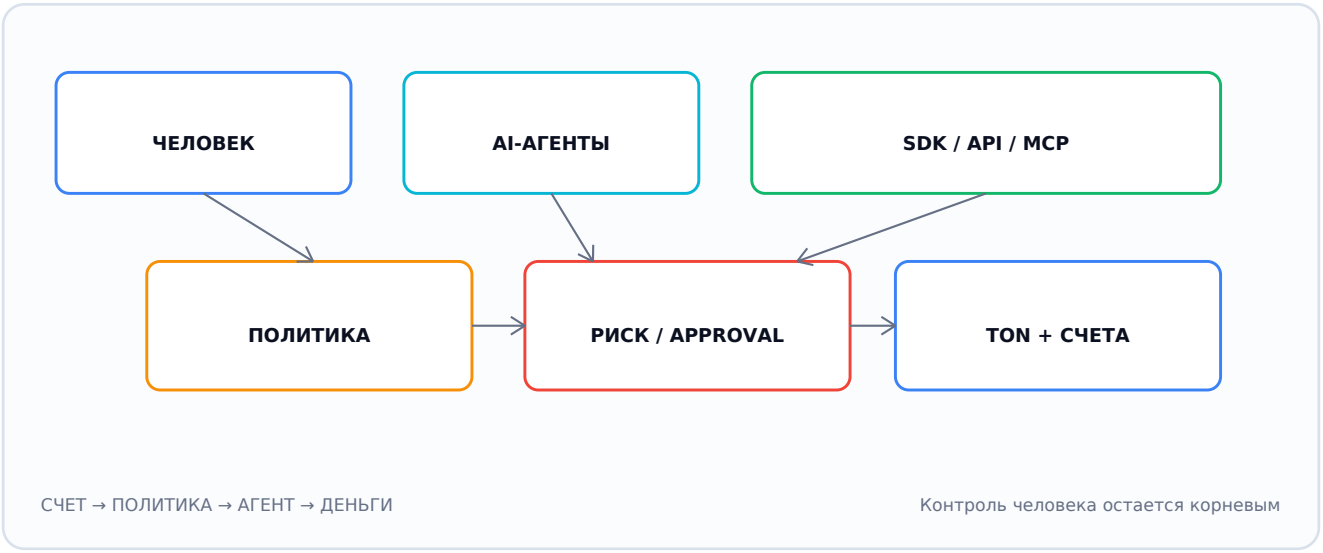
TONBANKCARD

WHITEPAPER 2.0

Программируемый финансовый слой счетов для TON

Банковский интерфейс для человека + программируемые финансовые счета для AI-агентов и приложений.



Главная идея

Финансовый счет должен быть не просто адресом и не просто экраном с балансом. Это программируемый объект с явным владельцем, правами доступа, политиками, правилами исполнения и проверяемой историей операций.

Статус документа: архитектурный и продуктовый Whitepaper 2.0. Компоненты разделяются на существующие, находящиеся в разработке и предлагаемые. Документ не означает, что все описанные модули уже развернуты в mainnet.

Тезис на одной странице

TONBANKCARD развивается от некостодиального протокола счетов к программируемому финансовому слою для людей, приложений и AI-агентов.

Новая архитектура объединяет NFT-модель счетов на TON с контролем полномочий, политиками, платежными и расчетными примитивами, SDK/API и агентским слоем, совместимым с MCP. Задача не в том, чтобы дать AI право владеть деньгами. Задача в том, чтобы владелец счета определял, что агенту разрешено делать, а система обеспечивала соблюдение этих правил.

СЧЕТ	Устойчивый финансовый объект, связанный с владельцем и поддерживаемыми активами.
ПОЛИТИКА	Явные лимиты, права, условия, approvals и правила риска.
АГЕНТ	Делегированный исполнитель, работающий только в пределах полномочий.
ДЕНЬГИ	TON, jettons, NFT и поддерживаемые финансовые операции.

Архитектурное правило: человек остается источником полномочий, AI является делегированным исполнителем.

Это разделяет собственность, состояние счета, авторизацию, исполнение и интерфейс. Благодаря этому один и тот же счет может использоваться человеком через банковский UI, приложением через SDK/API и агентом через MCP.

Содержание

1. Аннотация

TONBANKCARD — некостодиальный финансовый протокол, построенный вокруг блокчейна TON. Его исходная архитектурная идея: NFT может представлять уникальный финансовый счет, а движение активов и операции контролируются владельцем через блокчейн.

Whitepaper 2.0 расширяет эту модель. Счет становится программируемым финансовым объектом, который может безопасно использоваться приложениями и AI-агентами. Агент может читать баланс, формировать платежный intent, выполнять разрешенные операции и вести аудируемый журнал действий, но не получает права владельца.

2. Резюме

Следующее поколение цифровых финансов будет управляться не только людьми. Программные агенты смогут сверять счета, контролировать treasury, обрабатывать инвойсы, выполнять подписки и взаимодействовать с финансовыми протоколами. Обычный blockchain-адрес не выражает достаточно контекста: кто может действовать, что именно разрешено, при каких условиях нужна подпись и как контролировать риск.

TONBANKCARD предлагает слой Programmable Financial Account. TON остается слоем расчетов и проверяемого состояния, а TONBANKCARD добавляет модель счета, права, политики, платежные intent-ы и финансовые модули.

• Human-first: привычный банковский UX остается главным интерфейсом.

• Non-custodial: пользователь сохраняет контроль над необходимой авторизацией.

• Policy-driven: полномочия фиксируются явно.

• Agent-ready: AI работает как ограниченный делегат.

• Composable: платежи, DEX, lending, recurring payments и multisig используют общие семантики.

• Auditable: intent, решение авторизации и execution должны быть связаны там, где это технически возможно.

3. Проблема

Криптокошелек отлично решает задачу хранения ключей и подписи транзакций. Но финансовому счету нужен дополнительный уровень абстракции: лимиты, роли, подписки, approved counterparties, инвойсы, approvals и правила treasury.

Поэтому нужен не еще один wallet UI, а control plane для финансового счета, соединяющий владение, политику, идентичность агента, риск и исполнение.

• Кошелек показывает адрес; счет показывает финансовые возможности.

• Транзакция фиксирует факт; intent описывает намерение.

• Ключ доказывает право подписи; policy ограничивает делегированное право.

• AI может автоматизировать исполнение, но не должен незаметно получать полный контроль.

• Один счет должен иметь единый semantic layer для UI, приложений и агентов.

4. Принципы дизайна

• Self-custody by default. Пользовательский контроль и подпись остаются базовыми.

• On-chain truth. Критические факты о состоянии и расчетах должны проверяться на TON.

• Least privilege. Делегату выдаются только необходимые полномочия.

• Human approval. Высокорисковые действия могут требовать явного подтверждения.

• Deterministic policy. Правила авторизации должны быть предсказуемыми.

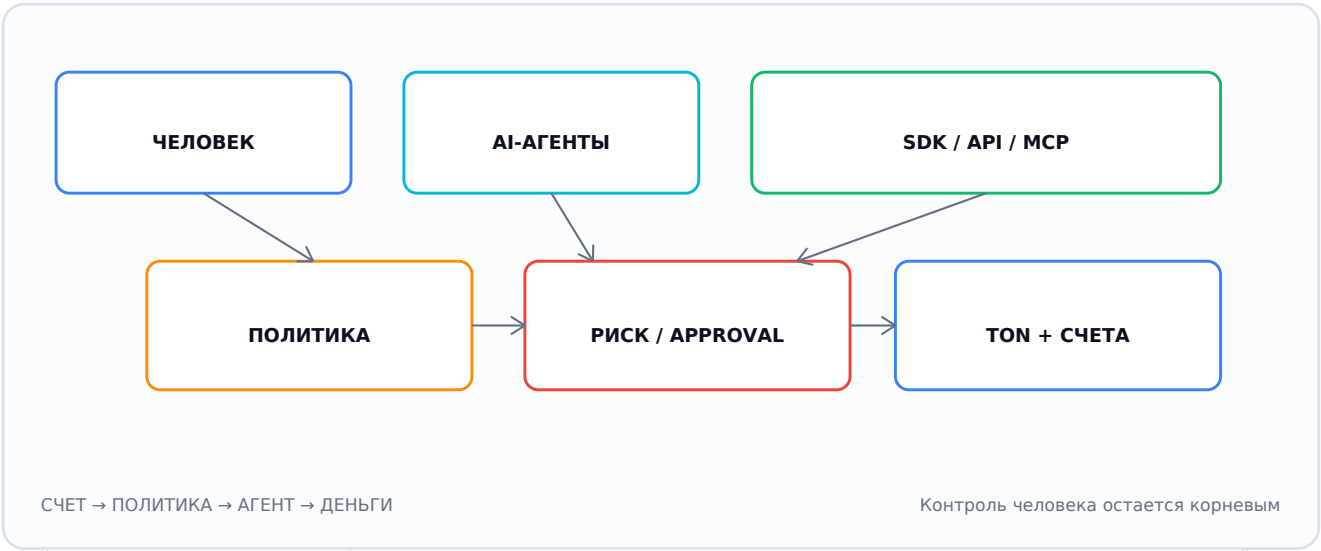
• Composable modules. Финансовые функции являются модульными и версионируемыми.

• Observable execution. Intent, решение и settlement должны быть связуемыми.

• Fail closed. Неоднозначная авторизация не должна превращаться в право тратить.

5. Архитектура протокола

Архитектура разделена на несколько уровней. TON обеспечивает blockchain settlement. TONBANKCARD определяет semantics финансового счета и модулей. SDK/API обеспечивают программный доступ. MCP выступает в качестве стандартизированного интерфейса взаимодействия для совместимых AI-агентов.



Слой	Ответственность
TON	Расчеты, сообщения, smart contracts, токены и on-chain state.
TBC Account Layer	Идентичность счета, account state, NFT-представление, ownership и операции.
Policy & Authorization	Права, лимиты, условия, роли, approvals и delegation.
Financial Modules	Payments, merchant, recurring payments, DEX, lending, multisig, gateways.
Developer Layer	SDK, API, indexer/read services и integration contracts.

Agent Layer	TBC Agent Protocol / MCP для machine-readable финансового взаимодействия.
Human UX	Web/mobile banking UI, approvals, activity и policy management.

6. Модель счета

В архитектуре TONBANKCARD NFT может представлять уникальный финансовый счет. Это создает устойчивую идентичность вокруг счета вместо того, чтобы рассматривать каждую транзакцию как отдельное событие.

Состояние счета включает как минимум owner/control authority, поддерживаемые активы, capabilities, policy configuration, делегатов, pending approvals, operational status и историю.

Карточный интерфейс является одной из форм представления счета. Тот же счет может обслуживать платежи, treasury, подписки, DEX и агентские workflows.

• Account identity — стабильный идентификатор.

• Ownership — корневая власть над счетом.

• Capabilities — поддерживаемые операции.

• Policy — ограничения для делегатов.

• State — балансы, лимиты, pending actions.

• History — операции и связанные intent/decision.

7. Владение, контроль и некостодиальность

Некостодиальность — архитектурное ограничение. Пользователь должен сохранять контроль над полномочием, необходимым для движения активов. Пользовательский интерфейс не должен хранить приватные ключи.

TON Connect позволяет использовать wallet как отдельный слой подписи: приложение формирует запрос, кошелек показывает его пользователю, а подпись выполняется кошельком.

Та же модель применяется к агентам: агент — делегированный оператор с ограниченным набором действий, а не новый владелец счета.

8. Активы и расчеты

Счет должен работать с TON-native активами и поддерживаемыми токенами. Для Jetton-интеграций приложения обязаны корректно учитывать decimals, master-контракт и wallet-контракты владельца.

Внутренние расчеты протокола могут использовать TBC там, где это определено развернутыми модулями. Поддержка каждого актива и операции должна быть capability-driven: клиент сначала выясняет, что именно поддерживает счет.

9. Платежи и merchant layer

Merchant payment превращает account abstraction в понятный commerce workflow. Merchant формирует payment intent с recipient, asset, amount, reference, сроком действия и дополнительными условиями.

Intent проходит policy/risk-проверку. При авторизации создается blockchain execution. Merchant API должен поддерживать уникальные идентификаторы, защиту от replay, idempotency и прозрачные статусы.

10. DEX, ликвидность и внешние шлюзы

DEX и внешние шлюзы рассматриваются как модульные адаптеры. Account layer может управлять swap-операциями через те же policies и ограничения.

Swap должен учитывать slippage, срок действия quote, лимиты активов и суммы. Внешний шлюз создает дополнительную trust boundary, поэтому его условия и комиссии должны быть явными.

Интеграция никогда не должна неявно расширять права счета.

11. Рекуррентные платежи и multisig

Recurring payments естественно моделируются как mandates/policies: merchant, максимальная сумма, периодичность, срок действия и условия отмены.

Multisig переносит тот же подход на командные и корпоративные счета. M-of-N подпись может разделять treasury control, operational roles и recovery. Агентские полномочия находятся ниже multisig authority и не заменяют ее.

12. Кредитование и collateral

Lending-модули могут использовать данные счета и сигналы collateral, не превращая сам account layer в кредитора или кастодиана.

Collateral signals и public lookup следует рассматривать как данные для политики или интерфейса, пока конкретный deployed contract не дает более сильных гарантий. Всегда нужно различать проверяемый on-chain факт и off-chain оценку.

13. Governance и TBC Diamonds

Governance нужен для управления изменениями протокола, параметрами и развитием экосистемы. TBC Diamonds образуют описанный в протоколе governance membership layer.

Для голосования важны корректное разрешение владельца NFT на момент голосования, защита от повторного использования одного governance NFT в рамках proposal и аудируемая история решений.

14. Слой AI-агентов

AI-агент — новый тип финансового участника: программа, которая может наблюдать события, планировать действия и выполнять workflows. Безопасная архитектура не делает его владельцем. Агент становится policy-bound operator.

Владелец может разрешить агенту читать счет, мониторить события, создавать intents, исполнять небольшие операции или ребалансировать портфель в заданных пределах.

• Read permissions — balances, assets, activity, policies.

• Planning permissions — создавать intents без исполнения.

• Execution permissions — выполнять конкретные классы операций.

• Approval permissions — запрашивать человеческое подтверждение.

• Administrative permissions — обычно только owner/governance.

15. TBC Agent Protocol и MCP

TBC Agent Protocol следует позиционировать как финансовый доменный протокол поверх MCP, а не как просто MCP-сервер. MCP стандартизирует способ взаимодействия; TONBANKCARD определяет financial semantics: счета, policies, intents, approvals, capabilities и execution.

TON имеет собственную MCP-инфраструктуру для взаимодействия с блокчейном. Отличие TBC — более высокий уровень: programmable accounts, policy-aware execution, делегированные права, payments и финансовые модули.

Для чувствительных операций интерфейс должен сохранять возможность явного отказа человеком и не должен превращать естественный язык в неограниченное право тратить.

• Resources: account state, balances, capabilities, policies, execution status.

• Tools: intent.create, intent.validate, approval.request, payment.execute, swap.quote, swap.execute.

• Structured outputs: детерминированные machine-readable результаты.

• Human-in-the-loop: sensitive actions требуют явного контроля.

• Auditability: tool call связывается с intent и settlement там, где это возможно.

16. Policy Engine и авторизация

Policy Engine — центральная граница безопасности между владельцем и automation. Политика может учитывать actor, operation, asset, amount, counterparty, frequency, time window, risk score и approval requirements.

Практическая модель авторизации состоит из трех исходов: ALLOW, DENY, APPROVAL_REQUIRED. Human review является состоянием протокола, а не просто UI-исключением.

Измерение	Пример
Actor	agent:treasury-bot-01
Operation	transfer / swap / merchant_payment
Asset	TON / TBC / approved jettons
Amount	≤ 50 USDT за операцию
Period	daily / weekly limit
Counterparty	merchant allowlist
Approval	обязателен выше порога
Expiry	timestamp окончания policy

17. Цикл PLAN → AUTHORIZE → EXECUTE

Финансовая операция должна рассматриваться как жизненный цикл, а не одна кнопка. Это особенно важно для AI, где рассуждение и исполнение являются разными событиями.

- PLAN: агент формирует structured intent.
- AUTHORIZE: policy, ownership, risk и approval rules проверяются.
- EXECUTE: авторизованный субъект отправляет операцию.
- SETTLE: TON фиксирует on-chain результат.
- AUDIT: intent, decision и settlement связываются в историю.

18. Безопасность

Модель угроз должна исходить из того, что AI-агент, API или web UI могут быть скомпрометированы. Поэтому authorization должна работать по принципу least privilege и fail closed.

Чувствительные MCP tools должны валидировать вход, проверять доступ, использовать rate limits, очищать outputs, иметь bounded timeouts и audit logging.

Ключевые угрозы: prompt injection, malicious tool output, replay, policy bypass, confused deputy, compromised agent credentials, malicious counterparties и unsafe transaction construction.

• Prompt injection: недоверенный текст не должен менять policy.

• Confused deputy: execution всегда привязывается к account и principal.

• Replay: уникальный intent ID, expiry, nonce/idempotency.

• Policy bypass: проверка на execution boundary.

• Key compromise: минимальные полномочия и отдельные operator credentials.

• API compromise: проверка критических on-chain фактов.

• Approval fatigue: UI должен показывать recipient, amount, risk и причину approval.

19. Приватность и данные

Публичный blockchain раскрывает transaction facts. Поэтому чувствительные персональные данные следует минимизировать on-chain.

Публичная аналитика может использовать агрегированные данные и пороги приватности. Архитектура индексатора и dashboard должна разделять on-chain facts и off-chain presentation.

Agent tools должны возвращать только данные, необходимые для текущей операции, чтобы не превращаться в канал утечки account context.

20. Developer Platform: SDK, API, MCP

SDK, API и MCP должны использовать общий semantic model: Account, Asset, Capability, Policy, Intent, Approval, Execution, Event.

MCP не должен быть отдельной несовместимой моделью. Он должен представлять тот же protocol vocabulary, что и web/mobile/merchant integrations.

Объект	Назначение
Account	Идентичность и capabilities
Asset	Актив и его баланс
Policy	Правила делегированной авторизации
Intent	Запрошенная финансовая операция
Approval	Состояние approval
Execution	Статус отправки/settlement
Event	Изменение состояния

21. Utility TBC и экономика

TBC должен рассматриваться прежде всего как protocol utility и settlement asset в тех потоках, где это определено развернутыми контрактами.

Конкретные функции токена должны описываться через опубликованные параметры и on-chain механизмы, а не через обещания роста цены. Потенциальные utility-направления: внутренние расчеты, поддерживаемые payment/fee flows, liquidity и governance-related функции там, где они реально реализованы.

Whitepaper не является обещанием доходности, прибыли, yield или роста стоимости токена.

22. Версионирование и governance of change

Изменение semantics policy, execution adapters или AI tools может менять risk profile даже без изменения базового token contract.

Каждый release должен фиксировать version, сеть, contract addresses, migration notes, security status и совместимость. Развернутые и еще не развернутые компоненты должны быть четко разделены.

AI-facing interfaces также должны иметь версионирование, чтобы новая версия инструмента не могла неожиданно расширить полномочия агента.

23. Дорожная карта

Roadmap лучше строить по capability, а не по календарным датам. Mainnet-статус всегда должен сверяться с актуальным deployment registry и release notes.

• Фаза 1 — Account foundation: NFT-счета, некостодиальный UX, базовые assets и settlement.

• Фаза 2 — Financial modules: payment hub, merchant, DEX, recurring, multisig, lending по мере готовности.

• Фаза 3 — Agent read layer: discovery, balances, capabilities, activity, policy inspection через TBC Agent Protocol / MCP.

• Фаза 4 — Controlled execution: intents, policies, approvals, constrained execution и audit correlation.

• Фаза 5 — Autonomous finance: policy-bounded agents для recurring, treasury и оптимизации.

24. Риски и ограничения

• Smart-contract risk — уязвимости возможны даже при тестировании и аудитах.

• Integration risk — DEX, bridges и gateways добавляют внешние зависимости.

• Agent risk — AI может ошибаться или подвергаться манипуляциям.

• Policy risk — неправильные правила могут разрешить нежелательные операции.

• Data/oracle risk — off-chain данные могут быть неполными или устаревшими.

• Regulatory risk — финансовые требования зависят от юрисдикции и сценария.

• Operational risk — RPC, API и indexer могут быть недоступны при работающем blockchain.

• UX risk — пользователь может подписать операцию, не поняв ее фактические параметры.

25. Заключение

TONBANKCARD 2.0 — это не просто crypto card и не просто интеграция с MCP. Это попытка определить программируемый финансовый слой счетов, где один и тот же счет может использоваться человеком, приложением или AI-агентом, при этом ownership и policy остаются явными.

- Счет дает программный объект для финансовых действий.
- Политика задает границы полномочий.
- Агент обеспечивает automation внутри этих границ.
- TON обеспечивает settlement и проверяемое состояние.
- Human approval остается доступным там, где риск этого требует.

Приложение A. Пример Policy

Ниже показан только иллюстративный формат. Production schema должна быть формально специфицирована, версионирована и enforced в execution boundary.

{ "version": "1.0", "account": "tbc_account", "delegates": [{ "id": "agent:treasury-bot-01", "permissions": ["account.read", "payment.create", "payment.execute"], "limits": { "per_transaction": "50 USDT", "daily": "250 USDT" }, "counterparties": ["merchant:approved-001"], "approval_required_above": "50 USDT", "expires_at": "2026-12-31T23:59:59Z" }] }

Приложение B. MCP Tools

Продакшен-профиль должен раскрывать только минимальный набор инструментов, необходимый конкретному deployment mode.

Tool	Класс	Назначение
account.get	read	Account identity/capabilities
account.balances	read	Поддерживаемые balances
policy.get	read	Текущая policy
intent.create	draft	Создание structured intent
intent.validate	write	Policy/risk validation
approval.request	sensitive	Запрос human/multisig approval
payment.execute	sensitive	Исполнение платежа
swap.quote	read	Получение DEX quote

swap.execute	sensitive	Исполнение swap
execution.status	read	Settlement status

Приложение C. Терминология

Счет: программируемый финансовый объект, контролируемый владельцем.

Владелец: субъект с корневым контролем над счетом.

Делегат: субъект с ограниченными полномочиями.

Агент: программный делегат, работающий по policy.

Policy: машинно-исполняемые правила разрешенного поведения.

Intent: структурированное описание финансового действия.

Approval: явное подтверждение чувствительной операции.

Execution: отправка и settlement blockchain-операции.

TBC Agent Protocol: доменный протокол TONBANKCARD для агентского взаимодействия.

MCP: Model Context Protocol, используемый как interoperability layer; policy и финансовая семантика остаются частью TBC.

Источники

TONBANKCARD Protocol: github.com/xlabtg/tonbankcard-protocol

TON Docs — Jettons и token standards: docs.ton.org

TON Docs — TON Connect API: docs.ton.org/applications/ton-connect/api-reference/protocol

TON Docs — AI / MCP: docs.ton.org/onboarding/ai/mcp

Model Context Protocol — specification и security guidance: github.com/modelcontextprotocol/modelcontextprotocol

Google AI Studio Build mode: ai.google.dev/gemini-api/docs/aistudio-build-mode

Издательская оговорка

Этот Whitepaper 2.0 является архитектурным и продуктовым документом. Перед публичной публикацией необходимо сверить deployment status, contract addresses, аудит, token parameters и список поддерживаемых интеграций с актуальными release notes и registry TONBANKCARD.