The 0-100 Engine®

The First Fair Launch Primitive

The Collective

January 2026

v1.0

Abstract

Bonding curves enabled permissionless launches but favored speed over fairness. The 0-100 Engine is the next primitive: a fully onchain launch mechanism where the rules are fixed, verifiable, and equal for everyone.

Each launch follows a single, immutable preset: capped contributions, ticket-based funding, randomized selection for overflow, and automatic liquidity deployment. No creator-side tuning. No discretionary decisions. All parameters enforced by the protocol.

Unused contributions are fully refundable. Supply splits are standardized. Liquidity is protocol-owned. Post-listing fees flow back to creators, the community, and the Xyber treasury.

Fairness is a property of the code, not a promise.

Contents

1 Problem Statement and Design Principles

1.1 Structural Issues in Existing Launch Mechanisms

Open token launches expose participants to a set of well-documented structural failures:

- **Execution asymmetry.** Participants with superior infrastructure (custom RPCs, MEV bots, private orderflow) gain persistent advantages through priority gas auctions and optimized transaction ordering.
- **Allocation opacity.** Distribution is governed by allowlists, off-chain negotiations, or configurable policies, making independent verification of allocation integrity impossible.
- **Liquidity initialization risk.** Pool parameters, initial pricing, and lockup conditions are frequently controlled by creators, enabling fake pools, mispriced listings, or liquidity withdrawal post-launch.
- **Capital inefficiency.** In oversubscribed sales, excess funds are trapped or returned through ad-hoc logic. Users incur transaction costs without guarantees on final allocation or refundability.

As demand for a launch increases, these structural problems become more severe: more bots, more gas competition, and greater uncertainty around how and when liquidity will actually be deployed.

1.2 Design Principles

The 0-100 Engine addresses these failures through six invariants enforced at the protocol level:

1. **Symmetric participation.** Per-wallet caps and ticketized contributions ensure allocation is determined by protocol rules, not transaction speed, gas bidding, or off-chain negotiation.
2. **Deterministic execution.** The entire lifecycle (funding, selection, minting, liquidity creation, claims) is encoded in a self-contained program with deterministic state transitions and explicit authority separation via PDAs.
3. **Unbiased randomness.** Overflow selection and LP timing derive from blockhash-based randomness. All participants face identical probabilistic conditions.
4. **Protocol-owned liquidity.** LP parameters are precommitted. A fixed supply share is routed into protocol-controlled liquidity with defined lockup behavior.
5. **Transparent economics.** Supply splits and post-listing fee distribution are standardized and onchain. No per-project negotiation.
6. **Non-custodial execution.** All funds flow through PDAs. No single key can redirect assets, override selection, or modify parameters post-initialization.

These invariants collectively define the constraints under which the 0-100 Engine operates: one preset, fully observable onchain behavior, and no reliance on discretionary off-chain decisions for either allocation or economics.

2 Launch Models Comparison

Existing launchpads generally follow one of three patterns: gas-priority IDOs, private/whitelisted allocations, or unstructured meme-driven listings. Each model defines execution rules, allocation logic, liquidity formation, and incentive alignment in fundamentally different ways.

The 0-100 Engine introduces a fourth category: a standardized, non-configurable preset where all core parameters are enforced directly onchain.

Criterion	**Traditional IDOs**	**Private/ Whitelisted**	**Meme Pumps**	**0-100 Engine**
Execution	Gas bidding, mempool ordering	Off-chain selection, allowlists	Instant trading, high bot activity	Deterministic flow, fixed tickets, strict caps
Allocation	First-come-first-served	Discretionary per-user	No structured sale	Randomized selection; full refunds
Transparency	Low	Low	Medium	High
Fairness	Bot advantage	Insider advantage	Early-buyer dominance	Symmetric constraints
Liquidity	Manual, discretionary	Negotiated terms	Creator-controlled	Protocol-owned, automated
Capital Efficiency	Low	Medium	Unpredictable	High

Table 1: Launch Model Comparison

3 The 0-100 Engine Overview

3.1 Actors & Roles

- **Creator.** Creates the agent and initiates the token launch through the 0-100 Engine. Receives a vested allocation after a successful launch and operates the project's Rewards Portal.
- **Contributors.** Deposit SOL during the funding phase in exchange for tickets. Selected contributors receive token allocation proportional to their approved deposits; unselected contributions are fully refundable.
- **Engagers.** Participate in quests and leaderboards within the project's Rewards Portal post-launch. Earn a share of trading fees routed through the income dispatcher.
- **Xyber Protocol.** Provides the standardized launch framework, enforces all parameters onchain, deploys and owns the liquidity pool, and routes trading fees into the dynamic fee distribution system.

3.2 Launch Lifecycle

The 0-100 Engine structures every launch as a deterministic, fully onchain sequence:

1. Creator initializes the launch with precommitted parameters
2. Users contribute through ticketized deposits within wallet caps
3. Protocol resolves overflow using blockhash-based randomness
4. Liquidity is deployed at a probabilistically selected block
5. Post-launch trading fees flow to creators, community, and buybacks

This lifecycle removes timing manipulation, enforces strict per-wallet and hard caps, guarantees refundability of unused contributions, and standardizes liquidity and fee mechanics across all projects.

3.3 Standardized Launch Parameters

All launches follow the same fixed parameter set:

Parameter	SOL	ETH Equivalent	Description
Total Supply	1,000,000,000	–	Hard-coded per launch
Sale Allocation	48.14%	–	Distributed to selected contributors
Liquidity Allocation	41.86%	–	Permanently paired into initial pool
Creator Allocation	10%	–	Linear vesting over 12 months
Creator Self-Snipe	Up to 8 SOL	$\approx$ 0.5 ETH	Each 1 SOL adds 24h vesting
Hard Cap*	450 SOL	$\approx$ 30 ETH	Maximum accepted raise
Minimum Raise*	100 SOL	$\approx$ 6.5 ETH	Required for launch to proceed
Per-Wallet Cap*	1.5 SOL	$\approx$ 0.1 ETH	Maximum user contribution
Ticket Size* τ	0.05 SOL	$\approx$ 0.003 ETH	Unit contribution size
Funding Window	120 hours	–	Deposit/refund period
LP Deployment	+30-60 min	–	Blockhash-triggered pool creation

** Proposed parameters. May be adjusted before mainnet. ETH equivalents approximate. Future changes updated through governance.*

Table 2: Standardized Launch Parameters

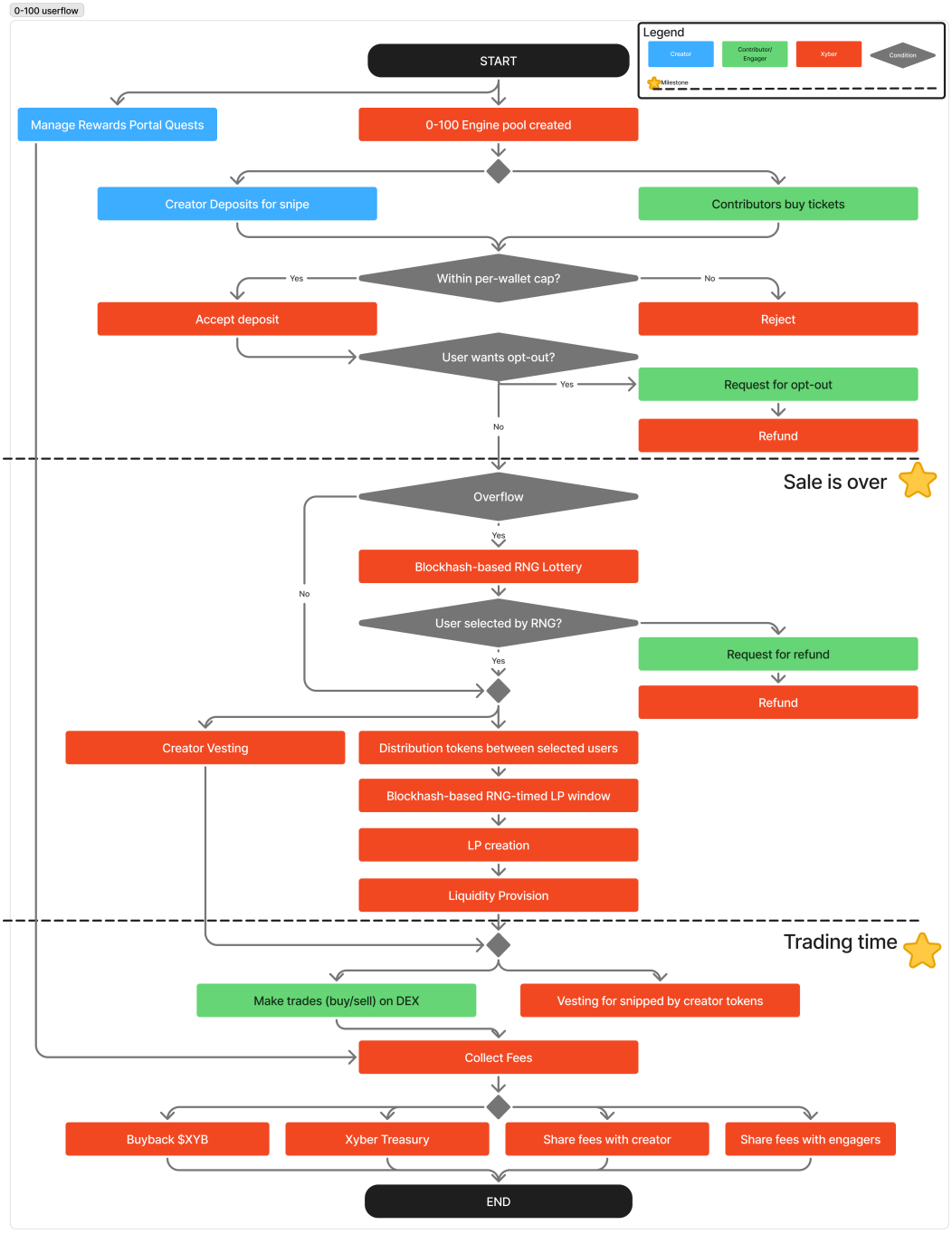

Figure 1: The 0-100 Engine Lifecycle

4 Core Mechanics

4.1 Creator Self-Snipe & Vesting Extension

A launch creator may deposit up to $S_{\max} = 8$ SOL (≈ 0.5 ETH) into their own pool through a dedicated self-snipe instruction. For each SOL deposited, the vesting period for the creator allocation increases linearly by 24 hours per SOL.

Let $s \in [0, S_{\max}]$ be the creator's self-snipe in SOL. The vesting time applied to the creator allocation is:

$$T_{\text{vest}}^{\text{creator}} = 24\text{h} \cdot s \tag{1}$$

No flooring is applied; fractional SOL extends vesting proportionally. Because $s \leq S_{\max}$ and $S_{\max} = 8$, it follows that $T_{\text{vest}}^{\text{creator}} \leq 8$ days.

Self-snipe deposits do not receive public tickets and do not participate in the randomized selection process. However, the deposited lamports are included in the total raise and therefore influence overall capacity and allocation.

4.2 Capped Wallet Funding Pool

Per-wallet cap C is enforced onchain. Deposits must be in exact multiples of the ticket size τ; any non-multiple causes the transaction to revert.

Let a deposit amount be $\text{amount}_i = m_i \cdot \tau$ with integer $m_i \geq 0$. The number of tickets minted in this deposit is:

$$k_i = \min\left(m_i, \left\lfloor \frac{C - \text{deposited}_i}{\tau} \right\rfloor\right) \tag{2}$$

where deposited_i is the wallet's cumulative accepted deposit before this transaction. Receipts are issued to each wallet for the number of tickets minted.

4.3 Overflow + Randomized Allocation

If $\sum n_i \cdot \tau \leq H$, all tickets are approved.

If oversubscribed, define capacity $K = \lfloor H/\tau \rfloor$ and uniformly sample K tickets without replacement from the multiset of all tickets using a blockhash-derived seed.

For wallet i, selected tickets y_i define:

$$a_i = y_i \cdot \tau \qquad \text{(approved amount)} \tag{3}$$

$$r_i = \text{deposit}_i - a_i \qquad \text{(refund)} \tag{4}$$

4.4 Selection Probability

Let total ticket count be N and capacity $K = \lfloor H/\tau \rfloor$. A wallet with n_i tickets has probability of at least one selection:

$$P(Y_i \geq 1) = 1 - \frac{\binom{N-n_i}{K}}{\binom{N}{K}} \tag{5}$$

Tickets are non-transferable (receipt mints are bound to the depositing wallet); they only confer selection and claim rights to that wallet.

4.5 Sale Allocation & Claim

Let A_{sale} denote the Sale bucket (48.14% of total supply), a_i denotes the approved ticket value for wallet i, and each a_j denotes the approved ticket value for wallet j across all selected wallets.

Each approved wallet i receives:

$$\text{tokens}_i = \frac{a_i}{\sum_j a_j} \cdot A_{\text{sale}} \tag{6}$$

Sale tokens have no vesting or cliff; claims open immediately after LP deployment. Creator allocation (10%) follows 12-month linear vesting.

4.6 Randomized LP Creation Window

The creator precommits an LP deployment window of +30-60 minutes after selection finalization. During this window, each eligible blockhash h_k is interpreted as a 256-bit integer.

A blockhash-based RNG triggers LP deployment if:

$$h_k \in \mathcal{R}_{\text{LP}} \tag{7}$$

where $\mathcal{R}_{\text{LP}}$ is a predefined numeric interval unique for the launch (to prevent multi-project collisions).

During the +30-60 min window, LP creation is triggered when the first blockhash that falls within the predefined interval appears. Anyone can execute LP deployment once a valid blockhash is observed.

Initial price is set by the pair ratio:

$$P_0 = \frac{\text{base}_{\text{LP}}}{\text{token}_{\text{LP}}} \tag{8}$$

4.7 Built-in Anti-Snipe

Because allocations are fixed before trading, and LP timing is determined by a blockhash within a bounded window, coordinated sniping has limited effect on initial distribution. The randomized LP trigger further prevents predictable listing moments.

4.8 Post-Listing Fees & Distribution

DEX trading fees from the protocol-owned LP position are harvested by the engine program via the `income_dispatcher` authority. Let F_{DEX} be accumulated fees. Distribution follows a market-cap indexed dynamic split:

$$F_{\text{DEX}} = F_{\text{Creator}} + F_{\text{Community}} + F_{\text{Xyber}} \tag{9}$$

Platform-fee buybacks: Protocol platform fees designated for buybacks are used to market-buy $XYBER (80% from all protocol fees). Routing is entirely onchain and publicly auditable.

5 Token Supply & Economic Flows

The 0-100 Engine enforces a uniform supply structure and post-launch economic flow that applies identically to every project. All allocations, vesting rules and fee mechanics are predetermined by the Engine preset and executed onchain without creator-side tuning.

5.1 Standardized Supply Split

Every launch follows the same fixed supply distribution:

- **48.14% – Sale Allocation.** Distributed to selected contributors in proportion to their approved deposits. Sale tokens unlock immediately after LP deployment.
- **41.86% – Liquidity Allocation.** Permanently paired with the raised SOL to form protocol-owned liquidity. LP tokens remain locked under program authority.
- **10% – Creator Allocation.** Subject to a 12-month linear vesting schedule. The vesting timeline may be extended through creator self-snipe deposits.

5.2 Dynamic Fee Model

Once trading begins, fees from the protocol-owned LP position are harvested onchain and routed through a dynamic distribution model indexed by fully diluted market capitalization.

Step	Market Cap (SOL)	Market Cap (ETH)	Creator %	Xyber %	Community %
1	0 – 500	0 – 20	25	60	15
2	501 – 1,500	21 – 65	56	30	14
3	1,501 – 4,000	66 – 175	53	34	13
4	4,001 – 10,000	176 – 435	51	37	12
5	10,001 – 20,000	436 – 875	49	40	11
6	20,001 – 30,000	876 – 1300	47	43	10
7	30,001 – 50,000	1301 – 2200	44	47	9
8	50,001 – 70,000	2201 – 3050	41	51	8
9	70,001 – 100,000	3051 – 4350	37	56	7
10	$>$ 100,000	$>$ 4350+	34	60	6

Table 3: Dynamic Fee Model

5.3 $XYBER Buybacks & Treasury Flows

The protocol's share of fees is divided into two channels:

- **80% – Continuous $XYBER Buybacks.** Used to market-buy $XYBER via the designated DEX pair. All buyback operations are fully onchain and verifiable.
- **20% – Xyber Treasury Growth.** Accumulated in the protocol vault to support long-term incentives and ecosystem development.

6 On-Chain Architecture

The 0-100 Engine is a fully permissionless launch system composed of two onchain programs. Once a launch is initialized, all steps (funding, selection, minting, liquidity creation, and fee routing) are driven exclusively by program logic and protocol-owned authorities. No creator, admin, or off-chain service can override the preset or seize user funds.

6.1 Contract Surface

6.1.1 engine program

Core execution layer for a launch. It:

- initializes launches under a single immutable preset (caps, τ, supply split, vesting)
- handles deposits, withdrawals and ticketization into the protocol-owned escrow account
- finalizes selection using a blockhash-derived permutation of public tickets
- mints the full token supply, initializes metadata and creates the Raydium/UniSwap CLMM pool
- pairs SOL/ETH with the Liquidity Allocation and opens claims for users, creator and team
- harvests DEX fees and applies the dynamic fee split

All state transitions are validated onchain; any violation of invariants causes the transaction to revert.

6.1.2 income_dispatcher program

Minimal coordinator that:

- stores the platform income configuration
- exposes a protocol-owned authority allowed to call the engine's fee-collection instruction via CPI

It cannot change launch parameters or move user funds by itself; it only proves that fee collection is authorized.

6.2 Permissionless Trust Model

The Engine is designed so that safety and fairness follow from code, not from trusted operators:

- **Permissionless entrypoints.** Any wallet can invoke public instructions (deposit, withdraw, roster finalization, LP creation, fee collection) as long as time and state checks pass. There are no whitelists or privileged callers for the launch flow.
- **Immutable preset.** Parameters such as hard cap, minimum raise, per-wallet cap, ticket size, supply split, vesting rules and fee schedule are enforced by the engine program. The creator cannot tune them per launch and they become immutable once the launch is initialized.
- **Non-custodial PDAs.** All SOL/ETH and token flows go through PDAs (not EOAs). The `escrow_authority` PDA signs transfers and mints, the `income_dispatcher_authority` PDA is the only allowed signer for CLMM fee collection. No private key can unilaterally redirect assets.
- **Scoped admin multisig.** A 2-of-3 or 3-of-3 admin set controls only the global `EngineConfig` (e.g. XYBER mint, creation fee, treasury address). Admins cannot modify an existing launch, its selection results, LP parameters, vesting or refunds.

- **Deterministic randomness and ordering.** Selection and LP timing rely on recent blockhashes from the `SlotHashes` sysvar within bounded windows. This removes reliance on off-chain randomness and prevents ex post rescheduling of listing events.

Together, these properties make each 0-100 launch a deterministic, auditable state machine with permissionless access and strictly limited governance surface.

7 Security & Audit Status

The 0-100 Engine is undergoing a comprehensive security audit by a tier-1 blockchain security firm. The audit scope covers:

- Engine program logic and state transitions
- PDA authority separation and fund flows
- Randomness derivation and manipulation resistance
- Fee distribution and buyback routing

The full audit report will be published prior to the first launch.

All smart contract code will be open source and available for public review following release.

8 Limitations & Risks

The 0-100 Engine minimizes discretionary risk but does not eliminate all forms of uncertainty:

- **Market risk.** The Engine guarantees fair distribution and protocol-owned liquidity. It does not guarantee token price performance post-launch.
- **Smart contract risk.** Despite audits, undiscovered vulnerabilities may exist. Users should not contribute more than they can afford to lose.
- **Regulatory uncertainty.** Token launches may be subject to evolving legal frameworks across jurisdictions. Participants are responsible for compliance with applicable laws.

9 Conclusion

The 0-100 Engine replaces discretionary launch mechanics with deterministic, verifiable protocol rules. Fairness constraints are encoded directly into the execution layer. The structural advantages that have historically favored bots, insiders, and sophisticated actors are eliminated by design.

Every launch follows the same preset. Every allocation is auditable. Every fee flow is onchain. No exceptions. No negotiations. No backdoors.

This is not an incremental improvement. It is a structural reset.

Trust becomes a property of code, not reputation. And for the first time, the rules apply equally to everyone.

References

1. Solana Foundation. *SlotHashes Sysvar Documentation.* https://docs.solana.com
2. Raydium Protocol. *CLMM Technical Specification.* https://docs.raydium.io