Open Store

Decentralized Application Distribution Protocol

Andrei Chupin

July 30, 2025

Содержание

Context

Проект OpenStore был инициирован в 2025 году.

Первая версия Whitepaper служит основополагающим документом, описывающим архитектуру и принципы проекта до его публичного запуска в testnet.

Основная цель документа — структурировать ответы на ключевые вопросы:

- Мотивация и проблемы, которые решает проект
- Техническая реализация и архитектурные решения

1. Motivation

Существующие магазины приложений и других цифровых товаров построены как закрытые экосистемы (walled gardens), что создает ряд критических проблем при работе с ними в рамках Web3. Open Store предлагает альтернативу, решающую ряд проблем на уровне протокола.

1.1 A Protocol for Enhanced Security

Протокол OpenStore закладывает основу для построения более безопасных приложений за счет новых архитектурных возможностей:

- **DAO-Governed Releases:**

  Архитектура совместима с системами управления на базе DAO, что позволяет сообществам создавать полностью децентрализованные и прозрачные процессы публикации для критически важных объектов.

- **Separation of Concerns:**

  Индустрия тратит огромные ресурсы на безопасность: аудиты смарт-контрактов, формальная верификация кода, защитные механизмы и отказоустойчивость блокчейнов. Однако прочность любой системы безопасности определяется прочностью ее самого слабого звена. В экосистеме мобильных приложений существует огромная, но часто игнорируемая уязвимость — процесс публикации.

  Работая в одном из самых популярных криптокошельков, на протяжении многих лет у меня было право на force-push в основную ветку и публикацию новых версий приложения. И это не частный случай. Некоторые мои знакомые до сих пор работают в кошельках в тех же обстоятельствах. Потенциально, всего один человек может нанести индустрии ущерб на десятки и сотни миллиардов долларов.

  Решение данной проблемы являлось одной из главных задач при создании Open Store. Протокол позволяет реализовать модульную архитектуру, разделяя приложения на независимые компоненты с разным уровнем критичности. Это дает возможность изолировать наиболее чувствительные части (например, логику управления ключами) от элементов, требующих частого обновления (например, пользовательского интерфейса).

  Процесс публикации критических модулей может быть максимально прозрачным и безопасным, например, через верификацию сообществом или голосование в DAO. Это сделает скрытые атаки практически невозможными.

- **Local Artifact Verification**

  Протокол поддерживает механизм локальной верификации публикуемых файлов. Для этого автор публикует on-chain доказательство владения сертификатом, привязанное к адресу его контракта. При установке любой пользователь может запросить эти данные напрямую из блокчейна и убедиться в подлинности приложения.

1.2. Fair Fee Model

- **No Royalty Fees**

  Протокол не взимает процент с дохода авторов.

- **Explicit Network Fees**

  Все затраты сводятся к явным комиссиям блокчейна за конкретные операции. Автор платит только за те ресурсы, которые фактически использует.

- **Cost Optimization via Custom Distribution**

  Протокол не навязывает единый способ доставки контента. Для снижения расходов авторы могут использовать собственные или сторонние системы дистрибуции (CDN) для доставки файлов приложения.

1.3. Self-Custody

Протокол гарантирует, что разработчики сохраняют контроль над своими приложениями, а пользователи — над доступом к ним.

2. Principles

2.1. Openness

Протокол должен быть открытым, насколько это возможно:

- **Open Source**

  Исходный код протокола и его основных компонентов должен быть публично доступен для аудита и проверки.

- **Open Data**

  Все данные должны быть публичными и по возможности максимально храниться on-chain, обеспечивая полную прозрачность состояния экосистемы.

- **Open Participation**

  Протокол является открытым (permissionless), что позволяет участнику выбрать любую роль, придерживаясь установленных правил.

2.2. Decentralization

Стремление к устранению централизованных точек отказа и контроля.

- **Self-Custody by Design**

  Протокол должен поддерживать self-custody всех компонентов, гарантируя, что пользователи и издатели сохраняют полный контроль над своими файлами и данными.

- **No Single Point of Failure**

  Архитектура должна быть устойчивой и независимой от какой-либо одной сущности для своей работы или доступности.

- **Consensus-Driven Governance**

  Критически важные решения могут управляться DAO.

- **On-Chain Source of Truth**

  Основные метаданные и изменения состояния должны фиксироваться on-chain, создавая неизменяемый и проверяемый источник истины.

2.3. Viability and Pragmatism

Протокол должен соблюдать баланс, избегая крайностей.

Протокол стремится хранить максимум данных в блокчейне (on-chain), однако это создает проблему с хранением больших файлов. Многие приложения достигают 100–150 МБ и обновляются еженедельно, а в случае с мобильными играми речь идет о гигабайтах данных. Требование хранить и оплачивать все эти файлы в сети сделало бы протокол нежизнеспособным.

Следуя данному принципу, мы предоставляем возможность авторам удалять старые, неиспользуемые версии публикаций. Вместе с этим у валидаторов пропадает возможность самостоятельно проверить все цепочки блоков.

Является ли это критичным?

- **Atomic Publication.** В отличие от классических блокчейнов, где вся история транзакций необходима для воссоздания текущего состояния (например, балансов), здесь каждая публикация является независимой единицей. Удаление старой версии никак не влияет на валидность последующих.
- **Dual-Layer Validation.** On-chain валидация в первую очередь определяет, будет ли контент отображаться в общем каталоге (например, в поиске). Финальную и самую важную проверку подлинности и целостности файлов выполняет клиентское приложение локально, непосредственно перед установкой или взаимодействием.
- **Ecosystem Integrity & Spam Prevention.** Без on-chain валидации каталог мог бы наполниться приложениями с некорректными подписями или невалидными данными (случайно или умышленно). Это привело бы к массовым ошибкам на стороне пользователей. Таким образом, валидация служит важным барьером, отсеивающим некачественный или вредоносный контент на начальном этапе.

Возможность удаления старых версий — это прагматичный компромисс между полной децентрализацией и экономической реальностью.

2.4. Symmetric Incentives: Advantage & Responsibility

Следуя Game Theory, я старался спроектировать протокол так, чтобы каждый актор должен был получать какую-либо выгоду при использовании протокола, при этом возлагая на него соответствующий уровень ответственности за свои действия.

Actor	Advantage	Responsibility
User	Предоставление доступа к безопасным приложениям для критически важных задач и к устойчивой к цензуре платформе	Проявление должной осмотрительности при установке и взаимодействии с новыми приложениями
Publisher	Отсутствие процентных сборов, устойчивость к блокировкам, доступ к пользователям, возможность создания прозрачного процесса разработки, релизов и публикации	Обеспечение доступности и распространения данных своего приложения.
Validator/Oracle	Вознаграждения за валидацию данных	Наказание (slashing) за вредоносное поведение или предоставление неверных данных.
Owner	Вознаграждения за развитие и поддержание работы протокола	Долгосрочное развитие и поддержание здоровья экосистемы.

2.5. Humanity and Responsible Freedom

Одна из целей Web3 — создание более свободных систем. Однако свобода не должна означать вседозволенность. Протокол должен находить баланс между открытостью и предотвращением очевидно деструктивного использования.

Для реализации данной концепции в протоколе заложен следующий компромисс:

- Платформа не предназначена для файлов и приложений, основной функцией которых являются общепризнанные преступления (см. Приложение A. Inhumane list).
- Подобного рода материалы могут быть скрыты из публичных каталогов и результатов поиска по имени и описанию.
- Любой контент навсегда остается доступным при поиске по адресу конкретного файла или приложения.

Пример 1 (Допустимо)

Обычный мессенджер, используемый для продажи тяжелых наркотиков, не несет ответственности за действия пользователей.

Пример 2 (Недопустимо)

Мессенджер, главной функцией или позиционированием (на это так или иначе должен указывать UI) которого является продажа тяжелых наркотиков или продажа людей, будет скрыт из общего поиска.

Этот механизм позволяет защитить пользователей от наиболее вредоносного контента, не нарушая при этом базовый принцип открытости.

3. Overview

3.1. Glossary:

Ссылки на разделы Glossary:

[a] - Actor

[e] - Entity

[u] - UI Application

[c] - Contract

[p] - Process

[n] - Node

[s] - Smart Contract Structure *(добавлено для ясности, так как используется в тексте)*

[f] - Fee *(добавлено для ясности, так как используется в тексте)*

Изменяемость данных:

(immutable) - Данные контракта, которые задаются один раз.

(mutable) - Данные контракта, которые можно изменить.

3.1.1. Actor

- **Publisher**

  Публикует [e]Asset в [u]Open Store Studio.

- **Validator**

  Обрабатывает [s]ValidationRequest, создает блоки, участвует в голосовании.

- **Oracle**

  Синхронизирует данные из Web2 в Web3.

- **Broadcaster**

  Отслеживает события в smart-контрактах, хранит их в структурированном виде и предоставляет пользователям через API.

- **Owner**

  Принимает участие в разработке или управлении Open Store.

- **User**

  Непосредственный пользователь [u]Open Store App, взаимодействующий с [e]Asset.

3.1.2. Node

- Validator
- Oracle
- Daemon (Broadcaster)
- API (Broadcaster)
- Statistic (Broadcaster)
- Blockchain (BSC, opBSC, Greenfield)

3.1.3. Entity

- **Asset**

  Любой тип публикуемого ресурса (приложение, игра, книга и т.д.).

- **Assetlink**

  Механизм, связывающий домен (сайт) с приложением. Включает в себя сертификаты и данные о приложениях, которые имеют право ссылаться на этот домен для подтверждения связи (см. **https://developer.android.com/training/app-links/verify-android-applinks**).

- **Asset Artifact**

  Файл, представляющий собой конкретную, уникальную версию [e]Asset. Имеет on-chain представление в виде [s]BuildInfo и хранится в хранилище, принадлежащем [a]Publisher.

- **Asset Endpoint**

  Сайт владельца приложения. Этот домен виден пользователю ([a]User) и используется им для принятия решения об установке.

  Должен содержать путь **$ENDPOINT/.well-known/assetlinks.json** с перечислением всех **Certificate SHA-256 Fingerprint**, которыми подписано приложение

  (см. **Android Assetlinks**).

- **3.1.4. Greenfield**

Блокчейн для хранения файлов от компании Binance.

- **Bucket**

  Пространство имен в хранилище Greenfield для хранения файлов. Аналогично AWS S3.

- **Ownership Version**

  Версия данных о владении [e]Asset. Хранится в плагине [c]AppOwnerPluginV1 и имеет on-chain представление в виде [s]OwnerInfo. Автоматически повышается при обновлении [s]OwnerInfo.

- **Cross Chain**

  Мост, обеспечивающий взаимодействие между блокчейнами BSC, opBSC и Greenfield.

- **ProofOfCertificateOwnership**

  Цифровая подпись, подтверждающая, что [a]Publisher владеет сертификатом. Формируется путем подписания строки данных формата 'APP_ADDRESS::SHA_256_CERT_FINGERPRINT' приватным ключом, соответствующим этому сертификату.

- **Distribution Link**

  Ссылка на сервер для раздачи файла. [a]Publisher могут использовать такие ссылки для оптимизации затрат на передачу данных (например, через CDN).

- **Validation Block**

  Объект в формате Protobuf, содержащий метаданные блока и информацию о валидации запросов. Передается в поле calldata BSC-транзакции. Ссылка на транзакцию хранится в [s]BlockRef.

- **Proposal Window**

  Время, в рамках которого [e]Validator обязан выполнить [p]Block Proposing.

  Устанавливается [a]Owner в контракте [c]OpenStore.

3.1.5. UI

- **Open Store Studio**

  Сайт для создания, управления и публикации [e]Asset.

- **Open Store App**

  Приложение для поиска, установки и обновления [e]Asset.

3.1.6. Contract

- **OpenStore**

  Основной контракт, в котором реализована система консенсуса, хранящий опубликованные версии [e]Asset, статусы их валидаций, а также данные самих валидаторов.

- **AssetlinkOracleV1**

  Основной контракт для хранения запросов и результатов [p]Ownership Verification.

- **Plugins**

  Контракты, предназначенные для расширения базовой функциональности других контрактов.

- **DevFactory**

  Контракт-фабрика для создания DevAccount.

- **DevAccount** — представление [a]Publisher в системе.
  - **DevAccount**

    Хранит имя пользователя издателя и подключенные плагины.

  - **AppsPluginV1**

    Хранит приложения, принадлежащие издателю.

  - **GreenfieldPluginV1**

    Контракт для [e]Cross-Chain взаимодействия с [e]Bucket в хранилище [e]Greenfield.

- **App** — представление [e]Asset типа «Application».
  - **App**

    Хранит общую информацию о приложении.

  - **BuildsPluginV1**

    Хранит версии приложения и ссылки на on-chain файлы.

- **OwnershipPluginV1**

  Хранит информацию о владении приложением и доказательства этого владения.

- **DistributionPluginV1**

  Хранит Web2/Web3 ссылки для распространения (скачивания) файлов.

3.1.7. Smart Contract Structures

- BuildInfo - представление [e]Asset Atifact
  - versionCode - int64
  - versionName - string
  - referenceId - bytes
  - protocolId - uint16
  - checksum - bytes
- OwnershipInfo - данные владения [e]Asset
  - endpoint - string
  - proofs
    - sha256CertificateFingerprint - bytes32
    - proofOfOwnership = signature(appAddress::sha256CertificateFingerprint) - bytes
- BlockRef - метаданные [e]Validation Block
  - id - uint256
  - fromRequestId - uint256
  - toRequestId - uint256
  - result - uint256
  - objectId - bytes
  - protocolId - int32
  - objectHash - bytes32
  - blockMask - uint8
  - createdBy - address
- ValidationRequest - абстрактные данные для валидации [e]Asset
  - id - uint256
  - type - uint32
  - target - Address
  - data - bytes

3.1.8. Process

- **Ownership Verification**

  Процесс сверки данных из плагина [c]AppOwnerPluginV1 с информацией, опубликованной по адресу ENDPOINT/.well-known/assetlinks.json, для подтверждения связи между приложением и доменом.

- **Artifact Validation**

  Процесс валидации (проверки) новой версии [e]Asset.

- **Block Proposing**

  Процесс, в ходе которого [a]Validator предлагает [e]Validation Block на рассмотрение.

- **Block Voting**

  Процесс, в ходе которого [a]Validator голосует за существующий [e]Validation Block в рамках [p]Block Proposing или создает новый, инициируя [p]Block Discussing.

- **Block Discussing**

  Процесс, который наступает, когда на голосование выдвинуто как минимум два конкурирующих [e]Validation Block.

- **Block Finalisation**

  Процесс, в котором данные победившего [e]Validation Block фиксируются в состоянии [c]OpenStore, а также происходит награждение победивших и наказание (slashing) проигравших [a]Validator.

3.1.9. Fees

- **Network Fee**

  Gas Fee - Стандартная комиссия за выполнение транзакции

  Cross Chain Fee - Комиссия за отправку сообщения из BSC в Greenfield

- **Greenfield Fee** - (см. **Greenfield Billing**)

  Download Quote Fee - Еженедельная предоплата за определенный объем (в ГБ), который пользователи могут скачать из вашего [e]Bucket

  Storage Fee - Еженедельная предоплата за хранение ваших файлов

- **Oracle Fee**

  Комиссия от [a]Publisher в пользу [a]Oracle за проведение [p]Ownership Verification.

  Размер комиссии устанавливается [a]Owner в [c]AssetlinkOracleV1.

- **Validation Fee**

  Комиссия от [a]Publisher в пользу [a]Validator за обработку [e]Validation Request.

  Размер устанавливается [a]Owner в контракте [c]OpenStore.

- **Proposal Stake**

  Залог (стейк), который [a]Validator вносит за [p]Block Proposing. В случае, если другой [a]Validator оспорит блок в рамках [p]Block Discussion, первый потеряет свой стейк, а награда делится между победителями (инициатором [p]Block Discussion и проголосовавшими за него).

- **Vote Stake**

  Залог (стейк), который [a]Validator вносит за [p]Block Voting. В случае, если [a]Validator проиграет голосование в рамках [p]Block Discussion, все [a]Validator, проголосовавшие за проигравший [e]Validation Block, потеряют свой стейк, а награда делится между победителями (инициатором [p]Block Discussion и проголосовавшими за него).

- **Proposal Penalty**

  Штраф для [a]Validator, если он не выполнил [p]Block Proposing в течение времени [e]Proposal Window после того, как был выполнен [p]Block Proposing для предыдущего блока, при условии наличия хотя бы одного [e]Validation Request в очереди.

  Размер устанавливается [a]Owner в контракте [c]OpenStore.

- **Minimum Stake**

  Минимальный размер баланса валидатора в [c]OpenStore для регистрации в сети. Вычисляется по формуле:

$$\text{Vote Stake} \times (\text{MAX\_CONCURRENT\_VOTINGS} - 2) + \text{Proposal Stake} \times 2$$

  MAX_CONCURRENT_VOTINGS — максимальное количество одновременно активных [p]Block Voting.

3.2. Workflow

TL;DR

Ниже представлена упрощенная модель взаимодействия основных акторов с протоколом. В реальности некоторые пункты могут быть выполнены в рамках одной атомарной операции.

1. Publisher создает и настраивает contract Publisher Account.
2. Publisher создает новый contract Asset (Application).
3. Publisher указывает данные владения (Ownership Info) для нового Asset (домен сайта, хеши сертификатов, а также доказательства владения данными сертификатами).
4. Publisher отправляет Ownership Info на верификацию (Ownership Verification).
5. Oracle выполняет Ownership Verification.
6. Publisher загружает Asset Artifact (файл новой версии Asset).
7. (Optional) Publisher может добавить собственные ссылки для распространения Asset Artifact (e.g. CDN).
8. Publisher отправляет новый Asset Artifact на валидацию (Artifact Validation).
9. Validator выполняет Artifact Validation, проверяя структуру и подписи Asset Artifact.
10. Publisher публикует провалидированный Asset Artifact в общий доступ.
11. Daemon синхронизирует необходимые данные из Blockchain в DB.
12. API предоставляет данные пользователям из DB в структурированном формате.
13. User взаимодействует с Asset (Asset Artifact) через приложение.

3.2.1. Создание contract DevAccount

1. [a]Publisher создает в [u]Open Store Studio новый [c]DevAccount через [c]DevFactory, указывая:
    1. Name (immutable) - unique username
    2. File Storage (mutable) - хранилище для [e]Asset Artifact
        1. На данный момент единственное доступное хранилище файлов — [e]Greenfield.
2. При создании к [c]DevAccount подключаются 2 плагина — [c]DevGreenfieldPluginV1 и [c]DevAccountAppsPluginV1.
3. В рамках подключения [c]DevGreenfieldPluginV1 происходит:
    1. Отправка минимального количества BNB в [e]Greenfield для пополнения баланса и оплаты хранилища, используя [e]Cross Chain.
    2. Создание [e]Bucket в [e]Greenfield, используя [e]Cross Chain.

3.2.2. Управление DevAccount

[a]Publisher, используя [u]Open Store Studio, может менять такие параметры, как:

1. Bucket Read Quote - размер данных в GB, доступных для скачивания из [e]Bucket. При превышении лимита [n]Greenfield может предоставлять данные на минимальной скорости ИЛИ полностью блокировать раздачу данных (для оптимизации можно использовать [e]Distribution Link).
2. Bucket Balance - оплата хранения и раздачи файлов осуществляется в [e]Greenfield BNB; пополнять баланс можно напрямую с кошелька BSC BNB, используя [e]Cross Chain.
3. Invisibility - параметр, позволяющий [a]Publisher скрывать приложение в [c]OpenStore. После активации опции [e]Asset будет невозможно найти в [u]Open Store App до момента выключения данной функции.

3.2.3. Создание App

1. [a]Publisher создает новый [c]App через [c]DevAccountAppsPluginV1 в [u]Open Store Studio, указывая:
    1. PackageName (immutable) - text unique identificator
    2. Name (mutable)
    3. Description (mutable)
    4. ProtocolId (mutable) - app metadata storage
    5. PlatformId (immutable) - type of OS platform (e.g. Android, iOS etc)
    6. CategoryId (mutable) - type of [e]Asset category (Books, Tools, Sport etc)
2. При создании [c]App подключает к себе 3 базовых плагина — [c]AppOwnerPluginV1, [c]AppBuildsPluginV1 и [c]AppDistributionPluginV1.

3.2.4. Ownership Verification для Android App

1. [a]Publisher, используя [u]Open Store Studio, заполняет форму владения ([s]OwnershipInfo):
    1. [e]Asset Endpoint (mutable)
    2. Доказательства владения сертификатами подписи Android-приложения:
        1. **Certificate SHA-256 Fingerprint** (mutable) - e.g. F8:F9:21:DA:21:05:21:A2:21:BC:21:9A:21:81:21:E0:21:AB:21:2D:93:EA:53:41:7A:45:81:98:F8:ED:5D:80)
        2. **[e]ProofOfCertificateOwnership** (mutable)
2. [a]Publisher сохраняет [s]OwnershipInfo в [c]AppOwnerPluginV1.
3. [a]Publisher отправляет запрос [p]Ownership Verification в [c]AssetlinksOracle.

    1. [p]Ownership Verification требует оплаты комиссии [f]Oracle Fee.
    2. В случае успеха [a]Publisher будет иметь возможность отправлять [e]Asset Artifact верифицированного [c]App на [p]Artifact Validation в [c]OpenStore.
    3. В случае ошибки [a]Publisher теряет возможность отправлять на [p]Artifact Validation новые [e]Asset Artifact (если до этого последняя версия [s]OwnershipInfo была верифицирована); все ранее опубликованные [e]Asset Artifact останутся доступными.
4. При изменении [s]OwnershipInfo в [c]AppOwnerPluginV1 повышается [e]Ownership Version, после чего необходимо повторно пройти [p]Ownership Verification.
    1. Количество раз, которое одна и та же [e]Ownership Version может быть отправлена на [p]Ownership Verification, неограниченно.
5. В идеале [c]App проходит [p]Ownership Verification всего один раз; результат данной проверки переиспользуется в последующих проверках.

3.2.5. Роль Oracle

1. В данный момент [a]Oracle централизован и находится во владении [a]Owner.
2. [a]Oracle необходим, чтобы [a]Validator в процессе [p]Artifact Validation мог получить все необходимые данные напрямую из [n]Blockchain. В ином случае открывается огромное количество возможностей для манипуляций с данными, что, в свою очередь, ведет к проблемам безопасности всего протокола.
3. Функции [a]Oracle могли бы быть возложены на [a]Validator и локальную проверку [u]Open Store App, но в данной имплементации [a]Validator не гарантируют 100% надежность верификации, а у [u]Open Store App могут быть проблемы с доступом к [e]Asset Endpoint.
4. [a]Oracle будет децентрализован в последующих релизах, когда все остальные компоненты системы будут стабилизированы.

3.2.6. Обработка запроса Ownership Verification для Android App

1. [a]Oracle получает запрос в виде события [n]Blockchain.
2. [a]Oracle пытается получить JSON, используя Asset Endpoint **$ENDPOINT/.well-known/assetlinks.json** (см. **https://developer.android.com/training/app-links/verify-android-applinks**).
    1. Если [e]Asset Endpoint недоступен, [p]Ownership Verification завершается **ошибкой**.
3. [a]Oracle в полученном JSON пытается найти [e]Assetlink, соответствующий [c]App (e.g. в случае с Android “namespace”: “android_app” и”package_name”: “org.openstore.example.android”).

    1. Если [e]Assetlink не был найден, [p]Ownership Verification завершается **ошибкой**.
4. [a]Oracle проверяет все ‘sha256_cert_fingerprints’ найденного [e]Assetlink с теми, что указаны в [c]AppOwnerPluginV1.
    1. Если какой-либо ‘sha256_cert_fingerprints’ отсутствует в [c]AppOwnerPluginV1, [p]Ownership Verification завершается **ошибкой**.
    2. Если контракт AppOwnerPluginV1 содержит все ‘sha256_cert_fingerprints’, [p]Ownership Verification завершается **успешно**.
5. [a]Oracle в качестве награды получает [f]Oracle Fee.

3.2.7. Artifact Validation для Asset Artifact в OpenStore

1. [a]Publisher, используя [u]Open Store Studio, указывает:
    1. [e]Asset Artifact для валидации.
        1. В случае необходимости [a]Publisher загружает новый файл, используя [u]Open Store Studio.
    2. Необходимость выполнения [p]Artifact Validation (см. 2.2.17).
2. [a]Publisher отправляет [e]Validation Request в [c]OpenStore.
    1. [p]Artifact Validation требует оплаты [f]Validation Fee.
    2. В случае ошибки [a]Publisher может пройти верификацию [e]Asset Artifact еще раз.
    3. В случае успеха валидируемый [e]Asset Artifact обозначается как валидный в [c]OpenStore и может быть опубликован.
    4. Каждый валидируемый [e]Asset Artifact должен иметь номер версии больше, чем последний, успешно прошедший валидацию.

3.2.8. Регистрация Validator

1. [a]Validator запускает [n]Validator, указав необходимые параметры для работы.
2. [a]Validator пополняет баланс в [c]OpenStore.
    1. [a]Validator имеет право вывода средств из [c]OpenStore.
3. [n]Validator регистрирует себя в [c]OpenStore.
    1. Для регистрации необходимо иметь на балансе [f]Minimum Stake.
4. [n]Validator пытается назначить себя на валидацию блока N.
    1. **IMPORTANT!** Если ближайший свободный блок больше текущего валидируемого блока на X блоков (параметр задается [a]Owner в [c]OpenStore), [n]Validator обязан заплатить:

$$\text{Voting Amount} = \frac{\text{Total Stake}}{\text{Validator Stake}} - 1$$

2. **EXAMPLE!**
    1. Total Stake = 150,
    2. Validator Stake = 15
    3. (Total Stake / Validator Stake) - 1 = 9
    4. В данном случае размер Validator Stake составляет 1/10 от общего количества, что значит, что мы будем должны каждый 10-й блок.
3. Чтобы убедиться, что [a]Validator участвовал в создании блоков, ему необходимо голосовать за чужие блоки, тем самым пополняя Voting Amount.
    1. За каждый голос присваивается 1 Voting Amount.
    2. Все расчеты с Voting Amount проходят с точностью 10^9, или 1 gwei.
4. Если [n]Validator не имеет достаточного количества Voting Amount, он продолжает функционировать, выполняя [p]Artifact Validation и голосуя за другие блоки.
5. [n]Validator должен временно заблокировать в контракте сумму [f]Proposal Stake.

5. [n]Validator обязан выполнить дерегистрацию при завершении работы.

3.2.9. Обработка Artifact Validation для Android Asset Artifact

1. [a]Validator получает событие (event) из [n]Blockchain.
2. Используя данные из события, [a]Validator скачивает APK ([e]Asset Artifact).
3. [a]Validator парсит метаданные APK (см. **APK Signing V2**).
4. [a]Validator проверяет метаданные APK:
    1. APK должен иметь валидную структуру и подпись.
    2. VersionCode, PackageName и APK Checksum должны совпадать с теми, что указаны в [e]Asset Artifact и [c]App.
    3. Все SHA256 Certificate Fingerprint должны быть указаны в [c]AppOwnerPluginV1.
    4. Все [e]ProofOfCertificateOwnership должны быть валидными; проверка сигнатуры происходит путем ручного воссоздания материала подписи и использования публичного ключа сертификата из APK.
5. [a]Validator сохраняет результат проверки до [p]Block Proposal или [p]Block Voting.
6. После создания блока все результаты проверки сохраняются в [c]OpenStore.

3.2.10. Формирование и выдвижение блока на голосование Validator

1. Выдвижение блоков на голосование происходит инкрементально.
    1. **EXAMPLE!** Нельзя выдвинуть на голосование блок №5, если блок №4 еще не был выдвинут.

2. [a]Validator дожидается момента, когда очередь дойдет до него.
3. [a]Validator формирует [e]Validation Block из имеющихся результатов валидации [p]Artifact Validation.
    1. [a]Validator обязан предложить блок в рамках [e]Proposal Window, иначе он будет оштрафован на [f]Proposal Penalty, а любой другой [a]Validator получит право на [p]Block Proposing данного блока, получая в награду [f]Proposal Penalty.
4. [a]Validator отправляет Validation Block в бинарном ProtoBuf-формате в Binance Smart Chain в виде calldata, дожидаясь выполнения транзакции.
5. [a]Validator выдвигает блок (ссылку на структуру данных) на голосование в [c]OpenStore.

3.2.11. Голосование за блок Validator

1. [a]Validator должен сформировать собственный [e]Validation Block для всех запросов, используемых в исходном [e]Validation Block.
    1. Если сформированный блок отличается от исходного, [a]Validator может начать [p]Block Discussing и предложить свою версию блока.
        1. Для выдвижения альтернативной версии блока необходимо заблокировать в контракте сумму [f]Proposal Stake.
        2. Если выдвигаемый блок проиграет в голосовании, [f]Proposal Stake будет распределен между победителями, иначе [f]Vote Stake возвращается [a]Validator.
    2. Если сформированный блок совпадает с исходным, [a]Validator голосует за него.
        1. Для голосования необходимо заблокировать в контракте сумму [f]Vote Stake.
        2. Если блок проиграет в голосовании, [f]Vote Stake будет распределен между победителями, иначе [f]Vote Stake возвращается [a]Validator.
2. Замечания:
    1. В один момент времени, в рамках одного блока, [a]Validator может выполнять только одну из следующих функций:
        1. [p]Block Proposing
        2. [p]Block Discussing
        3. [p]Block Voting

3.2.12. Финализация блока Validator

1. Финализация, как и выдвижение блока, происходит инкрементально.

    1. **EXAMPLE!** Нельзя финализировать блок №5, если блок №4 еще не был финализирован.
2. [a]Validator инициирует финализацию блока в [c]OpenStore.
    1. Финализация блока происходит в том случае, если:
        1. блок набрал достаточное количество голосов, так, что оставшиеся голоса не могут повлиять на ситуацию;
        2. время для голосования вышло.
3. После финализации данные из блока сохраняются в [c]OpenStore.
4. [a]Validator, выполняющие [p]Block Voting и [p]Block Discussing проигравшего [e]Validation Block, теряют свои [f]Proposal Stake и [f]Vote Stake; данная сумма распределяется между [a]Validator, победившими в голосовании.

3.2.13. Терминальные состояния Validation Request

1. В рамках протокола блок хранится в виде 2 сущностей:
    1. [e]Validation Block - полная версия, в виде Protobuf-объекта.
    2. [s]BlockRef - урезанная версия, в виде структуры.
2. [s]BlockRef, будучи урезанной версией блока, не хранит всей информации о результатах проверки [e]Validation Request, а только его терминальное состояние.
3. В [s]BlockRef терминальное состояние хранится в поле uint256 result и может принимать 4 значения:
    1. 2 бита на каждый [e]Validation Request.
    2. До 128 [e]Validation Request в блоке.
4. Типы терминального состояния:
    1. 0b00 - UNAVAILABLE
    2. 0b01 - SUCCESS
    3. 0b10 - NONE, не используется, зарезервирован
    4. 0b11 - ERROR

3.2.14. Обработка UNAVAILABLE состояния

UNAVAILABLE статус необходим, чтобы протокол продолжал работать, в случае если один или несколько [n]Blockchain вышли из строя во время валидации.

Логика обработки UNAVAILABLE:

1. [a]Validator в рамках [p]Block Proposing может объявить ЛЮБОЙ [e]Validation Request как UNAVAILABLE; в этом случае [f]Validation Fee не возвращается [a]User, но и [a]Validator не получает его в качестве награды — он остается на балансе протокола.

    1. Данное поведение необходимо с точки зрения теории игр, чтобы ни одна сторона не могла это использовать для атаки на протокол.
2. [a]Validator в рамках [p]Block Discussing должен воспринимать UNAVAILABLE статусы исходного [e]Validation Block как не имеющие значимых различий.
    1. Значимое различие — различающиеся статусы, НЕ являющиеся UNAVAILABLE.
        1. UNAVAILABLE == SUCCESS/NONE/ERROR - незначимое различие.
        2. SUCCESS ≠ NONE - значимое различие.
3. [a]Validator в рамках [p]Block Voting так же должен воспринимать UNAVAILABLE статусы исходного [e]Validation Block как не имеющие значимых различий.
4. [a]Validator в рамках [p]Block Voting может прикрепить unavailabilityMask с типом uint128, в случае если исходный блок не имеет значимых различий.
    1. По 1 биту на каждый [e]Validation Request.
    2. Если большинство [a]Validator прислали unavailabilityMask с UNAVAILABLE статусами запросов, которые в изначальном блоке были помечены статусами НЕ UNAVAILABLE, они меняют свой статус на UNAVAILABLE.
    3. В таком случае [e]Validation Block и [s]BlockRef будут различаться; в данном случае source of truth выступает финальный (скорректированный) [s]BlockRef и [c]OpenStore. [e]Validation Block лишь отражает блок, на основании которого проходило голосование.
5. Если unavailabilityMask изменяет состояние, которое, в свою очередь, влияет на блок [p]Block Discussing так, что пропадают значимые различия с исходным блоком, — [f]Proposal Stake и [f]Vote Stake возвращаются своим [a]Validator.

    EXAMPLES!

    1. Допустим, в [s]BlockRef 2 результата со статусами SUCCESS и UNAVAILABLE.
        1. Если [a]Validator в ходе своей проверки получил результаты SUCCESS и SUCCESS, он должен воспринимать исходный блок как равный и инициировать [p]Block Voting.
        2. Если [a]Validator в ходе своей проверки получил результаты ERROR и SUCCESS, он должен воспринимать исходный блок как отличающийся и инициировать [p]Block Discussing со статусами ERROR и SUCCESS.
    2. Допустим, в [s]BlockRef 2 результата со статусами SUCCESS и SUCCESS.
        1. Если [a]Validator в ходе своей проверки получил результаты SUCCESS и UNAVAILABLE, он голосует за изначальный блок, прикладывая unavailabilityMask со значением 0b0000...0010 (второй запрос

UNAVAILABLE).

3. Допустим, в [s]BlockRef 2 результата со статусами SUCCESS и SUCCESS.
    1. У [s]BlockRef есть [p]Block Discussing со статусами SUCCESS и ERROR.
    2. Первый блок со статусами SUCCESS и SUCCESS победил в голосовании.
    3. Но большинство [a]Validator предоставили unavailabilityMask, так что второй статус превратился в UNAVAILABLE.
    4. Финальный блок стал иметь результаты SUCCESS и UNAVAILABLE.
    5. В таком случае второй блок со статусами SUCCESS и ERROR теряет значимое различие.
    6. [a]Validator, кто выдвинул и проголосовал за второй блок со статусами SUCCESS и ERROR, не будут наказаны и получат назад свой Stake.

3.2.15. Публикация Asset Artifact в OpenStore

Существует 3 способа публикации [s]BuildInfo в [c]OpenStore:

1. **Publication** — в данном случае [e]Asset Artifact обозначается опубликованным, но НЕ провалидированным; он будет доступен только при поиске по адресу [e]Asset.
2. **Раздельная [p]Artifact Validation и Publication** — можно сперва пройти [p]Artifact Validation, и в таком случае [e]Asset Artifact будет помечен как провалидированный и НЕ опубликованный, после чего его можно опубликовать в любое удобное время.
3. **Совместная [p]Artifact Validation и Publication** — при отправке на [p]Artifact Validation имеется возможность добавить опцию автопубликации, и в таком случае:
    1. При успешном прохождении [p]Artifact Validation в [c]OpenStore публикуемый [e]Asset Artifact будет автоматически помечен как провалидированный и опубликованный.
    2. Если [p]Artifact Validation завершился ошибкой, [e]Asset Artifact никак не помечается, то есть он будет НЕ провалидированный и НЕ опубликованный.

3.2.16. Настройка дистрибуции Asset

1. В обычной ситуации раздача [e]Asset Artifact происходит непосредственно из [e]Greenfield.
2. При необходимости [a]Publisher может указать одну или более [e]Distribution Link, используя свои серверы.
3. [a]Publisher при указании [e]Distribution Link может использовать параметры для

обобщения ссылок:

1. ${VERSION_CODE} - номер версии, указанный в [e]Asset Artifact (e.g. 132).
2. ${VERSION_NAME} - имя версии [e]Asset Artifact (e.g. 1.0.2-beta01).
3. ${REF_ID} - идентификатор ссылки [e]Asset Artifact в [e]Greenfield или ином протоколе (e.g. 0xFFAAWW).

3.2.17. Доступность Asset

1. В большинстве случаев [a]User получает данные из [n]API, которые синхронизируются посредством [n]Daemon из BSC, opBSC, Greenfield или иного блокчейна.
2. Доступность [e]Asset для пользователей через [n]API может быть ограничена, если:
    1. [e]Asset Artifact не прошел [p]Artifact Validation.
    2. [a]Publisher изменил параметры видимости [e]Asset в [u]Open Store Studio.
    3. Основной функциональностью [e]Asset является один или несколько пунктов, запрещенных на платформе (см. Приложение A).
3. Стоит учитывать, что любой [e]Asset всегда будет доступен при поиске по адресу, так как данные берутся напрямую из блокчейна; в данном случае [u]Open Store App выступает в качестве Explorer.
    1. За исключением случаев, когда [a]Publisher изменил параметры видимости в [c]OpenStore; в таком случае видимость ограничивается самим владельцем на уровне контракта.

3.2.18. Установка Asset Artifact через Open Store App

1. В [u]Open Store App существует 3 способа найти приложение:
    1. **Каталог** ([n]API) - [e]Asset, прошедшие [p]Ownership Verification и [p]Artifact Validation.
    2. **Поиск по имени** ([n]API) - [e]Asset, прошедшие [p]Ownership Verification и [p]Artifact Validation.
    3. **Поиск по адресу** ([n]Blockchain) - [e]Asset, опубликованные в [c]OpenStore; [p]Ownership Verification и [p]Artifact Validation необязательны, все данные берутся напрямую из блокчейна.
2. Данные для установки:
    1. Общая информация и актуальная версия [e]Asset берутся из:
        1. [n]API в случае перехода из каталога или поиска по имени.
        2. [n]Blockchain в случае поиска по адресу.
    2. Ссылка на скачивание актуальной версии [e]Asset Artifact получается всегда строго из [n]Blockchain.

3. Перед установкой [u]Open Store App предварительно выполняет валидацию [e]Asset Artifact, которая включает:
    1. Compare actual file checksum with struct BuildInfo checksum.
    2. Compare actual file certificates fingerprints with certificates fingerprints from [c]AppOwnerPluginV1.
    3. Validate certificates proofs from [c]AppOwnerPluginV1 with public key from actual file.

3.2.19. Обновление Asset через Open Store App

1. Новые версии [e]Asset Artifact устанавливаются в ручном режиме; при необходимости пользователь может разрешить автоматическое обновление.
2. [c]OpenStore может содержать несколько приложений с одинаковым identifier (packageName), и в таком случае в роли distinct identifier будет выступать адрес [c]App.

Приложение А.

Inhumane list

Если основной функцией приложения или файла является один из ниже перечисленных пунктов, они скрываются из общего поиска и будут доступны только при поиске по адресу.

Продажа запрещенных товаров и веществ

- Тяжелые наркотики
- Поддельные документы и валюта
- Краденые товары и имущество

Киберпреступность и вредоносная деятельность

- Вредоносное и шпионское программное обеспечение (Malware)
- Хакерские услуги и инструменты для взлома
- Кардинг и кража финансовых учетных данных

Финансовые преступления и мошенничество

- Отмывание денег
- Мошенничество и аферы
- Продажа украденных данных

Насилие, экстремизм и эксплуатация

- Материалы с сексуальным насилием над детьми (CSAM)
- Торговля людьми и их эксплуатация
- Заказные убийства
- Терроризм

Нарушение неприкосновенности частной жизни и \ конфиденциальности

- Продажа личной информации (доксинг)
- Незаконная слежка и прослушивание

Приложение B.

Publisher components:

- Asset Contracts
  - DevAccount
  - App
- Store Contracts
  - Oracle
  - OpenStore
- UI
  - Open Store Studio
- Tools
  - Proof Generator CLI
  - Publishing CLI

Validator/Oracle components:

- Store Contracts
  - Oracle
  - OpenStore
- Node
  - Oracle
  - Validator

Broadcaster components:

- Node
  - API
  - Sync Daemon
  - Stat API (Optional)

User components:

- Open Store App