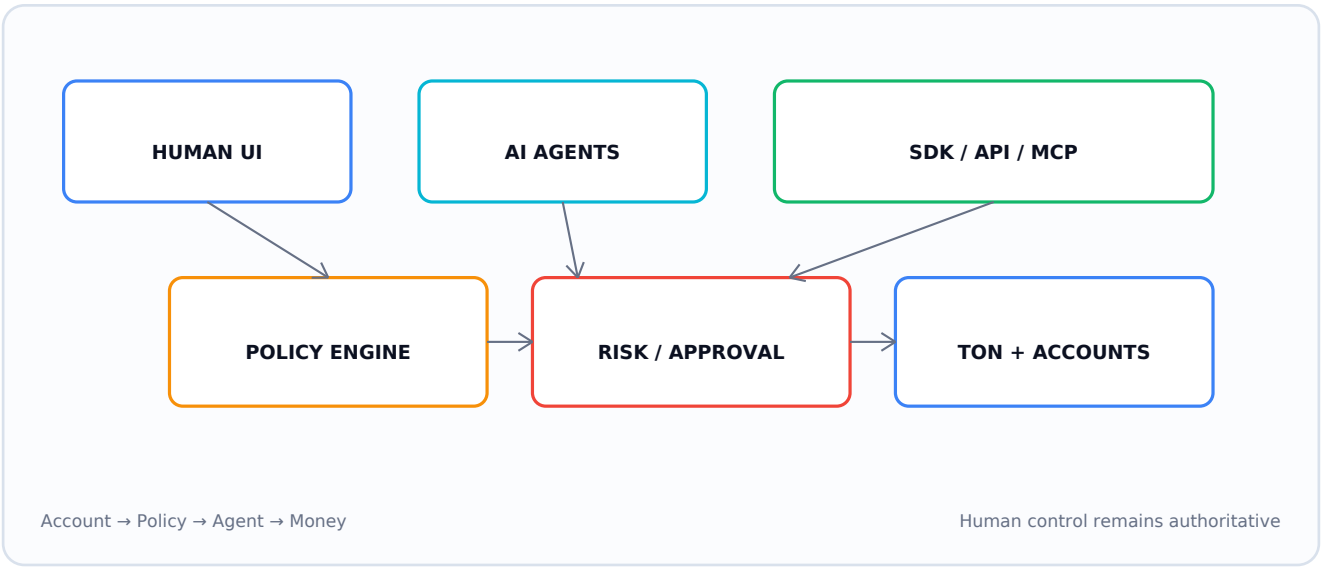
TONBANKCARD

WHITEPAPER 2.0

The Programmable Financial Account Layer for TON

Human banking interface + programmable financial accounts for AI agents and applications.



Core thesis

A financial account should be more than an address and more than a UI balance. It should be a programmable object with explicit ownership, permissions, policies, execution rules and an auditable transaction lifecycle.

Document status: Whitepaper 2.0 — protocol architecture and product direction. Features are classified as current, in development, or proposed; this document does not imply that every described module is deployed on mainnet.

One-Page Thesis

TONBANKCARD is evolving from a non-custodial account protocol into a programmable financial account layer for humans, applications and AI agents.

The protocol combines an NFT-based account model on TON with policy-controlled authorization, payment and settlement primitives, developer SDK/API interfaces, and an MCP-compatible agent layer. The objective is not to give AI ownership of money. The objective is to let humans define what an agent is allowed to do with a financial account — and enforce those rules.

ACCOUNT	A persistent financial account represented by a programmable on-chain object.
POLICY	Explicit limits, permissions, conditions, approvals and risk rules.
AGENT	An authorized operator that can act within policy; not the owner.
MONEY	TON, jettons, NFTs and supported financial operations executed through the account layer.

Design rule: Human authority is the root of trust; AI is a delegated execution layer.

This architecture separates five concerns that are often mixed together: asset ownership, account state, authorization, execution, and user experience. That separation enables both familiar banking interfaces and machine-native financial workflows.

Contents

1. Abstract

TONBANKCARD is a non-custodial financial account protocol built around the TON blockchain. Its original architectural idea is simple: an NFT can represent a unique financial account, while assets and account operations remain controlled by the account owner through blockchain transactions.

Whitepaper 2.0 extends this model. The account becomes a programmable financial object that can expose controlled capabilities to applications and AI agents. An agent can read balances, prepare payment intents, execute approved operations, interact with supported financial modules, and maintain an auditable history — but only within permissions defined by the account owner and enforced by protocol-level controls.

The resulting stack is designed to support both human-facing banking UX and machine-native finance: one account, multiple interfaces, explicit authorization, policy-based execution and verifiable settlement.

2. Executive Summary

The next generation of digital finance will not be exclusively human-operated. Software agents will increasingly monitor markets, reconcile invoices, manage treasury, execute recurring workflows and interact with merchants and protocols. A conventional wallet address is not a sufficient abstraction for this environment because an address alone does not express who may act, what they may do, under which conditions, or how an action should be approved.

TONBANKCARD proposes the Programmable Financial Account as the missing abstraction. The account owns or controls assets; policies define permissible behavior; agents operate as delegated actors; and the blockchain remains the settlement and verification layer.

• Human-first: familiar banking UX remains the primary entry point.

• Non-custodial: the protocol is designed so users retain control of their assets and signing authority.

• Policy-driven: permissions are explicit rather than implicit.

• Agent-ready: AI agents can use standardized machine interfaces without becoming asset owners.

• Composable: payments, DEX, lending, recurring payments, multisig and merchant workflows can share account semantics.

• Auditable: material actions should produce an inspectable intent, authorization and execution trail.

3. Problem Statement

Crypto wallets are excellent at holding keys and signing transactions, but a financial account needs more context. A user thinks in terms of spending limits, subscriptions, invoices, counterparties, approvals, treasury roles and business rules. An AI agent needs the same semantics in machine-readable form.

The missing layer is therefore not another wallet UI. It is a control plane for financial accounts: a system that connects ownership, policy, agent identity, risk decisions and execution.

• Wallets expose addresses; financial accounts expose capabilities.

• Raw transactions describe what happened; intents describe what the user wanted to happen.

• Private keys prove authority; policies define the scope of delegated authority.

• AI agents can automate actions, but should not silently acquire unrestricted control.

• Multiple interfaces should operate on the same account state and authorization model.

4. Design Principles

• Self-custody by default. User-controlled signing and ownership remain foundational.

• On-chain truth. Ownership and settlement facts should be verifiable on TON.

• Least privilege. Agents and applications receive only the permissions they need.

• Explicit human approval. High-impact or anomalous operations can require human confirmation.

• Deterministic policy. The same policy input should lead to predictable authorization outcomes.

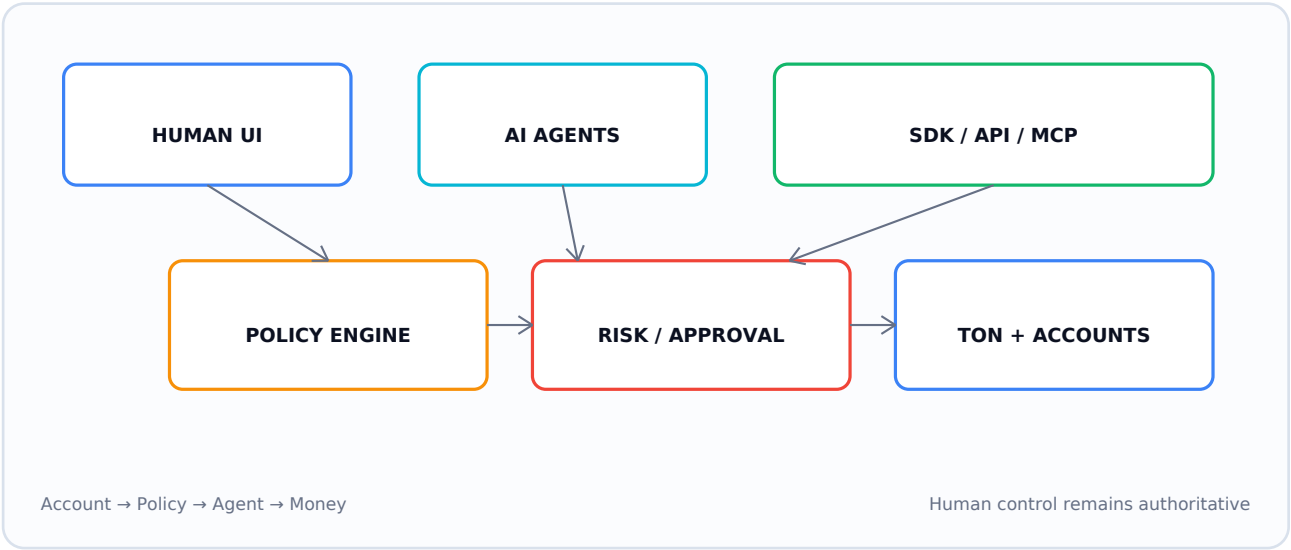
• Composable modules. Financial capabilities are modular and versioned.

• Observable execution. Intent, authorization decision and resulting transaction should be correlated where practical.

• Fail closed. Ambiguous or invalid authorization should not silently become permission to spend.

5. Protocol Architecture

The architecture is intentionally layered. TON provides the settlement substrate. TONBANKCARD provides account semantics and financial modules. SDK/API interfaces expose those semantics to applications. MCP provides a standardized interaction surface for compatible AI agents.



Layer	Responsibility
TON	Blockchain settlement, messages, smart contracts, token standards and final on-chain state.

TBC Account Layer	Account identity, account state, NFT-based account representation, ownership and supported operations.
Policy & Authorization	Permissions, spending limits, conditions, approvals, roles and agent delegation.
Financial Modules	Payments, merchant flows, recurring payments, DEX integration, lending/collateral, multisig and external gateways.
Developer Layer	SDK, API, indexer/read services and integration contracts.
Agent Layer	TBC Agent Protocol / MCP tools and resources for machine-readable financial interaction.
Human UX	Web/mobile banking interface, transaction review, approvals, activity and policy management.

6. Account Model

In TONBANKCARD, an NFT can represent a unique account. The NFT account abstraction creates a persistent identity around a financial account rather than treating each transaction as an isolated wallet event.

Account state should be understood as a combination of: owner/control authority, supported assets, enabled capabilities, policy configuration, delegates, pending approvals, operational status and historical activity.

The account abstraction is deliberately broader than a payment card. A card-like interface is one presentation of an account that can also participate in programmable workflows.

• Account identity: a stable identifier for applications and users.

• Ownership: authoritative control over the account.

• Capabilities: what operations the account supports.

• Policy: what delegated actors are permitted to do.

• State: balances, pending actions, limits and operational metadata.

• History: transactions and correlated intents/decisions where available.

7. Ownership, Control and Non-Custody

Non-custodial architecture is a core constraint rather than a marketing attribute. The user should retain control of the authority required to move assets. The user-facing wallet layer should not collect or store private keys.

TON Connect can be used as the user-controlled signing interface. The application prepares a transaction or structured request; a compatible wallet presents it to the user; the wallet performs signing. This preserves a clean separation between application UX and key custody.

The agent layer follows the same principle. An agent is a delegate or operator with constrained authority, not a replacement for the owner.

8. Asset and Settlement Model

The account layer is designed to work with TON-native assets and supported tokens. Jettons follow TON token standards, with a master contract defining token metadata and individual wallet contracts holding balances. Applications must treat token decimals and token-wallet addresses correctly.

Internal protocol settlement can use TBC where specified by the deployed protocol modules. The design goal is predictable, low-friction settlement while preserving transparent on-chain accounting.

Asset support should be capability-driven: a client must be able to discover whether a given account/module supports a particular asset or operation before attempting execution.

9. Payments and Merchant Layer

Merchant payments translate the programmable account model into real commerce workflows. A merchant should be able to request a payment without requiring the user to understand raw blockchain transactions.

A payment intent can contain recipient, asset, amount, reference, expiration, optional conditions and risk metadata. The account evaluates the request against policy and, if authorized, produces the corresponding blockchain execution.

Merchant integrations should use deterministic identifiers, replay protection, signature/authentication controls, idempotency and auditable status transitions.

10. DEX, Liquidity and External Gateways

TONBANKCARD can integrate decentralized exchanges and external gateways as modular adapters. The account abstraction allows the same authorization model to govern swaps, conversions and supported external operations.

DEX operations should include slippage protection, asset allowlists where configured, minimum/maximum amounts and deadline controls. External gateways require additional trust boundaries because execution may depend on an off-chain provider.

Adapters should never silently expand account permissions. Each integration should declare the assets, operations, fees and trust assumptions it introduces.

11. Recurring Payments and Multisig Accounts

Recurring payments are naturally expressed as policies and mandates. A mandate can specify a merchant, maximum amount, frequency, validity period and cancellation conditions. Each execution remains subject to the mandate state and account policy.

Multisig accounts extend the same model to teams and organizations. M-of-N signing rules can separate operational roles, treasury control and recovery. Agent permissions can be nested beneath the multisig authority rather than replacing it.

Examples include personal 2-of-3 recovery setups, corporate 3-of-5 treasury control and custom signer sets within supported protocol limits.

12. Lending and Collateral Signals

Lending modules can consume account state and collateral information without requiring the account abstraction itself to become a lender or custodian. A separation between account identity, collateral signals, lending adapters and liquidity providers reduces coupling.

Collateral signals and public lookup mechanisms should be treated as informational or policy inputs unless a specific deployed contract establishes stronger guarantees. Applications must distinguish between verified on-chain facts and off-chain estimates.

13. Governance and TBC Diamonds

Protocol governance provides a mechanism for controlling upgrades, parameter changes and community decisions. TBC Diamonds represent the governance membership layer described by the protocol.

A governance system should resolve NFT ownership at the time of voting, prevent duplicate votes by the same governance NFT for a proposal, and retain an auditable proposal and vote history. Governance authority should be separated from routine user asset custody.

14. AI Agent Layer

AI agents introduce a new class of financial actor: software that can reason, monitor events, prepare actions and execute workflows. The protocol's response is not to make agents owners. Instead, agents become policy-bound operators.

An agent may be allowed to read an account, monitor conditions, prepare an intent, execute small recurring payments or rebalance a portfolio — but only within an explicit permission envelope.

This creates a clean division of responsibility: the human defines authority; the policy engine evaluates authority; the agent proposes or performs an operation; the blockchain settles the result.

- Read permissions: balances, supported assets, activity and policy state.
- Planning permissions: create drafts or payment intents without execution.

• Execution permissions: execute only specified classes of operations.

• Approval permissions: request human confirmation for sensitive actions.

• Administrative permissions: generally reserved for the owner or governance authority.

15. TBC Agent Protocol and MCP

The TBC Agent Protocol should be positioned as a financial protocol interface powered by MCP, rather than as merely an MCP server. MCP is the standardized interaction layer; TONBANKCARD supplies the domain model: accounts, policies, intents, approvals, financial capabilities and execution semantics.

MCP-compatible tools can expose account operations to agents while preserving authorization and confirmation controls. The protocol should support both read-only and execution-capable deployments.

TON also provides an official MCP ecosystem for interacting with TON capabilities. TBC's differentiation is therefore the higher-level financial-account layer: policy-aware accounts, merchant/payment semantics, delegated permissions, risk controls and protocol-specific modules.

• Resources: account state, balances, capabilities, policies and transaction status.

• Tools: create intent, quote, validate policy, request approval, execute approved operation, inspect status.

• Structured outputs: deterministic machine-readable results rather than unbounded natural-language responses.

• Human-in-the-loop: sensitive operations must remain rejectable by the account owner.

• Auditability: tool calls should be correlatable to an intent and resulting transaction where possible.

16. Policy Engine and Authorization

The policy engine is the central safety boundary between account authority and delegated automation. A policy can combine subject, operation, asset, amount, counterparty, frequency, time window, risk score and approval requirements.

A useful authorization model is a three-way decision: ALLOW, DENY or APPROVAL_REQUIRED. This makes human review a first-class protocol state rather than an exceptional UI event.

Dimension	Example
Actor	agent:treasury-bot-01

Operation	transfer / swap / merchant_payment
Asset	TON, TBC, approved jettons
Amount	≤ 50 USDT per transaction
Period	daily / weekly limit
Counterparty	merchant allowlist
Approval	required above threshold
Expiry	policy validity timestamp

17. Transaction Lifecycle: PLAN → AUTHORIZE → EXECUTE

The protocol should treat financial actions as a lifecycle rather than a single button press. This is especially important for AI agents because reasoning and execution are separate events.

• PLAN: the agent or application creates a structured intent.

• AUTHORIZE: account policy, ownership, risk and approval rules are evaluated.

• EXECUTE: an authorized actor submits the transaction through the appropriate signing/execution path.

• SETTLE: TON provides the on-chain settlement result.

• AUDIT: the system correlates intent, decision and settlement where technically possible.

18. Security and Threat Model

Security must assume that agents, external APIs and user interfaces can be compromised. The account layer therefore follows least privilege and fail-closed authorization.

Sensitive MCP tools should validate inputs, enforce access control, apply rate limits, sanitize outputs, use bounded execution time, and record audit events. Authentication and authorization tokens should be audience-bound and short-lived where applicable.

The protocol must explicitly defend against prompt injection, malicious tool output, replayed intents, policy bypass, confused-deputy behavior, compromised agent credentials, malicious counterparties and unsafe transaction construction.

• Prompt injection: never let untrusted text redefine financial policy.

• Confused deputy: bind every execution to a concrete account and authorized principal.

• Replay: use unique intent IDs, expirations and nonce/idempotency controls.

• Policy bypass: evaluate policy at the execution boundary, not only in the UI.

• Key compromise: minimize delegated authority and isolate operator credentials.

• API compromise: treat off-chain services as untrusted and verify critical on-chain facts.

• Human approval fatigue: surface risk, recipient, amount and policy reason clearly.

19. Privacy and Data Architecture

Public blockchains expose transaction facts by design. The protocol should therefore minimize unnecessary personal data on-chain and keep sensitive metadata off-chain where possible.

Analytics and public dashboards can use aggregated views and privacy-preserving thresholds. The repository's analytics architecture describes an indexer/read-replica/aggregator/API/dashboard model and a k-anonymity floor for public reporting.

Agent systems should also avoid leaking private account context through tool responses. Tool outputs should return only the minimum information required for the requested operation.

20. Developer Platform: SDK, API and MCP

The developer platform should expose one consistent financial vocabulary across web, mobile, merchant integrations and agents. SDKs and APIs should share canonical objects such as Account, Asset, Capability, Policy, Intent, Approval and Execution.

The MCP surface should be generated from the same semantic model rather than implementing a second, incompatible API. This reduces integration drift and makes policy behavior consistent across humans and machines.

Object	Purpose
Account	Identity and current capabilities
Asset	Supported asset and balance representation
Policy	Delegated authorization rules
Intent	Requested financial operation
Approval	Human or multisig authorization state
Execution	Submitted/settled transaction status
Event	Machine-readable state change

21. Token Utility and Protocol Economics

TBC is intended to function as a protocol utility and settlement asset within supported TONBANKCARD flows. Its concrete utility should be defined by deployed contracts and published parameters rather than by speculative promises.

Potential protocol utility can include internal settlement, fee/payment flows where enabled, liquidity participation and governance-adjacent functions where explicitly implemented. Token economics should prioritize transparent supply, contract addresses, allocation disclosures and on-chain verifiability.

Nothing in this whitepaper should be interpreted as a promise of token price appreciation, yield, profit or investment return.

22. Versioning and Governance of Change

Programmable finance requires strong version discipline. A change to policy semantics, execution adapters or agent tools can alter the effective risk surface even if the underlying token contracts do not change.

Each protocol release should publish a version, supported networks, contract addresses, migration notes, security status and compatibility guarantees. Deployed contracts and undeployed components must be clearly distinguished.

AI-facing interfaces should be versioned independently enough to prevent a new tool definition from unexpectedly changing an agent's authority.

23. Roadmap

Roadmap sequencing is intentionally capability-based rather than date-based. Mainnet status must always be verified against the current deployment registry and release notes.

• Phase 1 — Account foundation: NFT account model, non-custodial UX, supported assets and core settlement.

• Phase 2 — Financial modules: payment hub, merchant flows, DEX adapters, recurring payments, multisig and lending integrations as modules reach deployment readiness.

• Phase 3 — Agent read layer: account discovery, balances, capabilities, activity and policy inspection through TBC Agent Protocol / MCP.

• Phase 4 — Controlled agent execution: intents, policy checks, approvals, constrained execution and audit correlation.

• Phase 5 — Autonomous finance: policy-bounded agents managing recurring workflows, treasury operations and optimization without receiving unrestricted ownership.

24. Risks and Limitations

• Smart-contract risk: code can contain vulnerabilities despite testing and audits.

• Integration risk: DEXs, bridges and gateways introduce external dependencies and trust assumptions.

• Agent risk: AI systems can misunderstand goals or be manipulated by untrusted inputs.

• Policy risk: badly configured limits can authorize unwanted behavior.

• Oracle/data risk: off-chain data may be stale, incomplete or adversarial.

• Regulatory risk: financial services rules differ across jurisdictions and use cases.

• Operational risk: indexers, RPC endpoints and APIs can fail even when on-chain contracts remain operational.

• UX risk: users may approve transactions without understanding the effective policy or recipient.

25. Conclusion

TONBANKCARD 2.0 is best understood not as another crypto card and not simply as an MCP integration. It is an attempt to define a programmable financial account layer in which the same account can be operated by a human, an application or an AI agent — while ownership and policy remain explicit.

The conceptual stack is simple:

• The account gives software a stable financial object to operate.

• The policy defines the boundary of authority.

• The agent provides automation within that boundary.

• TON provides settlement and verifiable state.

• Human approval remains available wherever the risk model requires it.

Appendix A — Example Policy

The following illustrative policy is intentionally simple. Production policy schemas should be formally specified, versioned and enforced at the appropriate 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" }] }

Appendix B — Example MCP Tool Surface

A production implementation should expose only the minimum tool surface required for the deployment mode.

Tool	Class	Purpose
account.get	read	Read account identity/capabilities
account.balances	read	Read supported balances
policy.get	read	Read active policy
intent.create	write/draft	Create a structured financial intent
intent.validate	write	Evaluate policy/risk before execution
approval.request	sensitive	Request human or multisig approval
payment.execute	sensitive	Execute an authorized payment
swap.quote	read	Get a DEX quote
swap.execute	sensitive	Execute an authorized swap
execution.status	read	Inspect settlement status

Appendix C — Terminology

Account: the programmable financial object controlled by an owner.

Owner: the authority that ultimately controls the account.

Delegate: an actor authorized to perform a limited set of actions.

Agent: software operating as a delegate under policy.

Policy: machine-enforceable rules defining permitted behavior.

Intent: a structured representation of a requested financial action.

Approval: explicit authorization required before a sensitive action can execute.

Execution: the transaction submission and settlement process.

TBC Agent Protocol: the TONBANKCARD domain protocol for agent-facing financial interactions, with MCP as a standardized interaction layer.

MCP: Model Context Protocol, used here as an interoperability surface rather than as the financial policy engine itself.

References

TONBANKCARD Protocol repository and documentation: github.com/xlabtg/tonbankcard-protocol

TON documentation — Jettons / token standards: docs.ton.org

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

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

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

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

Publication note

This Whitepaper 2.0 is an architectural and product specification draft. Contract deployment status, addresses, security/audit status, token parameters and supported integrations must be verified against the current TONBANKCARD release registry before publication as a final protocol document.