^{\mathsf{cp}} \leftarrow B$
    - Настроить $\mathsf{prevLock}^{\mathsf{cp}} \leftarrow \mathsf{currStart}^{\mathsf{cp}}$
    - Настроить $\mathsf{currStart}^{\mathsf{cp}} \leftarrow B$
17. иначе, если $1 < \mathsf{sl} \leq \frac{2}{3} * R$ тогда
18. $\mathsf{currStart}^{\mathsf{cp}} \leftarrow B$
19. конец, если
20. Настроить $B \leftarrow (B' || \mathrm{CP})$ и обновить $C_{\mathsf{loc}} \leftarrow C_{\mathsf{loc}} || B$. Многоадресная рассылка расширенной цепочки и ожидание..
21. Отправить $(\mathrm{MULTICAST}, \mathrm{sid}, C_{\mathsf{loc}})$ к $\mathcal{F}_{\mathsf{N-MC}}^{\mathsf{bc}}$ и действуйте отсюда при следующей активации этой процедуры.
22. иначе
23. Усовершенствуйте ключ подписи KES, отправив $(\mathrm{USign}, \mathrm{sid}, P, 0, \mathsf{sl})$ к $\mathcal{F}_{\mathsf{KES}}$ и установите привязку в конце процедуры для возобновления работы при следующей активации технического обслуживания.
24. конец, если

Рисунок 15: Процедура стейкинга Ouroboros Samasika.

На рисунке 7.5 показано, где в архитектуре протокола поддерживать и обновлять параметры min-density $\overrightarrow{\mathsf{Den}}$ окна. В частности, у нас будет каждый блок содержит текущие значения этих параметров. Для любого блока параметры генерируются лидером слота. В частности, лидер слота начинает с получения $\overrightarrow{\mathsf{Den}}$ для последнего блока в цепочке, которую группа собирается расширить.Затем партия обновляет параметры (используя алгоритмы **shiftWindow**$(\cdot)$ и **isWindowStop**$(\cdot, \cdot)$).

АЛГОРИТМ $\mathrm{getMinDen}(C)$

Позволять B_{last} быть последним блоком в C.

1. если $B_{\text{last}} = \mathbf{G}$ тогда // т. е. если B_{last} блок генезиса
2. возврат 0
3. если
4. Разбор B_{last} для получения параметра minDen.
5. возврат minDen

Рисунок 16: Протокол для получения текущей минимальной плотности окна данной цепочки.

7.6 Доказательства безопасности

В этом разделе мы доказываем безопасность протокола Ouroboros Samasika. Начнем с некоторых предварительных сведений, которые будут полезны в доказательствах.

7.6.1 Предварительные

В различных частях доказательства нам нужно оценить ожидаемое количество блоков для заданной характеристической строки. Ниже мы определяем и разрабатываем различные инструменты, которые облегчат оценку.

Во многих рассуждениях используется неравенство Azuma (см. [17], раздел 4), сформулированное ниже.

Lemma 1 (неравенство Azuma (Azuma; Hoeffding)). Пусть $X_0, \ldots, X_n$ — последовательность вещественных случайных величин, такая, что для всех t, $| X_{t+1} - X_t | \leq c$ для некоторой константы c. Если $E[X_{t+1} | X_0, \ldots, X_t] \leq X_t$ для всех t, то для каждого ≥ 0

$$\Pr[X_n - X_0 \geq \Lambda] \leq \exp\left(-\frac{\Lambda^2}{2nc^2}\right).$$

В качестве альтернативы, if $\mathbb{E}[X_{t+1} \mid X_0, \ldots, X_t] \geq X_t$ для всех t затем для каждого $\Lambda \geq 0$

$$\Pr[X_n - X_0 \leq -\Lambda] \leq \exp\left(-\frac{\Lambda^2}{2nc^2}\right).$$

Еще одна граница большого отклонения, которую мы используем в наших вероятностных рассуждениях, — это граница Чернова, о которой упоминается ниже.

Теорема 1 (граница Чернова). Пусть $X_1, \ldots, X.n$ — независимая случайная вариация.

$\mathbb{E}[X_i] = p_i$ та $X_i \in [0, 1]$ Пусть $X = \sum_{i=1}^{n} X_i$ та $\mu = \sum_{i=1}^{n} p_i = \mathbb{E}[X]$.

Тогда для всех $\Lambda \geq 0,$

$$Pr[X \geq (1+\Lambda)\mu] \leq e^{-\frac{\Lambda^2}{2+\Lambda}\mu}$$
$$Pr[X \geq (1+\Lambda)\mu] \leq e^{-\frac{\Lambda^2}{2+\Lambda}\mu}$$

Определение 2 (Супербиномиальные мартингальные условия). Рассмотрим семейство случайных величин $X_1, \ldots, X_n$, принимающие значения в0, 1 ^{n}. Мы говорим, что они удовлетворяют γ-супербиномиальным мартингальным условиям (или просто γ-мартингальным условиям), если

$$\Pr[X_k = 0 \mid X_1, \ldots, X_{k-1}] \geq \gamma, \text{ и следовательно}$$
$$\Pr[X_k = 1 \mid X_1, \ldots, X_{k-1}] \leq 1 - \gamma.$$

Мы можем естественным образом применить ту же терминологию к бесконечным последовательностям переменных, принимающих значения в {0, 1}.

Следствие 1 (следствие неравенства Azuma (см. Lemma 7 в [3])). Пусть $X_1, \ldots, X_n$ удовлетворяют γ -супербиномиальным мартингальным условиям с $\gamma \geq 1/2$. Тогда для любого $\delta > 0$,

$$\Pr[\#_0(X) \leq (1-\delta)\gamma n] \leq \exp(-\delta^2 n/2)$$

и

$$\Pr[\#_1(X) \geq (1+\delta)(1-\gamma)n] \leq \exp(-\delta^2 n/2)$$

где, $\#_0(X) = |\{i \mid X_i = 0\}|$ and $\#_1(X) = |\{i \mid X_i = 1\}|$.

7.6.2 Доказательства

Как и в Ouroboros Praos и Ouroboros Genesis, большая часть анализа характеристических строк выполняется путем преобразования их в их синхронные версии и анализа последних. Полученное распределение называется «индуцированным распределением» и обозначается p_Δ (·). В лемме 6 Книги Бытия показано, что индуцированное распределение является префиксом распределения, удовлетворяющего ç (1− ƒ) Δ+1--мартингальным условиям (см. определение 3). Ниже мы докажем некоторые полезные свойства распределения, которые используются в основе нашего основного доказательства.

Определение 3 (Супербиномиальные мартингальные условия). Рассмотрим семейство случайных величин $X_1, \ldots, X_n$ принимающие значения в $\{0, 1\}^n$. Мы говорим, что они удовлетворяют ç-супербиномиальным мартингальным условиям (или просто γ-мартингальным условиям), если

$$\Pr[X_k = 0 \mid X_1, \ldots, X_{k-1}] \geq \gamma, \text{ и следовательно}$$
$$\Pr[X_k = 1 \mid X_1, \ldots, X_{k-1}] \leq 1 - \gamma.$$

Мы можем естественным образом применить ту же терминологию к бесконечным последовательностям переменных, принимающих значения в {0, 1}.

Lemma 2 (Структура индуцированного распределения). Пусть $W = W_1, \ldots, W_n$ последовательность случайных величин, каждая из которых принимает значения из {0, 1, ⊥}, которые удовлетворяют (f ; $ç$)-характеристическим условиям и пусть

$$X = X_1, \ldots, X_4 = p_\Delta (W_1, \ldots, W_n)$$

— синхронные эквивалентные случайные величины, полученные путем применения отображения Δ- редукции к W . Также пусть $\Pr[W_i = \perp \mid W_1, \ldots, W_{i-1}] \le (1 - a)$. If $ç\ (1 - f\ ) \Delta+1 \ge (1 + s\ )/2$ для некоторого s ≥ 0, то выполняется следующее.

1. Для всех $\delta_\ell, \delta_0 > 0,$

$$\epsilon_{\#_0}(a,n) \triangleq \Pr[\#_0(X) < (1-\delta_0)\frac{(1+\epsilon)}{2}((1-\delta_\ell)an) - \Delta] \le \exp\left(-\frac{\delta_\ell^2 a^2 n}{2(1-a)^2}\right) + \exp\left(-\frac{\delta_0^2((1-\delta_\ell)an)}{2}\right) \quad (4)$$

2. Для всех $\delta_1 > 0,$

$$\epsilon_{\#_1}(a,n) \triangleq \Pr[\#_1(X) > (1+\delta_1)\frac{(1-\epsilon)}{2}an - \Delta] \le= \exp\left(-\frac{\delta_1^2 an}{2}\right) \quad (5)$$

Доказательство. Мы устанавливаем границы, используя следствие 1. Напомним, что следствие 1 дает ограничения на количество единиц и нулей в последовательности *бинарных* случайных величин, удовлетворяющих условиям мартингала. Однако W может также содержать ⊥. Поэтому рассмотрим Δ-редуцированное отображение W , а именно $X = X_1, \ldots, X_4 = p_\Delta (W_1, \ldots, W_n)$ и применим лемму об отображенном распределении.

Из Lemma 8 (i) и (ii) книги «Ouroboros Genesis» мы имеем, что $X_1, \ldots, X_{\ell-\Delta}$ — префикс к последовательности случайных величин $Z_1, Z_2, \ldots$ которые удовлетворяют y (1− f) Δ+1-условиям мартингала (определение см. в Приложении 7.6.1). Поэтому мы можем применить Lemma 1 $X_1, \ldots, X_{\ell-\Delta}$.

Доказательство (i) построено следующим образом. Сначала мы установим нижнюю границу числа 4, применив неравенство Azuma. Затем для этой нижней оценки применим Lemma 1 на $X_1, \ldots, X_{4-\Delta}$, чтобы получить нижнюю границу числа нулей.

Рассмотрим случайные величины

$$A_i \triangleq \begin{cases} 0, & \text{if } W_i = \perp \\ 1, & \text{if } W_i \neq \perp \end{cases}$$

и пусть $\ell = \sum_{i=1}^{n} A_i$. $\text{Then, } \Pr[A_i = 1 \mid A_1, \ldots, A_{i-1}] \geq a$. Применяя неравенство Azuma к случайным величинам $B_t \triangleq \sum_{i=1}^{t}(A_i - a)$, мы получаем

$$\Pr[\ell < (1-\delta_\ell)an] \leq \exp\left(-\frac{\delta_\ell^2 a^2 n}{2(1-a)^2}\right) \leq \exp\left(-\frac{\delta_\ell^2 a^2 n}{2}\right) \tag{6}$$

Установив этот предел длины, мы должны иметь $\#_0(X) \geq \#_0(Z_1, \ldots, Z_\ell) - \Delta$.Применим $\text{Lemma 1 to } Z_i$, мы омрачили это

$$\Pr[\#_0(Z_1, \ldots, Z_\ell) \leq (1-\delta_0)\frac{(1+\epsilon)}{2}\ell] \leq \exp\left(-\frac{\delta_0^2 \ell}{2}\right) \tag{7}$$

Объединение этих двух плохих событий дает уравнение (4).

Доказательство (ii). Предполагая наихудший случай $\ell = an$ и применяя Lemma 1 к префиксу Z_is, мы немедленно получаем (5).

Теорема 2. Рассмотрим протокол OUROBOROS-SAMASIKA с использованием **maxvalid-sc**, как описано на рисунке 7, выполненный в $\mathcal{F}_{\text{N-MC}}$-регистрации. Пусть f будет коэффициентом активного слота, пусть Δ будет верхней границей сетевой задержки. Пусть $\alpha,\beta \in [0, 1]$ обозначают нижнюю границу коэффициента оповещения и коэффициента участия на протяжении всего выполнения соответственно. Пусть и обозначают длину эпохи и общее время жизни системы (в слотах). Если для некоторого $\epsilon_w \in (0, 1)$ имеем $\alpha \cdot (1 - f)^{\Delta+1} \geq (1 + \epsilon_w)/2$ и если параметры **maxvalid-sc**, ϵ_w , s_{CG}, $s_{\exists\text{CQ}}$ и сетевой параметр удовлетворяют

$$288\Delta/(\epsilon\beta) < k,\ 2(1+\epsilon_w)s_{\text{CG}} + \Delta \leq R/3 \text{ или } s_{\text{CG}} + s_{\exists\text{CQ}} \leq R/3,$$

тогда следующие гарантии для общего префикса, роста цепи, качества цепи и качества экзистенциальной цепи сохраняются, за исключением дополнительной вероятности ошибки

$$\exp(\ln L - \Omega(k)) + \epsilon_{\#_0}(\beta f, \omega) + \epsilon_{\#_1}(\beta f, \omega)$$
$$\epsilon_{\text{CG}}(\beta f/16, s_{\text{CG}}) + \epsilon_{\exists\text{CQ}}(s_{\exists\text{CQ}}) + \epsilon_{\text{CP}}(s_{\text{CG}})$$

- **Общий префикс**. Вероятность того, что протокол нарушит свойство общего префикса с параметром k, не более

$$\epsilon_{\text{CP}}(k) \triangleq \frac{19L}{\epsilon^4}\exp(\Delta - \epsilon^4 k/18) + \epsilon_{\text{lift}};$$

- **Цепной рост**. Вероятность нарушения протоколом свойства роста цепи с параметрами $S \geq s_{\text{CG}} \triangleq 48\,\Delta/(\epsilon\beta f)$ або $\tau_{\text{CG}}=\beta f/16$

$$\epsilon_{\text{CG}}(\tau_{\text{CG}}, s) \triangleq \frac{sL^2}{2}\exp(-(\epsilon\beta f)^2 s/256) + \epsilon_{\text{lift}};$$

- **Экзистенциальное качество цепи**. Вероятность нарушения протоколом свойства качества экзистенциальной цепи с параметром $s \geq s_{\exists CQ} \triangleq 12\Delta/(\epsilon\beta f)$ не более чем

$$\epsilon_{\exists CQ}(s) \triangleq (s+1)L^2 \exp(-(\epsilon\beta f)^2 s/64) + \epsilon_{\text{lift}};$$

где ϵ_{lift} является сокращением для количества

$$\epsilon_{\text{lift}} \triangleq QL \cdot \left[R^3 \exp\left(-\frac{(\epsilon\beta f)^2 R}{768}\right) + \frac{38R}{\epsilon^4} \exp\left(\Delta - \frac{\epsilon^4 \beta f R R}{864}\right)\right]$$

Доказательство. Начнем с высокоуровневого описания доказательства. Напомним, что наша цель — показать, что при замене **maxvalid-bg** на **maxvalid-sc** общее выполнение протокола остается прежним. Чтобы увидеть это, рассмотрим запуск протокола с **maxvalid-bg** и рассмотрим первый sl_{curr}, когда честная сторона обнаружит форк дальнего действия. Пусть C_{loc} будет локальной цепью, а C_{cand} будет цепью-кандидатом. Мы покажем, что **maxvalid-sc** будет выводить те же рекомендации по цепочке, что и **maxvalid-bg**, с почти незначительной вероятностью. (Обратите внимание, что до sl_{curr} все выполнение происходило бы идентично, если бы стороны использовали вместо этого **maxvalid-sc**, так как в обоих случаях они всегда предпочли бы более длинную из сравниваемых цепочек, используя правило форка ближнего действия.) Это будет означать, что полный оператор, так как рассуждения могут быть индуктивно применены к каждому из слотов, где **maxvalid-bg** сталкивается с разветвлением дальнего действия, на протяжении всего выполнения.

Доказательство построено следующим образом:

1. В Lemma 3 мы покажем, что ω-окно с наименьшей плотностью в C_{loc} имеет по крайней мере t_{high} количество блоков.
2. В Lemma 4 и 5 мы рассмотрим конкретное ω-окно и установим верхнюю границу числа блоков в C_{cand} в этом окне.

Мы увидим, что $t_{high} > t_{low}$, тем самым установив теорему.

Lemma 3. Существует такое t_{high}, что **getMinDen**(C_{loc}) ≥ t_{high} , кроме как с вероятностью $\epsilon_{\#0}$ (βf,ω).

Доказательство. Пусть w_{loc}^{min} обозначает окно в C_{loc} с наименьшей плотностью (т.е. **getMinDen**(C_{loc}) = | C_{loc} [w_{loc}^{min}]. Пусть W обозначает характеристическую строку, вызванную выполнением этого протокола в пределах w_{loc}^{min} .

Обратите внимание, что количество блоков в w_{loc}^{min} не меньше $\#_0(W)$ так как $W_i = 0$ означает, что соответствующий слот имеет уникально предупреждающего лидера слота. Поэтому достаточно нижней оценки $\#_0$ W .

Из Lemma 2, полагая $a = \beta f$, получаем, что для любого $0 < \delta_l, \delta_{loc} < 1$,

$$\Pr[\#_0(W) \leq (1-\delta_{loc})\frac{(1+\epsilon)}{2}((1-\delta_\ell)\beta f \omega) - \Delta] \leq \epsilon_{\#_0}(\beta f, \omega)$$

Итак, мы это имеем $t_{high} = (1-\delta_{loc})\frac{(1+\epsilon)}{2}((1-\delta_\ell)\beta f\omega)-\Delta$ и $\text{getMinDen}(C_{loc}) \geq t_{high}$ разве что с вероятностью $\epsilon_{\#_0}(\beta f, \omega)$. □

Чтобы установить верхнюю границу плотности окна для C_{cand},, нам сначала нужно установить вспомогательную Lemma (Lemma 4), которая утверждает, что ни одна честная сторона не удерживает цепочку C_{cand} в любом слоте > $sl_{fork} + s_{CG} + \Delta$.

Lemma 4. Ни одна честная сторона не расширяет C_{cand} в любом слоте позже, чем $sl_{fork} + s_{CG} + \Delta$ кроме как с вероятностью s_{CG} ($\beta f/16$, s_{CG}) + $s_{\exists CQ}$ ($s_{\exists CQ}$) + s_{CP} (s_{CG}, $\beta f/16$).

Доказательство. На высоком уровне доказательство строится следующим образом. Рассмотрим два последовательных интервала слотов I_{growth} = [sl_{fork} + 1, sl_{fork} + s_{CG}] and $I_{stabilize}$ = [sl_{fork} + s_{CG} + 1, sl_{fork} + s_{CG} + $s_{\exists CQ}$]. Во-первых, мы покажем, что C_{loc} имеет большое количество блоков в I_{growth}, используя свойство роста цепочки; во-вторых, мы покажем, что у него есть хотя бы один честный блок в $I_{stabilize}$, используя свойство качества экзистенциальной цепи; наконец, в основе доказательства мы показываем, что если честная сторона удерживает C_{cand} в слоте позже, чем $sl_{fork} + s_{CG} + \Delta$, то это противоречит свойству префикса цепи.

В остальной части доказательства мы будем предполагать, что и

(CP) нет нарушения s_{CG} $\beta f/16$-CP

(∃CQ) нарушения $s_{\exists CQ}$-∃CQ нет, и

(CG) нарушения ($\beta f/16$, s_{CG})) - нет.

Заметим, что C_{loc} демонстрирует значительный рост на интервале I_{growth}: в частности, в силу свойства цепного роста, установленного в теореме 1 из [3],

$$|C_{loc}[I_{growth}]| \geq s_{CG}\beta f/16 \qquad (8)$$

Кроме того, обратите внимание, что C_{loc} имеет по крайней мере один честно сгенерированный блок на интервале $I_{stabilize}$:: в частности, поскольку $|I_{stabilize}| = s_{\exists CQ}$ и по свойству качества экзистенциальной цепи, установленному в теореме 1 из [3], должен существовать слот $sl^*loc \in I_{stabilize}$, для которого блок C_{loc} [sl^*loc]был сгенерирован честно.

Чтобы установить лемму, заметим, что честная партия, продолжающая C_{cand} после слота $sl_{fork} + s_{CG} + \Delta$ приведет к нарушению общего префикса. Предположим противное, что существует честно сгенерированный блок в C_{cand} после слотов $sl_{fork} + s_{CG} + \Delta$. Обозначим этот слот через sl^*_{cand}.

Пусть A = |C_{loc} [I_{growth}] | и пусть B = |C_{cand} [I_{growth}] |. Также обратите внимание на зубья $t$loc = Cloc [1, sl^*_{loc}] и $t$cand = Ccand [1, sl^*_{cand}].

Теперь рассмотрим следующие два случая.

Случай 1: B ≥ A. Напомним, что в C_{loc} есть честный блок на sl^*loc и в C_{cand} hasl^*cand. Следовательно, t_{loc} и t_{cand} являются жизнеспособными отростками. Это приводит к дивергенциидив(t_{loc}, t_{cand}) ≥ A. Иными словами, имеет место нарушение дивергенции (или CP-нарушение) с параметром $A \geq s_{CG}$ $\beta f/16$ (from (8)).

Случай 2: A > B: Вспомните, что существует честно сгенерированный блок в C_{cand} [sl^{*}_{cand}]. Также напомним, что $sl^{*}_{cand} > sl_{fork} + s_{CG} + \Delta$. Учитывая, что задержка Δ прошла, честная сторона, сгенерировавшая C_{cand} [sl^{*}_{cand}], полностью изучила C_{loc} [1, $sl_{fork} + s_{CG}$]. Поскольку честная сторона по-прежнему расширяла C_{cand} вместо C_{loc}, мы имеем, что $\hat{B} \triangleq |C_{cand}[sl_{fork} + 1, sl^{*}_{cand}]| \geq A$. Как и в предыдущем случае, это приводит к нарушению дивергенции с параметром $A \geq s_{CG}\, \beta f /16$.

Напомним, что getMinDen(C) выводит минимальную плотность блоков в окнах длиной w слотов. Рассмотрим окно, в котором getMinDen выводит плотность. Пусть слот непосредственно перед началом окна обозначается как sl_{min}. Рассмотрим интервал $I_{growth} = sl_{min}, sl_{min}\ \omega$. В силу свойства роста цепи, установленного в теореме 1 из [3], и предположения w = 1.1

Чтобы установить Lemma 5 и Lemma 3, мы будем использовать рост цепи и свойства качества экзистенциальной цепи; в частности, мы будем рассматривать два непересекающихся последовательных интервала длины s_{CG} и $s_{\exists CQ}$ между sl_{fork} и sl_{curr}.. Для этого нам нужно сначала показать, что между $s_{CG} + s_{\exists CQ}$ есть как минимум слоты $sl_{fork} + sl_{curr}$, чтобы мы могли применить свойства.

Lemma 5. **getMinDen**(C_{cand}) ≤ t_{low}, кроме как с вероятностью $\epsilon_{\#1}$ (βf, ω)

Доказательство. Чтобы установить верхнюю границу минимальной-плотности для C_{cand},, мы рассмотрим конкретное окно, близкое к разветвлению, и установим верхнюю границу его плотности. Отсюда следует, что min-плотность цепи не превышает верхней границы.

Теперь мы укажем окно, на котором мы сосредоточимся. Рассмотрим первое окно, которое начинается после слота $sl_{fork} + s_{CG} + \Delta$. Обозначим это окно через wc∗and.

Мы покажем, что $|w_{c^{*}and}| \leq t_{low}$.

Из Lemma 4 получаем, что в C_{cand} после слота $sl_{fork} + s_{CG} + \Delta$ нет честно сгенерированных блоков, кроме как с вероятностью s_{CG} (βf /16, s_{CG}) + $s_{\exists CQ}$ ($s_{\exists CQ}$) +s_{CP} (s_{CG}, βf /16). Следовательно, все блоки в C_{cand} после sl_{fork}+$s_{CG} + \Delta$

генерируются враждебно. Другими словами, $|w^{*}_{cand}| = \#_1(\hat{W})$, де $\hat{W}$ обозначает характеристическая строка, вызванная выполнением этого протокола внутри w^{*}_{cand}.

Теперь мы можем применить лемму 2, полагая $a = \beta f$, мы получаем, что для любого $0 < \delta_1 < 1$,

$$\Pr[\#_1(W) > (1+\delta_1)\frac{(1-\epsilon)}{2}\beta f - \Delta] \leq \epsilon_{\#_1}(\beta f, \omega)$$

Следовательно, мы имеем $t_{low} = (1 + \delta_1)^{\frac{(1-\epsilon)}{2}} \beta f - \Delta$ и **getMinDen**(C_{cand}) ≤ t_{low} разве что с вероятностью $s_{\#}1$ (βf , ω).

Последовательное распределение. Мы рассмотрели различные диапазоны слотов в Lemma 3, 4 и 5. Требуется, чтобы все эти диапазоны имели одинаковое распределение ставок и случайность, чтобы определить лидеров слотов во всех диапазонах.

Это обеспечивается следующими двумя ограничениями:

$$s_{CG} + \Delta + \nu + \omega \leq R/3$$
$$s_{CG} + s_{\exists CQ} \leq R/3$$

Эти ограничения соответствуют ограничениям в формулировке теоремы, поскольку мы поставили $\omega = (1 + s_w)s_{CG}$ и $\nu = s_w s_{CG}$.

Чтобы убедиться, что диапазоны слотов находятся в пределах sl_{fork} и sl_{curr}. Обратите внимание, что в приведенном выше доказательстве мы рассмотрели различные диапазоны позиций слотов. Остается убедиться, что все они лежат между sl_{fork} и sl_{curr}. Для этого мы сначала установим нижнюю границу $sl_{curr} - sl_{fork}$. Затем мы проиллюстрируем примеры присвоений различным параметрам, с которыми диапазоны слотов лежат в пределах.

Напомним, что sl_{fork} — это слот, связанный с последним общим блоком C_{loc} и C_{cand}. Напомним, что по замыслу протокола (независимо от базового правила **maxvalid**) для каждого слота sl_i существует событие E_i, такое, что если E_i происходит, то для слота sl_i. не может быть создан действительный блок. При этом события $E_1, E_2, \ldots$ независимы и $\Pr[E_i] = 1 - f$. Поэтому, используя границу Chernoff (см. Приложение 7.6.1) и границу объединения по времени работы системы, а также используя тот факт, что между sl_{fork} и sl_{curr}, имеется k блоков, мы можем ограничить количество слотов снизу. между sl_{fork} и sl_{curr}: $sl_{curr} - sl_{fork} \geq k/2f$, за исключением вероятности ошибки $\exp(\ln L - \Omega(k))$. В оставшейся части доказательства мы будем предполагать, что выполнение удовлетворяет slcurr: sl_{curr}: $sl_{curr} - sl_{fork} \geq k/2f$ для всех пар слотов, ограничивающих по крайней мере блоки в честно удерживаемой цепочке).

Ниже приведено потенциальное назначение, которое гарантирует, что диапазоны соответствуют ограничениям. s_{CG}, $s_{\exists CQ} = k/(6f)$. Обычно $\Delta \ll k/(6f)$. А полагая $\epsilon_w = 1$, мы получаем, что $2(1 + \epsilon_w)s_{CG} + \Delta \leq k/(2f)$. Далее, полагая $s_{\exists CQ} = k/(6f)$, получаем $s_{CG} + s_{\exists CQ} \leq k/(2f)$. В этих заданиях нам нужно было предположить, что $48\Delta/(\epsilon\beta) < k/6$ что согласуется с утверждением теоремы.

7.7 Эффективная реализация правила вилки ближнего действия

Напомним, что правило вилки ближнего действия заключается в том, чтобы просто выбрать самую длинную цепочку, как в Ouroboros Genesis. Напомним, что форк является ближней, если она меньше k блоков назад в истории. Наивная реализация этого правила состоит в том, чтобы всегда сохранять последние k блоков. Однако это неэффективно. Ниже мы предлагаем подход, при котором в любой заданный момент времени необходимо хранить информацию только о двух блоках.

Идея состоит в том, чтобы поддерживать две контрольные точки в каждую эпоху, которые могут дать оценку того, как давно произошло разветвление. Одна — «*начальная контрольная точка*», которая находится в начале каждой эпохи, а другая — «блокировочная контрольная точка», которая находится в последнем блоке первых двух третей эпохи. То есть в текущую эпоху, по мере того, как время уходит от первого слота, контрольная точка блокировки является последним блоком до тех пор, пока мы не достигнем последнего блока в первые две трети эпохи, когда контрольная точка блокировки «*замерзнет*» на тот блок. Эти контрольные точки сравниваются

для цепочек-кандидатов, чтобы определить, когда произошло разветвление. Для оценки положения вилки рассмотрим следующие два случая.

Вилка в текущую эпоху: мы классифицируем это как вилку ближнего действия. Это связано с тем, что распределение выбора лидеров для текущей эпохи уже было определено к концу первых двух третей предыдущей эпохи. Следовательно, мы можем с уверенностью предположить, что противник не искажал распределение для текущей эпохи, и в этом случае достаточно простого правила наибольшей длины цепочки.

Разветвление в предыдущей эпохе с той же контрольной точкой блокировки: поскольку контрольные точки блокировки для предыдущей эпохи одинаковы для цепочек-кандидатов,

как отмечалось в предыдущем случае, распределение выбора лидера хорошо распределяется даже после разветвления. Следовательно, опять же, достаточно простого правила самой длинной цепи.

8. Результаты экспериментов

Мы внедрили протокол Mina и запустили тестовую сеть с участием со всего мира. В этом разделе мы сообщаем результаты репрезентативной продолжительности между 12 ноября 2019 г., 10:00 по тихоокеанскому часовому поясу и 15 декабря 2019 г., 10:00 по тихоокеанскому часовому поясу.

8.1 Детали реализации

Реализация написана на OCaml. Сами SNARK написаны на специальном интуитивно понятном языке на основе OCaml под названием Snarky с бэкендом на основе libsnark [4]. Базовый протокол gossip основан на libp2p [1].

Инкрементально вычисляемый SNARK. Напомним, что протокол Mina основан на инкрементально вычисляемом SNARK. В реализации SNARK используется метод состояния параллельного сканирования из раздела 6.1, где для очереди блоков генерируется дерево доказательств SNARK, а корневое доказательство объединяется с доказательством для блокчейна, предшествующего очереди, чтобы получить новое доказательство для блока. обновленный блокчейн. Доказательство SNARK, подтверждающее достоверность блокчейна, называется доказательством блокчейна, а все остальные доказательства в деревьях называются доказательствами блоков. Напомним из Раздела 4.1.1, что под капотом есть три разных типа доказательств, а именно базовые доказательства, доказательства обертывания и доказательства слияния. По сути, у нас есть доказательства на основе блокчейна, доказательства на основе блокчейна, доказательства на основе блоков, доказательства на основе блоков и доказательства на основе слияния блоков. (Обратите внимание, что у нас нет доказательств слияния блокчейнов, поскольку доказательства блокчейна вычисляются только последовательно, и поэтому несколько доказательств блокчейна никогда не объединяются.)

Параметры консенсуса. Протокол Mina реализуется с помощью механизма консенсуса Ouroboros Samasika. Устанавливаем основные параметры консенсуса

Тип подтверждения Snark	Количество ограничений
блочное доказательство	42700
доказательство блокировки	34954
доказательство слияния блоков	206388
доказательство на базе блокчейна	248006
доказательство обертывания блокчейном	28313

Таблица 1: Количество ограничений в различных доказательствах, используемых для протокола Mina.

этэров при k = 10 (количество слотов до гарантированной завершенности), продолжительность слота
R = 240 с, а f = 0,5 (средняя доля заполненных слотов в эпоху).

Члены сообщества со всего мира участвовали в тестовой сети. Кроме того, несколько узлов также управлялись нами.

Напомним, что в протоколе Mina *каждый узел является полным узлом*, поскольку проверка всей цепочки блоков так же проста, как проверка только короткого доказательства постоянного размера. Однако каждый полный узел может иметь одну или несколько из следующих ролей: доказывающий и производитель блоков.

Всего в тестовой сети было 85 уникальных участников. Среди них было 49 производителей блоков (где 44 узла управлялись сообществом, а 5 — нами) и 8 уникальных пруверов.
Управляемые нами узлы-производители блоков использовали следующий размер экземпляра: «c5.2xlarge».

Управляемые нами узлы проверки использовали следующий размер экземпляра: «c5.9xlarge». Члены сообщества запускали свои узлы в Linux, OS X или Windows через WSL (подсистема Windows для Linux).

8.2 Результаты, достижения

Заполненные и незаполненные слоты. Обратите внимание, что общее количество слотов в интересующей продолжительности составляет 15839. Из них 7926 слотов были заполнены. Напомним, что f = 0,5 означает, что ожидаемая доля заполненных слотов равна 0,5. В эксперименте было заполнено 0,5004 доли слотов.

Транзакции и доказательства SNARK. Всего было отправлено 24826 транзакций, 17256 из которых от участников сообщества. Было 78 уникальных отправителей и 183 уникальных получателя. Всего было сгенерировано 53120 блочных доказательств SNARK.

Цифры 17, 18 и 19 показывают количество ежедневных транзакций, количество ежедневных блоков и количество ежедневных SNARK.

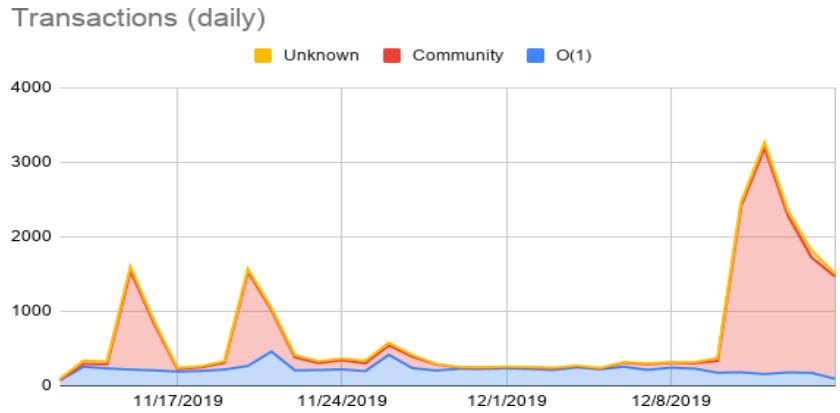


Рисунок 17. Транзакции, совершенные с 10:00 12 ноября 2019 г. до декабря. 15 декабря 2019 г., 10:00, тихоокеанский часовой пояс

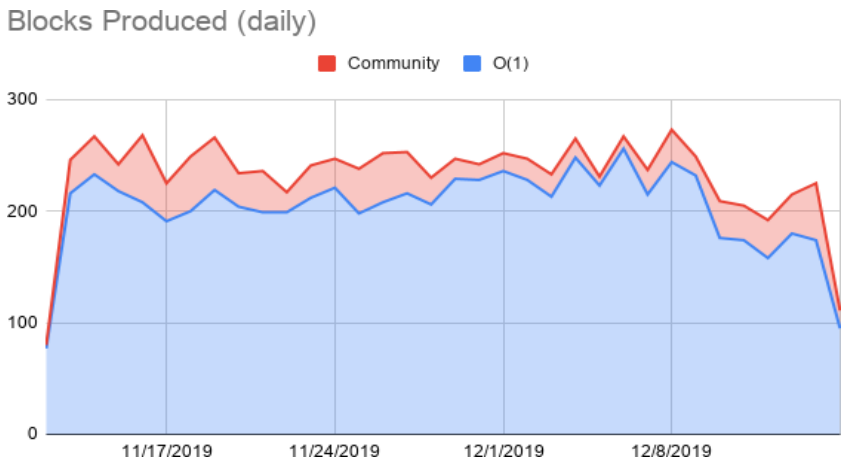


Рисунок 18: Блоки, произведенные с 10:00 12 ноября 2019 г. до 15 декабря. 2019, 10:00, тихоокеанский часовой пояс

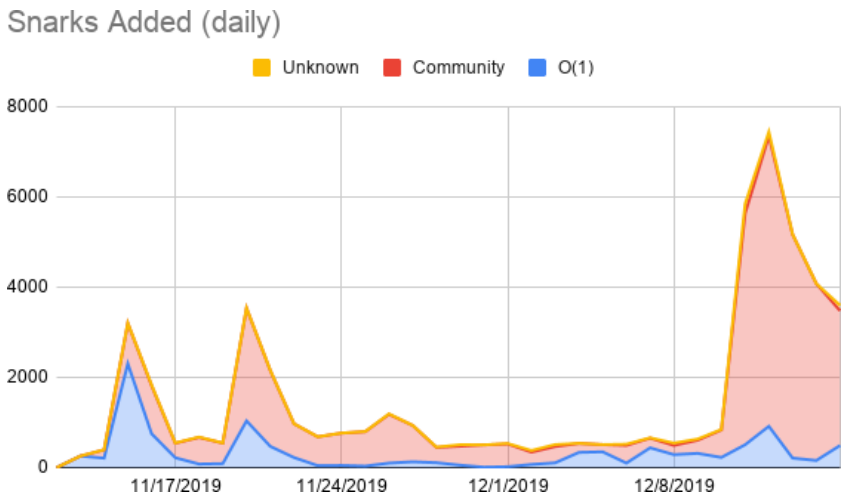


Рисунок 19: SNARK, созданные с 10:00 12 ноября 2019 г. до декабря. 15 декабря 2019 г., 10:00, тихоокеанский часовой пояс

9 Будущая работа

Хотя представленное описание протокола Mina предназначено для платежной системы, это понятие можно легко распространить на любые функциональные возможности, полные по Turing.

Например, инфраструктура может быть расширена для поддержки определяемых пользователем токенов и учетных записей с мультиподписью. Кроме того, наша дорожная карта включает обновление базовых SNARK до последних достижений в области исследований SNARK, например, с универсальной настройкой [9, 10].

10 Связанная работа

Mina, конечно, не единственный проект, работающий над поиском компромисса между масштабированием и децентрализацией. Действительно, этот компромисс был ключевой проблемой с самого начала блокчейна. Этот вопрос поднимался в самом первом ответе на оригинальный пост официального документа о Bitcoin [13].
С тех пор было предложено много решений, все с различными компромиссами [2]. Эти решения можно разделить на те, которые используют существующие цепочки, и такие, как Mina, которые предлагают новые архитектуры.

10.7 Существующие цепные решения

Сети Lightning и Plasma перемещают транзакции из основной цепочки в побочные каналы. Однако цепные операции по-прежнему требуют загрузки всего блокчейна и страдают от нерешенных проблем маршрутизации [22]. Также были критические атаки на Lightning, которые ограничивали его полезность [21].
Легкие узлы были предложены в качестве возможного решения для обеспечения более широкого доступа к криптовалютам. Они работают, загружая заголовки блоков, чтобы определить корень Merkle состояния базы данных, который имеет самое сильное состояние протокола.
Шардинг также был предложен как способ увеличения пропускной способности. Однако узлы имеют полную уверенность только в отношении осколков, для которых у них есть полные данные. В случае осколков, для которых узлы не имеют полных данных, эти узлы по существу должны доверять узлам консенсуса и при этом работают как легкие узлы. Кроме того, этот метод страдает от высокой стоимости загрузки нового сегмента при каждой ротации валидатора [20].
Еще одно предлагаемое решение для доступа к блокчейну — использование сторонних узлов. Вместо ненадежного подключения к блокчейну третья сторона управляет полным узлом, который используется для обновления состояния. По своей сути этот подход требует доверия третьей стороне. Такой доступ исключает как сопротивление цензуре, так и гарантированную живость.

Подтверждение

Мы благодарим Амита Сахаи за его ценные комментарии.

Использованная литература

[1] libp2p: модульный стек одноранговой сети. [Онлайн; по состоянию на февраль 15, 2020].

[2] Виталик Бутерин рассказывает о масштабируемости: «*Блокчейн Эфириума почти заполнен*», 2019 г. (по состоянию на октябрь 2019 г.). https://cointelegraph.com/news.

[3] Кристиан Бадерчер, Питер Гази, Ангелос Киайяс, Александр Рассел и Василис Зикас. Генезис Ouroboros: компонуемые блокчейны Proof-of-Stake с динамической доступностью. Cryptology ePrint Archive, отчет 2018/378, 2018. «https://eprint.iacr.org/2018/378».

[4] Эли Бен-Сассон, Алессандро Кьеза, Даниэль Генкин, Шауль Кфир, Эран Тромер, Мадарс Вирза и Ховард Ву. либснарк. https://github.com/scipr-lab/libsnark, 2017.

[5] Эли Бен-Сассон, Алессандро Кьеза, Эран Тромер и Мадарс Вирза. Масштабируемое нулевое знание с помощью циклов эллиптических кривых. В *CRYPTO* (2), объем 8617 *Конспектов лекций по информатике*, страницы 276–294. Спрингер, 2014.

[6] Эли Бен-Сассон, Алессандро Кьеза, Эран Тромер и Мадарс Вирза. Масштабируемое нулевое знание с помощью циклов эллиптических кривых. *Algorithmica*, 79 (4): 1102–1160, декабрь 2017 г.

[7] Нир Битански, Ран Канетти, Алессандро Кьеза и Эран Тромер. Рекурсивный композиция и начальная загрузка для SNARKS и подтверждающие данные. В СТОК, страницы 111–120. АКМ, 2013.

[8] Шон Боуи и Ариэль Габизон. Возможность извлечения симуляции zk-snark Грота в модели случайного оракула. Криптология Архив ePrint, отчет 2018/187, 2018. https://eprint.iacr.org/2018/187.

[9] Шон Боу, Джек Григг и Дайра Хопвуд. Halo: Рекурсивная композиция доказательства без надежной установки. Архив электронной печати *IACR Cryptology*, 2019: 1021,2019.

[10] Алессандро Кьеза, Юнконг Ху, Мэри Маллер, Пратюш Мишра, Ной Веселый и Николас П. Уорд. Marlin: Предварительная обработка zksnarks с помощью универсального и обновляемого SRS. Архив электронной печати *IACR Cryptology*, 2019: 1047,2019.

[11] Фил Даян, Рафаэль Пасс и Элейн Ши. Белоснежка: Надежно реконфигурируемый консенсус и приложения для доказуемой защиты Proof of Stake. В *Финансовая криптография*, том 11598 лекций по информатике, страницы 23–41. Спрингер, 2019.

[12] Бернардо Давид, Питер Гази, Ангелос Киайяс и Александр Рассел. Ouroboros praos: адаптивно-безопасное полусинхронное доказательство доли протокол. Cryptology ePrint Archive, Report 2017/573, 2017. http://eprint.iacr.org/2017/573.

[13] Джеймс А. Дональд. Бумага электронной наличности Bitcoin P2P, 2008 г. (по состоянию на октябрь 2019 г.).

[14] Хуан А. Гарай, Ангелос Киайяс и Никос Леонардос. Магистральный протокол биткойн: Анализ и приложения. В *EUROCRYPT* (2), том 9057 *Конспектов лекций по информатике*, страницы 281–310. Спрингер, 2015

[15] Йенс Грот и Мэри Маллер. Ядовитые подписи: Минимальные подписи знания от извлекаемых симуляцией снарков. В *CRYPTO* (2), объем 10402 *Конспектов лекций по информатике*, страницы 581–612. Спрингер, 2017.

[16] Ангелос Киайяс, Александр Рассел, Бернардо Давид и Роман Олейников. Ouroboros: доказуемо безопасный блокчейн-протокол с доказательством доли. В CRYPTO (1), том 10401 *Конспектов лекций по информатике*, страницы 357–388. Спрингер, 2017.

[17] Раджив Мотвани и Прабхакар Рагхаван. *Рандомизированные алгоритмы*. Издательство Кембриджского университета, Нью-Йорк, США, 1995.

[18] Сатоши Накамото. Bitcoin: одноранговая электронная денежная система». http://bitcoin.org/bitcoin.pdf, 2008 г.

[19] Эли Бен Сассон, Алессандро Кьеза, Кристина Гарман, Мэтью Грин, Ян Майерс, Эран Тромер и Мадарс Вирза. Zerocash: децентрализованные анонимные платежи в Bitcoin. В 2014 *Symposium on Security and Privacy*, страницы 459–474. ИИЭР, 2014.

[20] Александр Скиданов. *Unsolved Problems in Blockchain Sharding*, 2018 г. (по состоянию на октябрь 2019 г.). https://medium.com/nearprotocol.

[21] Саар Тохнер, Стефан Шмид и Авив Зоар. Угон маршрутов в оплате сети каналов: компромисс предсказуемости. CoRR, абс/1909.06890, 2019.

[22] Трастовые узлы. *У Lightning Network много проблем с маршрутизацией*, говорит ведущий Dev в Lightning Labs, 2019 г. (по состоянию на октябрь 2019 г.). https://www.trustnodes.com.

[23] Пол Валиант. Инкрементально проверяемые вычисления или доказательства знаний подразумевают эффективность времени/пространства. В TCC, том 4948 *лекций в Информатика*, страницы 1–18. Спрингер, 2008 г.

[24] Гэвин Вуд и др. Ethereum: безопасная децентрализованная обобщенная книга транзакций. Желтая книга проекта Ethereum, 151 (2014): 1–32, 2014 г.

Приложение A Выбор цепочки в Ouroboros Genesis

Здесь мы вспоминаем правило цепной селекции из Ouroboros Genesis. Протокол SelectChain такой же, как и в Ouroboros Samasika, за исключением вызова алгоритма **maxvalid-bg** вместо **maxvalid-sc**. Алгоритм **maxvalid-bg** вспоминается ниже.

Алгоритм maxvalid-bg$(C_{\text{loc}}, \mathcal{N} = \{C_1, \ldots, C_M\}, k, s)$

// Сравните C_{loc} с каждой цепочкой-кандидатом в $\mathcal{N}$

1. Настроить $C_{\text{max}} \leftarrow C_{\text{loc}}$
2. для $i = 1, \ldots, M$ выполнять

   если $(C_i$ вилки от C_{max} большинство k блоков назад $)$ тогда // Случай ближней вилки

   если $|C_i| > |C_{\text{max}}|$ тогда

   Настроить $C_{\text{max}} \leftarrow C_i$

   и тогда

   Еще // Случай длинной вилки

   Позволять $j \leftarrow \max\{j' \geq 0 \mid C_{\text{max}}$ и C_i есть такой же блок $\text{sl}_{j'}\}$

   если $\text{getLocalForkDen}(C_i, j + s) > \text{getLocalForkDen}(C_{\text{max}}, j + s)$ тогда

   Настроить $C_{\text{max}} \leftarrow C_i$

   конец, если

   конец если конец для
3. Возврат C_{max}

Рисунок 20: Правило выбора цепочки из книги Ouroboros Genesis.

Алгоритм $\text{getLocalForkDen}(C, n)$

возврат $|C[\text{sl}_1, \text{sl}_n]|$, количество блоков в первом n слоты C.

Рисунок 21: Алгоритм для сторон, чтобы получить плотность цепочки до слота, локального для разветвления.