TYKORA

CYCLE FLOW

Save together. Win together. On Rootstock

Table of Contents

Overview 2

1 - Key Concepts & Terminology 2

2 - Preconditions 2

User Journey Flow 3

1 – Open the dApp 3

2 – Connect Wallet 3

3 – Review the Main Sections (Orientation) 4

4 – Approve & Deposit (First Deposit or Insufficient Allowance) 5

5 – Monitor Position During the Cycle 7

6 – Withdraw (When Vault Is Unlocked) 7

7 – Refresh (Manual State Sync) 8

Vault Cycle Lifecycle 9

Notes on Transparency & Verification 9

Overview

Tykora is a savings vault dApp deployed on **Rootstock**, designed to let users deposit **DOC/USDRIF** and participate in yield generation and prize cycles.

- **DOC** is a USD-pegged asset from **MoneyOnChain**, widely used across the Rootstock ecosystem.
- **USDRIF** is a Rootstock-native stablecoin simple, transparent, and verifiable
- Deposited funds are allocated to generate yield via **Tropykus**, a lending market on Rootstock. The yield produced during each cycle is then split according to the vault's configured distribution (e.g., prize, treasury, keeper), and the **prize component** is accumulated for the current draw.

At a high level, the experience is:

1. Connect a wallet to the dApp.
2. Deposit DOC/USDRIF into the vault (first-time users must approve DOC/USDRIF spending).
3. Hold shares during the cycle while yield accrues via Tropykus.
4. Withdraw DOC/USDRIF when the vault is unlocked (subject to the vault's current lock state).
5. Claim a prize if the user has Prize owed > 0.
6. Track transparency via on-chain data (draw status, previous winner, and explorer links).

Tykora's UI exposes three primary information areas:

- **Prize (Current Draw):** draw lifecycle visibility (prize amount, yield, draw ID, status/lock state, countdown, previous winner).
- **My Position:** the connected user's balances and entitlement (shares, wallet balance, allowance, prize owed, action inputs/buttons).
- **Vault Stats:** vault-level aggregates (TVL/principal, total assets, underlying token, and the yield split such as prize/treasury/keeper).

1 - Key Concepts & Terminology

- **Underlying Token:** DOC/USDRIF (the asset deposited into the vault).
- **Shares:** vault accounting units representing a depositor's position. Shares typically increase only when depositing and decrease when withdrawing (depending on vault mechanics).
- **TVL (Principal):** total deposited principal across all users.
- **Total Assets:** principal plus any accrued yield (vault-level view).
- **Cycle / Draw:** a defined period in which yield accrues and is associated with a prize outcome.
- **Status / Locked:** indicates whether the vault is currently in a state that restricts certain actions (notably withdrawals).
- **Prize Owed:** claimable prize amount for the connected wallet (if any).
- **Allowance:** how much DOC/USDRIF the user has authorized the vault to spend from their wallet.

2 - Preconditions

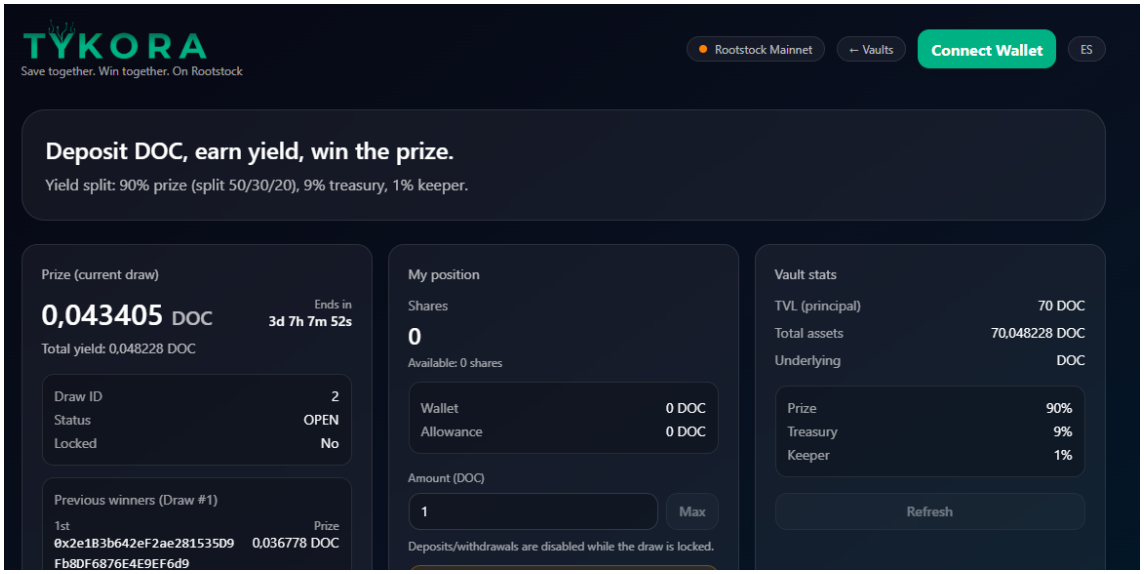
- A compatible wallet (e.g., **MetaMask**, **WalletConnect, and others**, hardware wallet via supported connectors).

- Wallet configured for **Rootstock** network.
- The user holds **DOC/USDRIF** and enough **RBTC** to pay transaction fees on Rootstock.
- Access to the Tykora dApp URL.

User Journey Flow

1 – Open the dApp

User action: Navigate to the Tykora landing page https://tykora.jxlabs.xyz/.



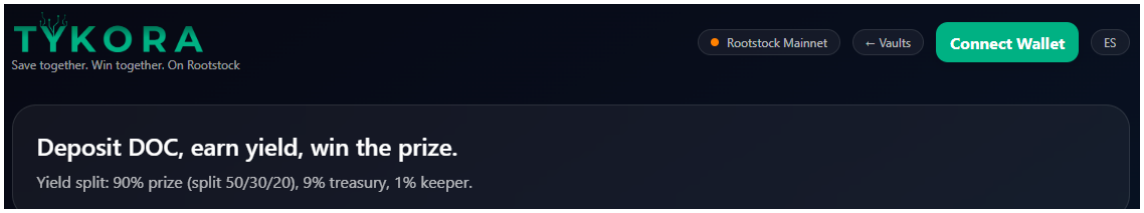
2 – Connect Wallet

User action: Click **Connect Wallet**, select a wallet provider, and approve the connection in the wallet.

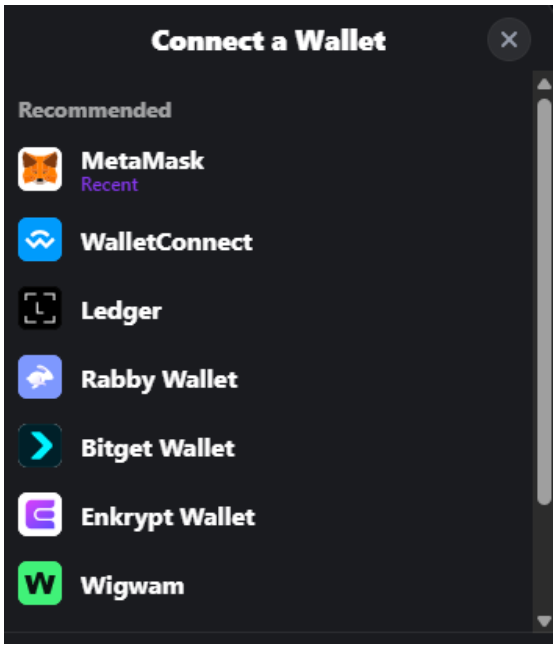
System behavior:

- The UI fetches the connected address and displays:
  - User DOC/USDRIF balance (Wallet)
  - Shares (if any)
  - Allowance
  - Prize owed
  - Current draw information

Outcome: The dApp is now personalized to the connected account.



Pressing the button will open the connection wallet options.



3 – Review the Main Sections (Orientation)

Prize (Current Draw)

- Prize amount and/or total yield accumulated for the draw
- Draw ID
- Status and whether the vault is locked
- Time remaining ("Ends in")
- Previous winners (typically linked to an explorer view)

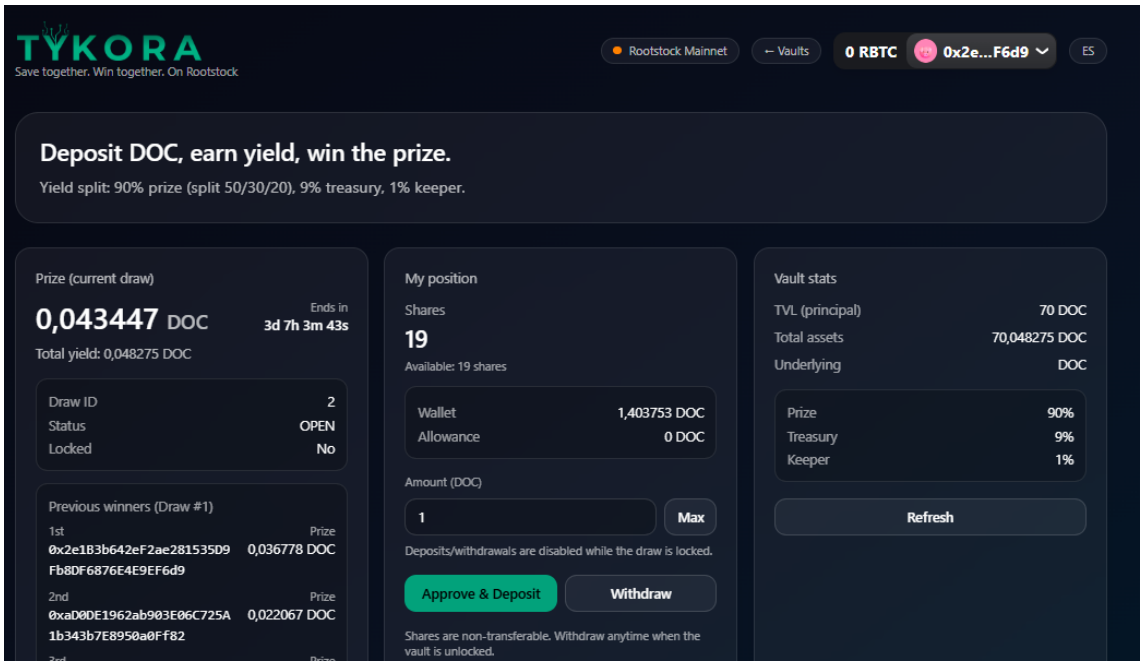
My Position

- Shares
- Wallet balance (DOC/USDRIF)
- Allowance (DOC/USDRIF approval)
- Amount input field + action buttons (Approve/Deposit/Withdraw/Claim)

Vault Stats

- TVL (principal)
- Total assets (principal + yield)
- Underlying token (DOC/USDRIF)
- Yield split (e.g., prize / treasury / keeper)

.



4 – Approve & Deposit (First Deposit or Insufficient Allowance)

User action: Enter an amount (e.g., **1 DOC/USDRIF**) and click **Approve & Deposit** (or **Deposit** if already approved).

Two-step transaction pattern (typical ERC-20 flow):

1. **Approve:** user authorizes the vault contract to spend DOC/USDRIF.
2. **Deposit:** user transfers DOC/USDRIF into the vault and receives shares.

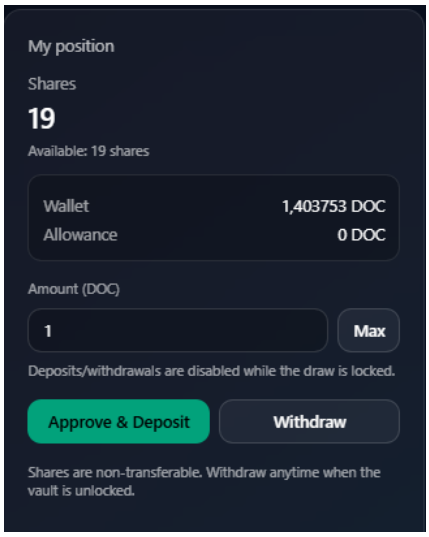
System behavior:

- Wallet prompts appear for each transaction that is needed.
- A confirmation/progress overlay is displayed while the transaction is mined/finalized on Rootstock.
- After confirmation, the UI updates:
  - **Shares** increase
  - **Wallet** DOC/USDRIF balance decreases by the deposited amount
  - Vault totals (TVL/Total assets) may update

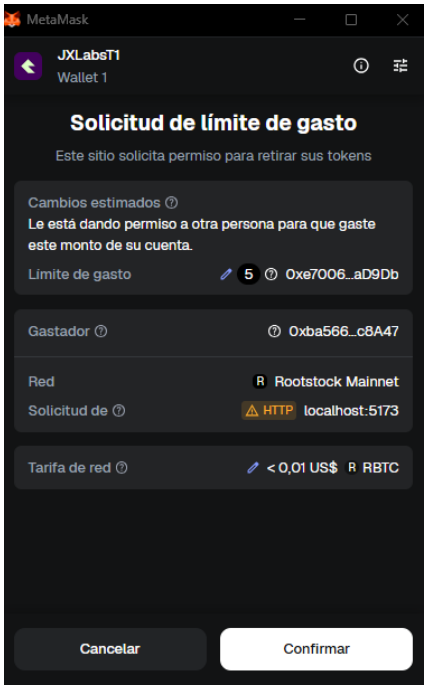
Outcome: User now holds shares and participates in the current cycle.

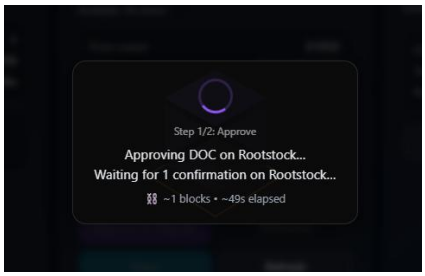
Example:

Set Amount (DOC/USDRIF) and press Approve & Deposit.

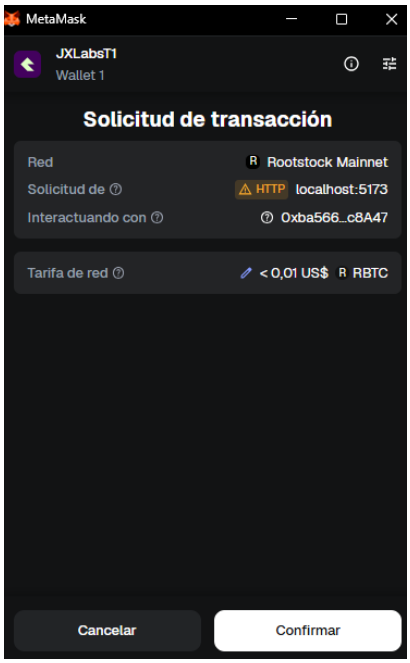


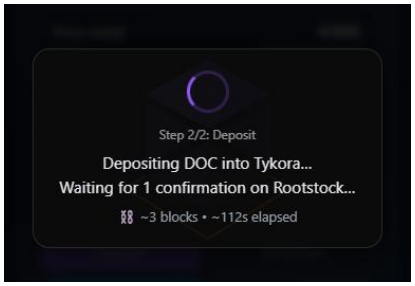
Approve in your wallet and wait





Now approve Deposit and wait





After confirmation user now holds shares and participates in the current cycle.

5 – Monitor Position During the Cycle

User action: No action required; user can periodically check:

- Current draw status, ends-in countdown, and prize size
- Their shares and wallet balance
- Prize owed (usually 0 unless the user becomes eligible to claim)

System behavior: On refresh, the dApp re-reads on-chain values and updates the UI.

6 – Withdraw (When Vault Is Unlocked)

User action: Enter a withdraw amount and click **Withdraw**, then confirm in the wallet.

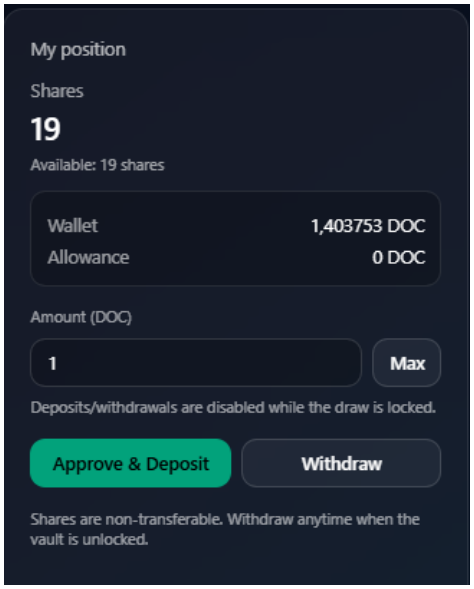
System behavior:

- If the vault is **locked**, withdrawing may be disabled or rejected (UI should prevent the action, but on-chain rules are the source of truth).
- On successful withdrawal:
- Shares decrease (burned or debited to match the withdrawn amount)
- Wallet DOC balance increases accordingly
- Vault totals may update after confirmation

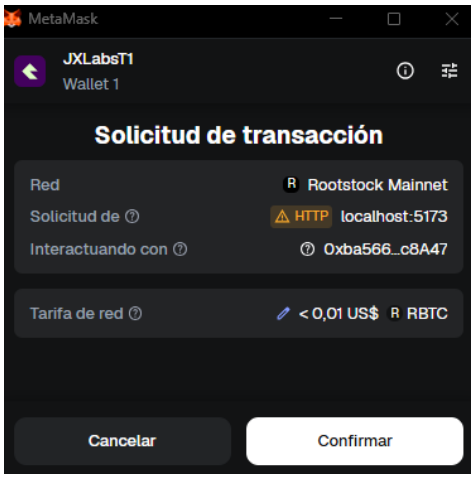
Outcome: User exits partially or fully when withdrawals are permitted by the vault state.

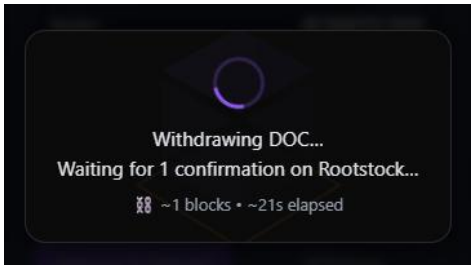
Example:

Set Amount (DOC) and press Withdraw**.**



Approve in your wallet and wait





When transaction in complete Amount returned in your wallet.

7 – Refresh (Manual State Sync)

User action: Click **Refresh** to re-fetch on-chain data.
Use cases:

- Immediately after a transaction

- If the UI appears stale
- To confirm draw status or countdown changes

Vault Cycle Lifecycle

The vault operates under a **fixed, deterministic cycle lifecycle** defined by the deployed contracts. The UI surfaces these stages and their current status, but the underlying sequence is not variable.

1. Cycle Opens
   - Deposits are accepted.
   - Yield begins accruing.
2. Cycle Progresses
   - Prize amount grows as yield accumulates (per yield split).
3. Vault Locks
   - Certain actions (commonly withdrawals) become restricted.
4. Cycle Finalizes
   - The draw ends and the winner is determined according to on-chain logic.
5. Prize Becomes Claimable
   - The winning address (or eligible claimants, depending on rules) can claim.
6. Next Cycle Begins
   - A new draw ID/status appears and the process repeats.

Notes on Transparency & Verification

Key draw fields (Draw ID, status, previous winner(s)) are publicly verifiable on-chain.

Explorer links (e.g., award/claim transactions and winner addresses) support auditability and user trust.

BTC-anchored seed and deterministic winner selection

Tykora's draw outcome is derived from a **Bitcoin-anchored seed** obtained via the **Rootstock BTC bridge**, and winners are selected **deterministically** on-chain from that seed.

1) BTC target height is fixed at draw close
When a draw is closed, the contract computes a **BTC target height** as:

- btcTargetHeight = bestBtcHeight + btcConfirmations

This height is stored in draws[drawId].btcTargetHeight at closeDraw(). (If bridge == address(0), the contract sets btcTargetHeight = 0 for local/manual modes.)

2) BTC block header is retrieved from the Rootstock bridge
To award the draw using BTC randomness, awardDrawFromBtc(drawId) requests the BTC block header at the stored height:

- header = bridge.getBtcBlockchainBlockHeaderByHeight(btcTargetHeight)

If the header is not yet available, the call reverts with RandomnessNotReady(btcTargetHeight).

3) BTC hash and seed derivation
Once the header is available, the contract derives:

- btcHash = sha256(sha256(header)) (double-SHA256)
- seed = keccak256(abi.encodePacked(btcHash, address(this), drawId))

Both values are stored on-chain (draws[drawId].btcHash and draws[drawId].seed) and emitted via DrawAwarded.

4) Winner selection (up to 3 distinct winners, weighted by shares)
Winner selection is performed with **share-weighted tickets** using a **Fenwick tree** (sum tree). At closeDraw(), the contract snapshots:

- tickets = totalTickets() into draws[drawId].tickets

Then _pickWinnersDistinct(seed, tickets) selects up to **3 distinct winners** without repetition:

For each winner index i ∈ {0,1,2}:

- s_i = keccak256(abi.encodePacked(seed, i))
- r = uint256(s_i) % remainingTickets
- The Fenwick tree resolves r to a participant index (_fenwickFindByCumulative(r)), which maps to a user address and its weight (shares).
- The selected winner's weight is **temporarily removed** from the Fenwick tree to prevent selecting the same address again, then restored at the end.

This makes the probability of winning **proportional to the user's share balance** at draw close (and ensures no duplicate winners within the same draw).

5) Prize split
After winners are selected, the contract computes prize amounts as:

- **3 winners:** 50% / 30% / 20%

6) Independent verification
Anyone can verify the draw by reading on-chain:

- btcTargetHeight, btcHash, seed, winners[], and winnerPrizes[] from draws[drawId] (and/or events DrawClosed / DrawAwarded / DrawClaimed),
  and reproducing:
- double-SHA256(header) → btcHash
- keccak256(btcHash, vaultAddress, drawId) → seed
- the weighted selection procedure described above.