DOI 10.37882/2223-2966.2025.08.03

ОТ АТОМАРНОГО ДОВЕРИЯ К РАСПРЕДЕЛЕННОМУ КОНСЕНСУСУ: СРАВНИТЕЛЬНЫЙ АНАЛИЗ АРХИТЕКТУР ДОВЕРЕННОЙ ТРЕТЬЕЙ СТОРОНЫ НА БЛОКЧЕЙН-ПЛАТФОРМЕ ETHEREUM

FROM ATOMIC TRUST TO DISTRIBUTED CONSENSUS: A COMPARATIVE ANALYSIS OF TRUSTED THIRD-PARTY ARCHITECTURES ON THE ETHEREUM BLOCKCHAIN PLATFORM

N. Beksaev

Summary. A trusted third party (TTP) is a fundamental element for ensuring secure digital interactions. The traditional model based on centralized certification authorities (CA) faces problems with scalability and a single point of failure. Blockchain and smart contract technologies provide an opportunity to think about decentralized alternatives. In this article, the author aims to demonstrate the fundamental possibility of creating such an alternative and to provide an economic assessment of the use of each architecture. A comparative analysis of three TTP implementation architectures is carried out: a classic CA, a fully distributed TTP using the Ethereum blockchain as an example, and a hybrid model that combines both approaches. The minimum required smart contracts for implementing alternative architectures are determined, and tasks are distributed between the components of the subsystems. Oracles are considered as technologies for connecting the blockchain with the outside world and the risks associated with them. The author conducted practical experiments to evaluate the metric of computational efforts to perform key operations (the so-called «gas»), which made it possible to implement an assessment of the cost of signing one document; such data can be used for calculations when designing distributed TTP systems in the future.

Keywords: trusted third party, blockchain, smart contact, blockchain oracle, certification authority.

Бексаев Николай Сергеевич
Аспирант,
Петербургский государственный университет путей сообщения Императора Александра I
n.beksaev@yandex.ru

Аннотация. Доверенная третья сторона (далее ДТС) является фундаментальным элементом для обеспечения безопасных цифровых взаимодействий. Традиционная модель, основанная на централизованных удостоверяющих центрах (далее УЦ), сталкивается с проблемами масштабируемости и единой точки отказа. Технологии блокчейн и смарт-контрактов дают возможность задуматься о децентрализованных альтернативах. В статье автор ставит цель показать принципиальную возможность создания такой альтернативы и дать экономическую оценку использования каждой из архитектур. Проводится сравнительный анализ трех архитектур реализации ДТС: классического УЦ, полностью распределенной ДТС на примере блокчейна Ethereum и гибридной модели, в которой сочетаются оба подхода. Определены минимально необходимые смарт-контракты для реализации альтернативных архитектур и выполнено распределение задач между компонентами подсистем. Рассматриваются оракулы как технологии для связи блокчейна с внешним миром и риски с ними связанные. Автором проведены практические эксперименты для оценки метрики вычислительных усилий для выполнения ключевых операций (т.н. «газа»), что позволило реализовать оценку стоимости подписания одного документа, такие данные могут быть использованы для расчетов при проектировании распределенных систем ДТС в дальнейшем.

Ключевые слова: доверенная третья сторона, блокчейн, смарт-контакт, блокчейн оракул, удостоверяющий центр.

Введение

В цифровой экономике потребность в надежном посреднике, который может подтвердить подлинность сторон и целостность данных, неоспорима. Эту роль традиционно выполняет доверенная третья сторона (ДТС) — чаще всего в виде иерархической системы удостоверяющих центров (УЦ), лежащей в основе инфраструктуры открытых ключей (PKI). Архитектура такой системы (далее Архитектура 1) характеризуется централизованным управлением [5], где стоимость услуг определяется договорными отношениями.

С появлением технологии распределенного реестра, в частности блокчейна Ethereum, возникли новые парадигмы реализации ДТС [2]. Блокчейн предлагает децентрализацию, прозрачность и неизменяемость записей, устраняя необходимость в едином центре доверия [1]. В частности, «Мастерчейн» — это российская национальная блокчейн-платформа, созданная в 2016 году Ассоциацией «ФинТех» совместно с Банком России. Она основана на модифицированной версии протокола Ethereum, адаптированной под требования российского законодательства [4], включая криптографию и процесс идентификации пользователей. Основные принци-

пы платформы [3] включают юридическую значимость информации, сертификацию ФСБ России, поддержку смарт-контрактов, отсутствие единой точки отказа и возможность масштабирования по числу участников и транзакций. Это позволяет взглянуть на проблему построения единого пространства доверия по-новому. И с этой целью рассматриваются две альтернативный модели архитектур с ДТС:

- Полностью распределенная ДТС (далее Архитектура 2). В этой модели все функции ДТС, включая управление идентификацией и верификацию документов, реализуются исключительно с помощью смарт-контрактов в сети Ethereum. Доверие переносится с централизованного органа на криптографически защищенный и экономически стимулированный консенсус сети [11].
- Гибридная ДТС (далее Архитектура 3). Эта модель сочетает надежность и юридическую признанность традиционного УЦ с прозрачностью и отказоустойчивостью блокчейна. УЦ отвечает за первичную верификацию и выдачу идентификационных данных, а блокчейн используется как публичный, неизменяемый реестр для регистрации событий, таких как подписание документов и отзыв сертификатов.

Проектирование смарт-контрактов и минимизация данных

Основной принцип проектирования — минимизация данных, хранимых в блокчейне, также называемое в профессиональной литературе «on-chain». Хранение в Ethereum является дорогостоящей операцией, такая операция называется «SSTORE», поэтому в блокчейн следует записывать только криптографические хэши документов и минимально необходимую метаинформацию: адреса, временные метки. Сами документы должны храниться во внешних системах, по аналогии называемой «off-chain». Для создания программного кода смарт-контрактов используется язык программирования Solidity [7].

1. Смарт-контракты для Архитектуры 2

Для этой архитектуры потребуются два контракта.

1. **IdentityRegistry.sol:** Контракт для управления цифровыми идентификаторами. Каждый пользователь может зарегистрировать свой адрес Ethereum как идентификатор, с которым будут ассоциироваться дальнейшие действия.
2. **DocumentNotary.sol:** Контракт для заверения документов. Он позволяет зарегистрированным пользователям «подписывать» документы путем записи хэша документа в блокчейн.

Минимально необходимые данные (on-chain):

1. В **IdentityRegistry:** mapping(address=>bool) для отслеживания зарегистрированных пользователей.
2. В **DocumentNotary:** mapping (bytes32 => SignatureRecord) для хранения информации о подписи, где SignatureRecord — это структура, содержащая адрес подписанта и временную метку (struct SignatureRecord { address signer; uint256 timestamp; }).

2. Смарт-контракты для Архитектуры 3

В гибридной модели УЦ сохраняет за собой функцию контроля доступа.

1. **HybridIdentity.sol:** Контракт управляется УЦ. Только адрес, принадлежащий УЦ, может добавлять и отзывать верифицированных пользователей.
2. **HybridNotary.sol:** Аналогичен DocumentNotary, но проверяет, верифицирован ли пользователь в контракте HybridIdentity.

Минимально необходимые данные (on-chain):

1. В HybridIdentity: Адрес владельца-УЦ (address private _caAuthority), mapping(address => bool) для отслеживания верифицированных пользователей.
2. В HybridNotary: Аналогично DocumentNotary.

Исходный код Смарт-контрактов

Исходный код смарт-контрактов для Архитектуры 2:

// SPDX-License-Identifier: MIT pragma solidity ^0.8.20; /*** @title IdentityRegistry * @dev Управляет идентификацией пользователей полностью децентрализованным образом. */ contract IdentityRegistry {mapping(address => bool) private _registeredIdentities; event IdentityRegistered(address indexed identity); function registerIdentity() public {require(!_registeredIdentities[msg.sender], «Identity already registered.»); _registeredIdentities[msg.sender] = true; emit IdentityRegistered(msg.sender);} function isRegistered(address identity) public view returns (bool) {return _registeredIdentities[identity];}}/*** @title DocumentNotary * @dev Позволяет зарегистрированным лицам нотариально заверять хэши документов. */contract DocumentNotary { struct SignatureRecord {address signer; uint256 timestamp;} IdentityRegistry private _identityRegistry; mapping(bytes32 => SignatureRecord) private _signatures; event DocumentSigned(bytes32 indexed documentHash, address indexed signer, uint256 timestamp); constructor(address registryAddress) { _identityRegistry = IdentityRegistry(registryAddress);} function signDocument(bytes32 documentHash) public {require(_identityRegistry.isRegistered(msg.

sender), «Signer identity is not registered.»); require(_signatures[documentHash].signer == address(0), «Document already signed.»); _signatures[documentHash] = SignatureRecord(msg.sender, block.timestamp); emit DocumentSigned(documentHash, msg.sender, block.timestamp);}function getSignature(bytes32 documentHash) public view returns (address, uint256) { SignatureRecord storage record = _signatures[documentHash]; return (record.signer, record.timestamp);}}

Исходный код для Архитектуры 3

// SPDX-License-Identifier: MITpragma solidity ^0.8.20;/*** @title HybridIdentity * @dev Управляет идентификацией, подтвержденной центральным центром сертификации (CA). */contract HybridIdentity {address private _caAuthority; mapping(address => bool) private _verifiedUsers; event UserVerified (address indexed user); event UserRevoked (address indexed user); modifier onlyCA() {require(msg.sender == _caAuthority, «Такое действие может выполнить только ЦСС.»); _;} constructor() {_caAuthority = msg.sender; // Контракт загружает ЦСС} function verifyUser(address user) public onlyCA {_verifiedUsers[user] = true; emit UserVerified(user);} function revokeUser(address user) public onlyCA {_verifiedUsers[user] = false; emit UserRevoked(user);} function isVerified(address user) public view returns (bool) {return _verifiedUsers[user];}}/*** @title HybridNotary * @dev Нотариально заверяет документы для пользователей, проверенных УЦ. */contract HybridNotary {struct SignatureRecord { address signer; uint256 timestamp;} HybridIdentity private _identityContract; mapping(bytes32 => SignatureRecord) private _signatures; event DocumentSigned(bytes32 indexed documentHash, address indexed signer, uint256 timestamp); constructor(address identityContractAddress) {_identityContract = HybridIdentity(identityContractAddress);} function signDocument(bytes32 documentHash) public { require(_identityContract.isVerified(msg.sender), «User is not verified by CA.»); require(_signatures[documentHash].signer == address(0), «Document already signed.»); _signatures[documentHash] = SignatureRecord(msg.sender, block.timestamp); emit DocumentSigned(documentHash, msg.sender, block.timestamp);} function getSignature(bytes32 documentHash) public view returns (address, uint256) {SignatureRecord storage record = _signatures[documentHash]; return (record.signer, record.timestamp); }}

Разделение задач в архитектурах ДТС

Для полного понимания различий между моделями необходимо четко разграничить, какой компонент системы выполняет каждую ключевую операцию ДТС (табл. 1).

Результат эксперимента по потреблению вычислительных ресурсов

Оценка расхода вычислительных ресурсов или «газа» является ключевым показателем экономической эффективности блокчейн-систем. Стоимость транзакции рассчитывается как $gas_{used} * gas_{price}$, где gas_{used} — количество вычислительных ресурсов («газа»), затраченных на операцию, а gas_{price} — цена единицы газа используемого в блокчейне. Автором был проведен практический эксперимент и получен расход gas_{used} для основных операций для блокчейна Ethereum (на июль 2025 г.). Результаты представлены в таблице 2.

Таблица 1.

Операции доверенной третьей стороны

Операция ДТС	Архитектура 1 (Классический УЦ)	Архитектура 2 (Полностью распределенная)	Архитектура 3 (Гибридная)
Идентификация и проверка личности	УЦ (Off-chain)	Вне системы (используется псевдонимность адреса)	УЦ (Off-chain)
Выпуск учетных данных	УЦ	Пользователь + Блокчейн (самостоятельная регистрация)	УЦ (инициирует запись о верификации в Блокчейн)
Регистрация события подписания	УЦ	Блокчейн (Смарт-контракт)	Блокчейн (Смарт-контракт)
Проверка статуса подписи	УЦ	Блокчейн (Смарт-контракт)	Блокчейн (Смарт-контракт)
Отзыв учетных данных	УЦ	Блокчейн (через логику контракта)	УЦ + Блокчейн (УЦ инициирует транзакцию отзыва)
Обеспечение неизменности реестра	УЦ (доверие к его базам данных)	Блокчейн (Консенсус сети)	Блокчейн (Консенсус сети)
Обеспечение доступности сервиса	УЦ (его серверная инфраструктура)	Блокчейн (децентрализованная сеть)	Гибридная: УЦ (для верификации), Блокчейн (для реестра)

Таблица 2.

Подсчета расхода газа для операций ДТС при использовании смарт-контрактов

Операция	Архитектура 2 (Полностью распределенная)	Архитектура 3 (Гибридная)	Количество вычислительных единиц (gasused)	Основные затратные операции
Развертывание контракта Identity	IdentityRegistry	HybridIdentity	95192	Код контракта
Развертывание контракта Notary	DocumentNotary	HybridNotary	173000	Код контракта, установка адреса
Регистрация/Верификация пользователя	registerIdentity ()	verifyUser ()	47000	SSTORE (запись в mapping)
Подписание документа	signDocument ()	signDocument ()	39900	SLOAD (проверка), SSTORE (запись)
Проверка подписи	getSignature()	getSignature()	5	SLOAD (чтение из mapping)

Подсчет расхода газа для операции «Подписание документа»

Процесс подписания документа в смарт-контракте имеет следующие этапы:

1. Проверка регистрации/верификации (require): Эта операция включает чтение из хранилища (SLOAD), что стоило 5100 единиц газа. В гибридной модели происходит внешний вызов (CALL) к контракту идентификации, что добавило 1700 единиц газа.
2. Проверка, что документ еще не подписан (require): Еще одно чтение из хранилища (SLOAD), еще 5100 единиц газа.
3. Запись подписи (_signatures[documentHash] = ...): Это основная статья расходов. Запись нового значения в хранилище (SSTORE) стоила около 20000 единиц газа. Также на этом шаге происходит запись block.timestamp.
4. Генерация события (emit): Стоимость зависит от количества индексируемых полей и объема данных, в рассматриваемом в статье случае было затрачено 8000 единиц газа.

Итоговое значение операции signDocument в единицах gas_{used} в обеих архитектурах отличалась на 1700 единиц и составила 38200 единиц газа для Архитектуры 3 и 39900 единиц для Архитектуры 2. Основные затраты приходятся на запись данных в блокчейн, а не на логику проверки.

Итоговая стоимость описанной выше транзакции для Архитектур 2 и 3 в криптовалюте рассчитывается по формуле:

$$Price_{singDocument} = gas_{used} * gas_{price} * Eth_{price},$$

где gas_{used} — количество единиц вычислительной мощности, затраченных на операцию; gas_{price} — стоимость вычислительной мощности в криптовалюте блокчейна Ethereum, которая в июле равняется $1{,}275 * 10^{-9}$; Eth_{price} — стоимость 1 монеты в блокчейне Ethereum в долларах США.

Расчет стоимости операции на примере Архитектуры 3 в долларах США будет выглядеть следующим образом (с округлением до сотых):

$$Price_{singDocument} = 39900 * 1{,}275 * 10^{-9} * 3864{,}12 = 0{,}20.$$

Таким образом, стоимость одной подписи составила, около 0,2 долларов США или около 16 рублей. Также важно подчеркнуть, что методика расчета едина для любых других блокчейнов, построенных на технологии Ethereum, и удешевления операции можно добиться путем использования других блокчейнов на базе Ethereum, в том числе корпоративных.

Роль и риски оракулов в блокчейнах

Смарт-контракты в Ethereum по своей природе изолированы от внешнего мира. Они не могут самостоятельно получать данные из интернета (например, курсы валют, результаты спортивных матчей или данные из традиционных баз данных). Для преодоления этого ограничения и связи с оффчейн-системами используются специальные сущности — оракулы [12]. В контексте наших архитектур ДТС оракулы становятся критически важным компонентом.

Типы оракулов

Оракулы можно условно разделить на две основные категории:

1. **Централизованные оракулы.** Это единственный поставщик данных, который контролируется одной организацией. Такой оракул быстр и прост в реализации, но он вновь вводит централизован-

ную точку отказа и доверия. Если оракул будет скомпрометирован или прекратит работу, вся система, зависящая от него, окажется под угрозой.

2. **Децентрализованные оракулы.** Представляют собой сеть независимых узлов, которые собирают и верифицируют данные из внешних источников. Используя механизмы консенсуса и экономические стимулы (стейкинг, штрафы), такие сети (например, Chainlink [8]) стремятся обеспечить достоверность и доступность данных, минимизируя риски манипуляции.

Задачи оракулов в системах ДТС

В архитектурах 2 и 3 оракул решает следующие задачи:

1. Верификация оффчейн-данных: В Архитектуре 3, удостоверяющий центр выступает в роли централизованного, доверенного оракула. Когда УЦ вызывает функцию verifyUser(), он поставляет в блокчейн данные (факт верификации), полученные и проверенные в реальном мире (проверка документов, биометрии и т.д.).
2. Связь с хранилищами документов: Хотя в нашей базовой модели это не реализовано, более сложная система могла бы использовать оракул для проверки доступности документа в IPFS[X] перед заверением его хэша.
3. Активация смарт-контрактов по внешним событиям: Система ДТС может быть частью более сложного процесса. Например, смарт-контракт условного депонирования (эскроу) может использовать нотариальный контракт [6] для подписи акта выполненных работ. Оракул в такой системе мог бы сообщить контракту о физической доставке товара, что послужило бы триггером для запроса цифровой подписи.

Риски и «Проблема Оракула»

Главный риск, связанный с использованием оракулов, известен как «проблема оракула»[9]. Она заключается в том, что безопасность и надежность детерминированного смарт-контракта становится зависимой от качества внешних, не детерминированных данных. Если оракул предоставит неверную информацию (случайно или умышленно), смарт-контракт исполнит ее как истинную, что может привести к необратимым финансовым потерям [10].

- В Архитектуре 3 этот риск полностью ложится на доверие к УЦ. Если УЦ будет скомпрометирован и верифицирует злоумышленника, блокчейн-часть системы не сможет это распознать.
- В Архитектуре 2 при использовании оракулов для расширения функционала (например, для связи с реальной личностью) необходимо выбирать децентрализованные сети оракулов, что усложняет систему и вводит дополнительные транзакционные издержки на их услуги.

Сравнение архитектур по основным критериям

Сравним три рассматриваемых архитектуры по следующим критериям, представленным в таблице 3.

Заключение и выводы

При низкой цене вычислительной мощности блокчейн-решения могут становиться конкурентоспособными по сравнению с некоторыми услугами традиционных УЦ, особенно для автоматизированных систем с большим объемом документов. К этой сумме может добавляться стоимость услуг оракулов. Автором приведены формулы расчета стоимости подписания документов, реализован набор смарт-контрактов для этих задач

Таблица 3.

Сравнение архитектур по критериям

Критерий	Архитектура 1 (Классический УЦ)	Архитектура 2 (Полностью распределенная)	Архитектура 3 (Гибридная)
Доверие	Централизованное (к УЦ)	Децентрализованное (к коду и сети)	Гибридное (к УЦ за верификацию, к сети за реестр)
Точка отказа	Единая точка отказа	Отсутствует	УЦ является точкой отказа для новых верификаций
Прозрачность	Непрозрачная	Полностью прозрачная	Прозрачный реестр действий
Зависимость от оракула	Отсутствует (сама является оракулом для других)	Появляется при необходимости связи с реальным миром	Критическая. УЦ выступает как доверенный оракул
Стоимость подписи	Фиксированная по договору	Переменная (зависит от параметра gas_price)	Переменная (зависит от параметра gas_price)
Контроль	Полный контроль у УЦ	Контроль у пользователей	Разделенный контроль

и проведен эксперимент, которых показал принципиальную возможность и стоимость такой операции.

Минимизация данных — ключ к эффективности: Хранение только хэшей и метаданных является единственным рабочим подходом для создания экономически оправданной системы нотариального заверения на блокчейне.

Появление оракулов — это компромисс. Блокчейн-архитектуры не могут существовать без внешних по отношению к ним информационных систем. Архитектура 3 явно признает это, инкапсулируя доверие к внешнему миру в УЦ, который действует как оракул. Архитектура 2 в своей чистой форме избегает этого, но теряет в возможности привязки к юридически значимой идентичности, а при попытке добавить такую связь неизбежно сталкивается с необходимостью внедрения оракулов и связанными с ними рисками.

Выбор архитектуры — это выбор модели доверия. Архитектура 2 подходит для сообществ и систем, где нет единого центра власти, и требуется максимальная открытость, Архитектура 3 является прагматичным компромиссом для организаций (например, государственных органов, корпораций), которые должны соответствовать регуляторным требованиям и сохранять контроль над процессом идентификации, но при этом хотят воспользоваться преимуществами блокчейна — неизменяемостью и аудируемостью реестра. Этот подход снижает риски «проблемы оракула» до уровня доверия конкретному, юридически ответственному УЦ.

Таким образоам, блокчейн Ethereum предоставляет мощные и гибкие инструменты для построения систем ДТС нового поколения. Хотя они и не заменят полностью традиционные УЦ в ближайшем будущем, гибридные модели уже сегодня предлагают значительные преимущества, а полностью распределенные системы открывают путь к созданию глобальных, не требующих доверия цифровых юрисдикций.

ЛИТЕРАТУРА

1. Данькив В.М. Технологии блокчейн за пределами криптовалют. Актуальные вопросы научных исследований в условиях формирования многополярного мира. Сборник статей Всероссийской научно-практической конференции с международным участием. Уфа, 2025. С. 23–27.
2. Исроилов С.Г., Верзунов С.Н. Разработка защищенной системы электронного документооборота на основе блокчейн-технологии // Проблемы автоматики и управления (ISSN: 1694–5050), 2021. №2(41).
3. Петренко А.С., Петренко С.А., Костюков А.Д. Эталонная модель блокчейн-платформы // Защита информации. Инсайд, 2022. №4(106). С. 34–44.
4. Петренко С.А. Киберустойчивость цифровой экономики: научно-популярная монография.
5. Станкевич Т.Л. Методики синтеза системы защиты информации и повышения производительности службы доверенной третьей стороны при трансграничном электронном взаимодействии. Автореферат диссертации, 2016.
6. Рисовская С.С. Нотариальный блокчейн: роль нотариуса как доверенной третьей стороны. International Law Journal, 2023. P. 204–210
7. Ямковой Н.С. Блокчейн технологии: разработка узла блокчейн. // ТОГУ-Старт: фундаментальные и прикладные исследования молодых, Материалы региональной научно-практической конференции.2022. С. 294–303.
8. Breidenbach, Lorenz, Christian Cachin, Benedict Chan, Alex Coventry, Steve Ellis, Ari Juels, and Farinaz Koushanfar. «Chainlink 2.0: Next Steps in the Evolution of Decentralized Oracle Networks.» 2021.
9. Caldarelli, Giulio, and Joshua Ellul. «The Blockchain Oracle Problem in Decentralized Finance — A Multivocal, Approach.» Applied Sciences, 2021. 11(16), 7572.
10. Suarez Barcia, Lucia «Decentralized Finance Oracles,» Journal of New Finance: Vol. 3: No. 1, Article 2. 2023. P. 6–7.
11. Terentyev D.E. Blockchain as a tool of legal transparency: prospects and challenges in the legal regulation of the digital economy. Theory and practice of modern science: the view of youth. Proceeding of the III all-russian scientific and practical conference in english. In 2 parts. Saint-petersburg, 2024. P. 27–30
12. Yinjie Zhao, Kang Xin, Tieyan Li, Cheng-Kang Chu, and Haiguang Wang. «Towards Trustworthy DeFi Oracles:Past, Present and Future.» IEEE Access. 2022.

Журнал «Современная наука: актуальные проблемы теории и практики»