KOmarkets

PROTOCOL WHITEPAPER · V1.0

A Kaspa-native protocol for on-chain prediction markets.

Binary prediction markets settled natively on Kaspa Layer 1 — covenant-locked stakes, oracle-witnessed resolution, and pro-rata payouts with no bridge, no wrapper, and no custodian.

KOMARKETS.IO · HELLO@KOMARKETS.IO

TESTNET-12 LIVE · MAINNET TARGET Q4 2026

BUILT IN ATHENS · POWERED BY KASPA MAY 2026

Contents

01

Abstract

KOM is a binary prediction market protocol built natively on Kaspa Layer 1. Users bet on yes-or-no outcomes by locking KAS into per-market covenant addresses; when a market closes, an oracle-witnessed transaction sweeps the losing side's pool pro-rata to the winners. There is no off-chain accounting, no custody intermediary, and no bridge or wrapped token. The pool visible on chain is the pool that pays out.

This paper describes the protocol's market model, its two on-chain primitives, the bet lifecycle, and the oracle and resolution design — and argues that Kaspa's one-second finality, MEV-resistant blockDAG, and native covenant scripting make it the first base layer on which a real-time, trust-minimized prediction market can actually be built.

02

The problem

Prediction markets are among the most useful financial instruments ever devised. They aggregate dispersed information into a single probability more accurately than polls, faster than journalists, and more honestly than pundits. **Polymarket proved the demand** at scale; **Kalshi proved a regulatory path** exists. Yet both depend on trust: in a custodian holding funds, in a centralized matching engine setting prices, and in a platform that retains the ability to change the rules after a bet is placed.

Crypto-native attempts have repeatedly tried to remove that trust — on Ethereum, Polygon, and a succession of L2s. Most fail for the same structural reason: **the chain underneath was never designed for sub-minute markets that settle in real time.** Block times are too slow to guarantee a late bet lands before resolution. Gas costs consume small stakes. A globally observable mempool lets MEV bots front-run high-information resolution transactions. And the standard workaround — moving to an L2 — reintroduces trust in a sequencer, a bridge, and an upgrade authority. For a protocol whose entire premise is trust-minimization, the cure is worse than the disease.

> **The category never lacked demand. It lacked a base layer fast enough, cheap enough, and trust-minimized enough to host it.**

03

Why Kaspa

Kaspa solves the underlying problem rather than layering trust on top of it. Five properties matter directly to a prediction-market protocol:

One-second finality

Kaspa's GhostDAG protocol confirms transactions in roughly one second — actual confirmation, not optimistic finality with a challenge period. A bet placed thirty seconds before expiry must be on chain before the resolver runs; Ethereum's twelve-second blocks plus reorg risk make that impossible to guarantee, while Kaspa makes it routine.

No MEV by design

On chains with a single global mempool, bots observe a pending transaction and front-run it. For prediction markets this is catastrophic: the moment an oracle's payout transaction is visible, arbitrage can be extracted from the underlying market. Kaspa's parallel-block architecture has no canonical pre-confirmation sequence to front-run — blocks are produced concurrently and merged deterministically by the DAG.

Native Layer-1 covenant scripting

KOM does not run on a virtual machine bolted onto Kaspa. Silverscript is Kaspa's native covenant language; it compiles to script bytecode executed as part of ordinary transaction validation. There is no Solidity, no EVM, and no rollup. The trust model is Bitcoin's UTXO model, with the script language extended just far enough to enforce per-market covenant rules.

No bridge, no wrapper, no L2

Every prediction-market effort that touches Kaspa today does so through a bridge or a wrapped token. Bridges have lost more value than any other vector in crypto; wrapped tokens are IOUs dependent on a wrapper's solvency. KOM uses native KAS directly — the KAS in a market's pool is the KAS paid to winners.

A blockDAG built for throughput

Settlement is bursty: when a popular market closes, many claims may want to land in the same window. A sequential chain spikes fees under that load; Kaspa's parallel DAG absorbs it.

> **You can build a prediction market on any chain. You can only build this protocol — sub-minute markets, native UTXO custody, no bridges, no L2 — on Kaspa.**

Protocol overview

A KOM market poses a binary question with a deadline — for example, *will KAS be above $0.0304 at 09:35 UTC?* Bettors lock KAS into one of two per-market covenant addresses corresponding to YES and NO. These addresses are the market's pool, fully visible on chain. When the deadline passes, an authorized oracle reads the outcome from cross-checked data and broadcasts a single transaction that distributes the combined pool to the winning side in proportion to each winner's stake.

Three properties define the protocol:

- **On-chain, end to end.** Every bet and every resolution is a Kaspa transaction; pools are real UTXOs verifiable in any explorer. The protocol needs no database for custody — the chain is the ledger of

record.

- **Built for real-time markets.** Auto-rolling 5-, 15-, and 60-minute markets across KAS, BTC, and ETH refresh every minute and settle within seconds of expiry.
- **No bridge, no wrapper, no IOU.** Native Silverscript covenants enforce the bet contract at the script level; native KAS is the only asset involved.

05

The market model

KOM is a **parimutuel pool**, not an order book. Polymarket matches a buyer of YES shares with a seller; each trade needs a counterparty, and an off-chain engine sets the price. KOM instead collects all YES stakes into a YES pool and all NO stakes into a NO pool. A bettor wagers against the pool, not against a person, and the implied odds are simply the ratio of the two pools — fully transparent and fully on chain.

Payout

At settlement the entire combined pot, less only the network transaction fee, is distributed to the winning side in proportion to stake:

```
totalPot = winPool + losePool
payable  = totalPot - txFee   // ~0.0002 KAS, on-chain only
payout   = (yourStake / winPool) × payable
// KOM takes no protocol rake on testnet today
```

A winner's profit is therefore precisely their proportional share of the losing side's stake. There is no leverage, no liquidation, and no margin: the maximum loss is exactly the amount signed into the bet.

The no-loser case

A pooled model must answer one edge case cleanly: what if every bettor was on the winning side and the losing pool is empty? Then `losePool = 0`, so `payable = winPool - txFee`, and each winner simply receives their own stake back, less a negligible shared network fee. There are no winnings — because no one was wrong — but there is no loss and no rake on a refund. The same holds for a sole bettor. This falls directly out of the model being honest: in a real parimutuel pool, winnings can only come from the losing side's stake.

> **The pool you can see on chain is exactly the pool that pays out. No phantom fees, no house edge on an uncontested bet.**

06

On-chain primitives

A market is built from two primitives: **BetLock** covenants that hold the stake pool, and **ShareToken** UTXOs that prove ownership of individual positions. They are separated because the pot must be sweepable by the resolver in a single transaction, while ownership must remain attributable per bet — one pot, many receipts.

BetLock — the per-side pool

Each market deploys a fresh pair of BetLock covenants, compiled at creation time from three constructor arguments: a `marketId`, a `sideByte` (1 for YES, 2 for NO), and the oracle's x-only schnorr `oraclePubkey`. The compiled output is a 65-byte script producing a unique address; every bet on a side lands at the same address. The script enforces a single rule: the UTXO can be spent only by a transaction carrying a valid oracle signature attesting to the outcome. No oracle signature, no spend.

ShareToken — the per-bet receipt

Every bet also creates a 2-KAS covenant UTXO — Kaspa's minimum dust — at an address keyed by `(ownerPubkey, marketId, side, shareAmount)`. This is proof-of-bet, spendable either by the owner's schnorr signature (to redeem winnings or list the position for sale) or by the settlement transaction at maturity. Because the receipt is on chain and bound to the owner's key, a position is recoverable from any wallet without KOM's frontend; because it is independent of the pool, ownership can be transferred on a secondary market without touching the stake.

A single bet transaction

Placing a bet constructs one transaction with three outputs — the stake to the BetLock address, a 2-KAS receipt to the ShareToken address, and change back to the bettor. The transaction is signed entirely in the browser; the server only forwards the signed bytes. The private key never leaves the device.

07

Bet lifecycle

A bet proceeds through six steps, all observable in the browser's network tab and all ending in a single on-chain transaction:

STEP	ACTION
1 · Address derivation	Client requests the BetLock YES/NO addresses for the market and selects the chosen side.
2 · ShareToken compile	Server compiles the ShareToken script from owner pubkey, marketId, side, and share amount.
3 · UTXO fetch	Client finds the smallest set of its own wallet UTXOs covering stake + dust + fee.
4 · Construction	Client builds the three-output transaction via kaspa-wasm. All signing is local.
5 · Submission	Signed hex is forwarded to kaspad's submitTransaction RPC; the txId is returned.

STEP	ACTION
6 · Position record	Server appends best-effort metadata. The on-chain transaction is canonical — a server crash does not invalidate the bet.

Market state machine

A market moves through five states: **Created** when the cron compiles its artifacts; **Open** once published and accepting bets; **Closed** when its deadline passes; **Settling** when the next cron tick computes the outcome and broadcasts payout; and **Resolved** once payout confirms. After 24 hours, resolved markets are pruned to an archive.

08

Oracle & resolution

A prediction market is only as trustworthy as the mechanism that decides who won. KOM's resolution layer is built on a single principle: **the oracle should refuse to sign unless the data behind the outcome is genuinely reliable.**

Two independent sources, cross-checked

For crypto markets, the closing measurement is cross-checked across two independent exchanges — **Coinbase and Kraken** — and the oracle acts only on a price the two venues agree on. A single exchange can wick, lag, or be briefly manipulated; two unrelated venues agreeing is far harder to fake. For weather markets, the equivalent source is a meteorological feed read once at close to settle every temperature bucket in that window.

The confidence gate

Before signing, the measurement passes a confidence gate. If the data source has fallen back to anything other than a live, cross-checked feed — a cached value, a demo source, or a single venue that lost its counterpart — the gate blocks the signature. No live data, no resolution: the system fails closed, preferring to wait over resolving on a number it does not trust.

Propose, wait, finalize

Resolution runs in three phases. The oracle **proposes** an outcome from cross-checked data; a **delay window** elapses before money moves; and only then does the oracle **finalize**, signing the settlement transaction that the BetLock covenant requires. The covenant releases nothing without that signature.

Trust model, stated honestly

On testnet today the oracle is a single authorized key — the holder of the schnorr private key matching the `oraclePubkey` baked into each BetLock. This is an honest single-oracle model, gated by cross-checked data and a delay window, but ultimately one signer; the covenant is oracle-gated, not trustless. The mainnet path replaces the single key with a **multi-signature oracle** of independent price providers and a formal dispute window. Because the covenant only checks a signature against a baked-in pubkey, this is a change in who holds the key, not a rewrite of the contract.

09

Architecture

KOM is four components, each with a clear ownership boundary:

LAYER	OWNS	DOES NOT OWN
Browser	Wallet keys (in IndexedDB, never sent), tx signing, live price feed, UI state	Anything that must survive a refresh
Next.js server	Market registry, position store, address derivation, tx forwarding	Wallet keys
kaspad	Source of truth for every balance and transaction	Application metadata
launchd cron	Heartbeat for market creation and settlement	State — it only calls HTTP endpoints

The browser never sees the server's keys; the server never sees the user's keys; the chain mediates everything.

> **The UI is a thin client; the chain is the source of truth. If the indexer and the chain disagree, the chain wins — and the indexer is the bug.**

10

Roadmap

KOM is live on Kaspa testnet-12 today, running the full bet-and-settle loop end to end. The path to mainnet has four checkpoints, each gated on evidence rather than a calendar.

Q2 2026 — Testnet hardening (current)

Auto-rolling catalog live; bet flow fully on chain; resolver on a 60-second cron; frontend in active polish. Goal: zero unrecoverable bugs across 1,000 simulated bets before locking the contract.

Q3 2026 — Audit and mainnet candidate

External audit of the Silverscript covenants by a Kaspa-native security firm; migration of position metadata to a database; removal of developer-mode logging; oracle architecture sign-off.

Q4 2026 — Mainnet launch

Deployment to Kaspa mainnet with real KAS. Initial categories crypto, then weather, then sports. A public trading API ships alongside launch for programmatic participation by bots and market makers.

2027 — Protocol expansion

Multi-asset and scalar markets beyond binary YES/NO; community-voted market listings; and opening the protocol to third-party frontends, with KOM as the settlement layer.
11

Risks & honest disclosures

- **Single oracle today.** v1 is gated by cross-checked data and a delay window but is ultimately one signer. The multi-sig + dispute design will be locked and published before mainnet.
- **Resolver liveness.** If the cron halts, no markets settle and KAS sits in the pools; because the covenant permits the oracle to settle any time after close, a halt is recoverable, not a loss. A multi-sig fallback is planned for mainnet.
- **Testnet resets.** Testnet bets are not recoverable across resets; protocol code, market designs, and wallets are unaffected.
- **Audit pending.** The covenants are unaudited until Q3 2026; mainnet is explicitly gated on audit completion.
- **Metadata is best-effort.** The position store is convenience metadata only; the on-chain transaction is always the canonical record.
