TECHNICAL WHITEPAPER | 26 JULY 2026

MOROS

Private prediction markets with verifiable USDC settlement on Stellar

A deployed protocol for encrypted batch positions, shielded Circle USDC notes, permissionless pooled liquidity, LMSR pricing, and oracle-gated settlement on Stellar mainnet.

STELLAR MAINNET | CIRCLE USDC | BN254 GROTH16 | LMSR BATCHES

Limited beta. Current deployment uses one Reflector provider family, a single-VM coordinator, and a single-contributor proof setup. Internal review only.

moros.fun

Document control

Field	Value
Title	Moros Technical Whitepaper
Publication date	26 July 2026
Network	Stellar mainnet
Status	Limited beta technical description
Canonical source commit	b8108b94376f665000c361049021c9be5b0cf138
Deployment manifest	deployments/private-mainnet.json
Proving manifest	deployments/private-mainnet-proving.json
Live application	https://moros.fun

Authority rule. The deployed manifests and implementation at the recorded source commit take priority over earlier specifications, prototypes, and planning documents.

Contents

The paper moves from market mechanics to privacy, proofs, resolution, security, deployment, and reference appendices.

References use numbered citations. Contract identifiers and hashes appear in Section 17 and the appendices.

Abstract

Prediction markets turn dispersed information into prices, but public execution also turns each participant's intent into observable data. A public ledger can expose the account, selected outcome, position size, timing, and repeated strategy of every trader. Moros is a binary prediction-market protocol on Stellar that separates public market solvency from individual position disclosure. It combines a logarithmic market scoring rule, short uniform-price execution epochs, a shared shielded Circle USDC vault, browser-generated Groth16 proofs, encrypted orders, permissionless pooled liquidity, and resolver-gated settlement.

Users deposit Circle USDC at a public Stellar boundary and receive reusable private notes. An order consumes a private balance note, commits to a market position, and encrypts its YES or NO quantity in the browser. Up to eight orders are collected in a 60-second epoch. The coordinator decrypts the accepted orders, proves the aggregate batch statement, and submits one state transition. The market applies the aggregate LMSR cost and updates its visible probability only after the batch executes. At resolution, winners prove ownership of eligible position notes and create new private USDC notes. Unexecuted orders and void markets use proof-bound refund paths. Liquidity providers hold private pool-share notes whose value reflects isolated market results and vested fees.

Moros is deployed on Stellar mainnet in limited beta. The current release supports 40 crypto, foreign-exchange, and gold price routes through one Reflector provider family. It does not enable event markets. The mainnet release uses a single-VM coordinator that can recover individual order values, a single-contributor Groth16 setup, and contracts that have completed internal review but not an independent external audit. These boundaries are part of the protocol description, not footnotes.

> *Technical scope. This paper describes the Moros implementation and mainnet deployment dated 26 July 2026. It is not legal, investment, or financial advice. Prediction-market positions and liquidity provision can lose value.*

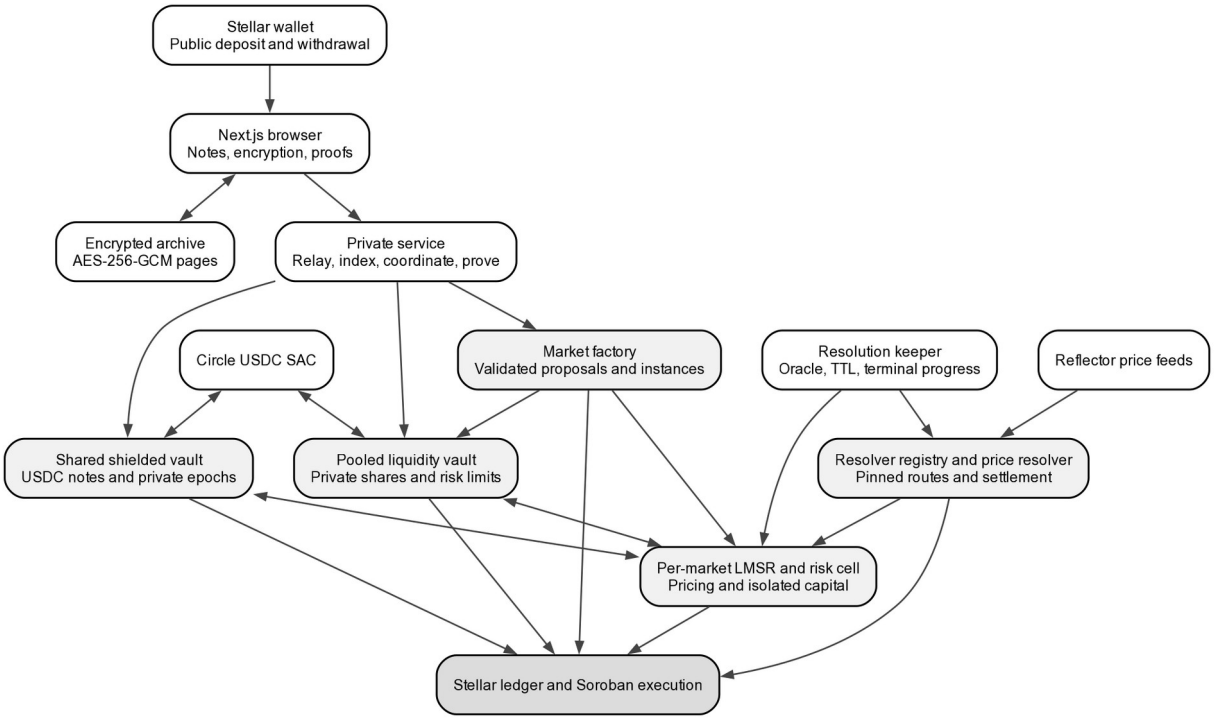

Figure 1. Moros system layers and settlement boundaries.

1. Motivation

1.1 Prediction markets as information systems

A binary prediction market defines a proposition that will eventually be true or false. A YES claim pays one collateral unit if the proposition is true, while a NO claim pays one collateral unit if it is false. Prediction markets can aggregate dispersed information, but a trade price is not an unconditional probability estimate [1]. Its interpretation depends on market structure, fees, liquidity, participant beliefs, and selection effects. Empirical work on Kalshi shows that prices can be informative while still exhibiting systematic biases and different outcomes for liquidity makers and takers [3].

The protocol problem is therefore broader than displaying a probability. A usable market must define:

- The exact proposition and settlement rule.
- The pricing mechanism and inventory state.
- The source and freshness of resolution evidence.
- The collateral and worst-case liability.
- The execution rule when order flow is sparse or one-sided.
- The fee rule and recipients.
- The claim, refund, and failure-recovery paths.

Moros treats these as one lifecycle. Market creation is not complete merely because a title is stored. A proposal becomes tradeable only when a supported resolver route is pinned, the isolated liquidity target is funded, the LMSR loss bound is covered, and the market is registered with the shared shielded vault.

1.2 The disclosure problem

Transparent on-chain markets can expose more than settlement requires. A transaction may reveal the wallet, side, amount, time, and price impact before or during execution. A sequence of positions can reveal risk appetite and strategy. Other participants can copy a large trader, infer inventory, or react before that trader completes a program.

Moros does not attempt to hide the existence of a market or its aggregate state. Public probabilities, liquidity, expiry, resolution source, executed aggregate quantities, and final outcome are necessary for a verifiable market. The design instead asks a narrower question:

> *Can the system verify that every order is funded, every batch is correctly priced, and every payout is solvent without publishing individual position ownership, side, and quantity as plaintext?*

The implementation answers this with shared private notes, encrypted order values, batch execution, and proof-bound settlement.

1.3 Design objectives

Moros is built around seven objectives:

1. Solvency before privacy. A private state transition must never create unbacked USDC or outcome claims.
2. Reusable private collateral. One public deposit should support many internal bets, LP actions, claims, refunds, and transfers.
3. Batch fairness. Orders accepted into one epoch receive one execution price per side, independent of submission order inside that epoch.
4. Permissionless creation. A user may propose a supported market without personally funding the full market-maker loss bound.
5. Isolated risk. Pooled capital may allocate across markets, but each market receives a separate risk cell and cannot spend another cell's reserve.
6. Resolver-gated scope. A category is available only when its resolution and terminal recovery operations exist.
7. Honest privacy. The system states what is public, what is encrypted, who can decrypt, and where timing or metadata correlations remain.

2. System model

2.1 Participants

Participant	Primary responsibility	Authority or exposure
Trader	Holds private notes, chooses side and quantity, generates proofs	Controls own note secrets; public wallet appears only at deposit or withdrawal boundary

Participant	Primary responsibility	Authority or exposure
Market proposer	Defines a supported asset, strike, expiry, liquidity tier, fee, and rules hash	Creator address and proposal are public
Liquidity provider	Deposits private USDC into the pooled vault and holds private share notes	Bears market-maker profit and loss within pool policy
Browser prover	Selects notes, encrypts order values, constructs witnesses, and generates user proofs	Sees the user's plaintext private state during the unlocked session
Private service	Relays actions, indexes outputs, coordinates epochs, proves batches, allocates liquidity, and harvests terminal cells	Sees request metadata and target markets; current coordinator can decrypt individual orders
Resolution keeper	Submits oracle-backed resolution and contract TTL transactions	Cannot choose a result outside resolver contract rules
Stellar contracts	Enforce note transitions, pricing, allocation, settlement, and route policy	Hold USDC and public protocol state
Reflector	Supplies public price observations	Current release depends on one provider family
Supabase	Stores public market metadata and opaque encrypted private pages	Can observe access metadata but should not receive private archive plaintext
Stellar network	Orders and finalizes transactions under SCP	Makes contract calls, events, and token boundaries publicly inspectable

2.2 Deployed components

Moros uses six shared contracts and two factory-deployed contracts per market:

Contract	Scope	Responsibility
Groth16 verifier	Shared	Stores immutable circuit verification keys and verifies typed proofs
Shared shielded vault	Shared	Holds USDC, maintains note commitments and nullifiers, accepts orders, executes batches, and processes claims and refunds
Pooled liquidity vault	Shared	Issues LP shares, manages idle capital, enforces risk limits, allocates to markets, and harvests terminal capital
Market factory	Shared	Validates proposals and deploys approved market and liquidity-cell WASM
Resolver registry	Shared	Maps assets to enabled resolver routes and revisions
Price resolver	Shared	Reads approved Reflector observations and resolves or voids eligible markets
LMSR market	Per market	Prices aggregate orders, tracks YES and NO inventory, enforces solvency, and settles outcome shares
Market liquidity vault	Per market	Isolates one market's allocated capital and terminal accounting

The application and services load these addresses from a network-scoped deployment manifest. Contract identifiers are not independently copied through the codebase.

2.3 Off-chain services are progress agents

Stellar contracts do not wake themselves when a clock reaches expiry. Every state change requires a transaction. Moros services supply transactions for actions such as:

- Allocating available pooled liquidity.
- Activating fully funded proposals.
- Sealing a full or expired batch epoch.
- Generating and submitting an aggregate batch proof.
- Opening the next epoch.
- Resolving an expired market.
- Moving a stale market into its void path.
- Harvesting terminal market capital.
- Extending contract instance and code time to live.

These services improve liveness, but contract validation remains the authority. A correctly formed proof-bound user action may be relayed without granting the relayer control over note values. Selected lifecycle steps are permissionless at the contract level, so a different operator can progress them if the primary service stops. Availability is nevertheless weaker than decentralization because the current coordinator secret and production automation are concentrated on one VM.

3. Market lifecycle

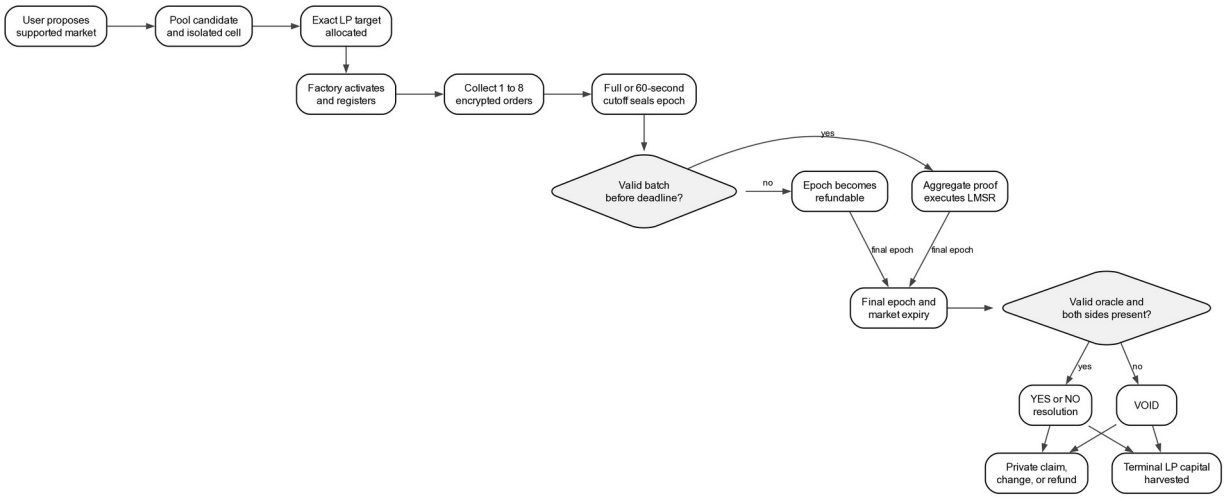

Figure 2. Market proposal, funding, trading, and terminal recovery.

3.1 Proposal

A connected Stellar account proposes a binary threshold market. The proposal includes:

- An enabled underlying asset.
- A USD strike price.
- An exact settlement time.
- A liquidity target from an approved tier.
- An execution fee not exceeding the protocol cap.
- A rules hash.
- A pinned resolver route and route revision.

The factory validates collateral, timing, fee, route status, risk group, approved WASM hashes, and proposal uniqueness. Proposal creation does not transfer the creator's USDC into the market. The creator address remains public because the proposal is an on-chain act.

3.2 Funding candidate

The factory deploys or identifies an isolated liquidity cell and registers a candidate with the pooled liquidity vault. The candidate records a queue sequence, proposal identifier, asset, risk group, target assets, and funding deadline.

The pool may allocate only if the candidate remains valid and the allocation satisfies:

- Total deployed-capital limit.
- Minimum idle-capital reserve.
- Per-market exposure limit.
- Per-risk-group exposure limit.
- Maximum active-allocation count.
- Available idle USDC.

The deployed mainnet tiers are 20, 50, and 100 USDC. The pool never partially funds a candidate. It transfers the exact target or leaves the candidate pending.

3.3 Activation

After the isolated cell receives its target, any eligible caller can progress activation. The factory deploys the approved LMSR market instance, binds it to the shared shielded vault as sole private batcher, binds the isolated liquidity vault, pins the resolver and rules hash, and verifies that the LMSR loss bound is funded.

Activation fails atomically if any required relationship is wrong. A public metadata-listing failure does not invalidate a correct on-chain deployment and may be retried without repeating the capital transfer.

3.4 Trading epochs

An active market maintains one current private epoch. The mainnet policy is:

- Collection duration: 60 seconds.
- Maximum accepted orders: 8.
- Quantity per order: 1 to 1,000 whole positions.
- Minimum side count: 0.
- One-sided batches: allowed.
- Grace period before refundability: 600 seconds.

An epoch seals when it reaches eight accepted orders or its cutoff time. A nonempty sealed epoch may execute with only YES orders or only NO orders. An empty epoch does not change market state.

Orders inside one executed epoch receive the same per-unit charge for their side and the same per-unit execution fee. A later epoch sees the inventory produced by earlier executed epochs.

3.5 Close and final batch

At market expiry, new orders stop. The final accepted epoch must execute or become refundable before price resolution proceeds. This ordering prevents the resolver from finalizing a market while accepted private orders remain economically ambiguous.

3.6 Resolution

For an enabled price market, the resolver reads the pinned Reflector route at the configured settlement time. YES wins when the validated settlement value is equal to or above the strike. NO wins when it is below the strike.

The resolver rejects missing, stale, malformed, or invalid observations. If the oracle remains unavailable through the configured recovery interval, the stale-market path can void the market. An ordinary binary resolution also becomes a void if one terminal inventory side is zero. This means a one-sided epoch may execute and move the price, but a market in which only one side ever accumulates inventory does not settle as a normal winner-takes-outcome market.

3.7 Claims, refunds, and terminal capital

Resolution does not push funds automatically to public wallets.

- An executed winning position creates a claimable private credit.
- An executed losing position has no winning payout, but its unused order budget was already recoverable through execution change.
- A void position can recover the protocol-defined refundable value.
- An unexecuted order in a refundable epoch can recover its full locked order budget.
- LP capital and vested fees return through terminal settlement and pool harvesting.
- A user may keep proceeds private or later withdraw through the public Stellar boundary.

Every claim or refund consumes a terminal nullifier so the same position cannot use more than one terminal path.

4. Binary contracts and LMSR pricing

4.1 Outcome claims

Let the terminal event be `X`, where `X = 1` for YES and `X = 0` for NO. One YES share pays:

$$Y = X$$

One NO share pays:

$$N = 1 - X$$

Before fees, complementary shares have a combined terminal payout of one collateral unit.

4.2 Cost function

Moros uses a binary logarithmic market scoring rule. Let `qY` and `qN` be outstanding YES and NO inventory and let `b > 0` be the liquidity parameter. The cost function is:

$$C(qY, qN) = b \ln(\exp(qY / b) + \exp(qN / b))$$

The instantaneous YES price is:

$$pY = \exp(qY / b) / (\exp(qY / b) + \exp(qN / b))$$

and:

$$pN = 1 - pY$$

At symmetric initialization, `pY = pN = 0.5`. The LMSR worst-case market-maker loss relative to initial cost is bounded by:

$$Lmax = b \ln(2)$$

This bounded loss is why the factory can determine a minimum activation subsidy from `b` [2].

4.3 Aggregate batch charge

For a batch with aggregate inventory increments `dY` and `dN`, the market charge is:

$$dC = C(qY + dY, qN + dN) - C(qY, qN)$$

The contract rounds the aggregate charge up to a USDC atomic unit. It then computes an average YES price by integrating the LMSR softplus along the aggregate inventory change. The complementary average NO price is one minus that value. Weighted side costs are allocated so their sum equals the aggregate market charge.

Within a side, the atomic side cost is divided by that side's total quantity. The bounded remainder is recorded as a rounding contribution rather than silently retained as revenue. The batch proof binds:

- Pre-batch and post-batch inventory.
- Pre-batch and post-batch YES probability.
- Aggregate YES and NO quantities.
- Average execution price for each side.
- Aggregate market charge.
- Per-unit side charges.
- Rounding contribution.
- Fee escrow and its allocation.

4.4 Uniform execution

Uniform execution means submission order inside an epoch does not determine price. Every YES unit in the same epoch receives one YES charge, and every NO unit receives one NO charge. YES and NO charges differ because the sides have complementary payoff and inventory exposure.

Batch execution has a deliberate information delay. A pending order does not immediately move the displayed probability. The visible probability changes only when the aggregate epoch executes. If one YES order is the only order during a 60-second collection period, that nonempty one-sided epoch can execute after its cutoff and move the probability. Prices do not move on an empty timer.

4.5 Price-movement bound

The factory-configured mainnet policy limits the absolute YES probability movement of one batch to `0.25` in Q32.32 fixed-point representation. A quote exceeding the bound fails. This limits sudden inventory jumps and prevents a batch from traversing too much of the curve in one transition.

4.6 Fixed-point implementation

The LMSR contract implements cost, probability, logarithm, exponential, and average-price calculations in signed 128-bit Q32.32 arithmetic. Financial bounds and checked arithmetic prevent unsupported ranges from reaching the approximation. Atomic USDC conversion uses explicit rounding direction:

- Charges round in the solvency-preserving direction.
- Withdrawals and payouts use the contract-defined conservative direction.
- Rounding differences are named accounting entries.

The contract checks projected assets against both YES and NO scenario liabilities before applying a private batch.

5. Execution fees and protocol revenue

5.1 Probability-sensitive fee

For lot size `L`, average batch probability `p`, and fee rate `r`, the per-position fee is based on:

$$\text{fee} = L\ r\ p\ (1 - p)$$

The curve is symmetric across complementary outcomes and largest near 50 percent probability. It decreases as the market approaches certainty. The deployed contract converts the result to atomic USDC and rounds the per-position fee up.

The application proposes a 2 percent fee by default. The factory rejects fees above 10 percent. The exact fee is part of the market configuration and must be read from the market rather than inferred from application copy.

5.2 Escrow and vesting

Execution fees are conditional until a normal result is final. Fee escrow first reimburses the explicit rounding contribution. The remaining distributable amount is split:

- 80 percent to liquidity providers.
- 20 percent to the protocol.

Void and unexecuted-refund paths do not create distributable execution revenue. The design avoids calling arithmetic residue a fee.

5.3 Economic interpretation

The fee compensates two different forms of capital:

- LP capital bearing the bounded but real LMSR inventory risk.
- Protocol operation, development, and treasury activity.

LP return can be positive or negative even before fees. Fee share does not guarantee principal or yield. Protocol revenue depends on executed, non-void market activity and is not a charge on gross pool deposits.

6. Pooled liquidity and isolated risk

6.1 Why market creators do not fund the market

Requiring every market creator to supply the full LMSR loss bound would make creation capital-intensive and would confuse proposal quality with creator wealth. Moros separates proposal authorship from market-making capital.

A creator proposes a supported market. Liquidity providers deposit into one Moros pool. The pool allocates eligible idle capital to isolated market cells under deterministic contract policy. Moros itself may participate as an LP, but does not receive a special allocation right.

6.2 Private pool shares

An LP deposit consumes private USDC notes and creates a private pool-share note. The public pooled vault tracks aggregate assets and total shares, while personal share ownership remains encoded in shielded notes.

Let `A` be the applicable pool net asset value and `S` the total pool shares. A deposit of `a` assets mints shares according to the contract's virtual-asset and virtual-share formula, with a conservative rounding direction. Virtual seeds reduce first-depositor and direct-donation manipulation. A direct token donation does not mint shares and is excluded from accounted idle assets.

6.3 Deposit and withdrawal NAV

Unresolved market allocations have outcome-dependent equity. The pool therefore maintains conservative accounting views rather than pretending every active market has one exact, risk-free redemption value.

For each allocation, the market cell reports scenario values derived from:

- Accounted market assets.
- YES liability.
- NO liability.
- Conditional LP fee accrual.
- Market state version.

The pooled contract uses bounded reads across at most eight active allocations. Deposit and withdrawal calculations use contract-defined upper or lower scenario treatment and explicit rounding to prevent stale or optimistic values from diluting existing LPs or allowing unsafe exits.

6.4 Allocation policy

The current mainnet pool enforces:

Control	Value
Deposit cap	100,000 USDC
Maximum active allocations	8
Maximum deployed capital	80 percent
Minimum idle reserve	20 percent
Maximum exposure to one market	80 percent
Maximum exposure to one risk group	80 percent
Withdrawal window	1 hour
Immediate withdrawal cap per window	10 percent

Candidates are scanned under a bounded queue. An eligible candidate receives exactly its target allocation. The allocation records principal, risk group, isolated cell, cell shares, and terminal status.

6.5 Withdrawal liquidity

Pool shares represent a claim on a portfolio that may include active, nonwithdrawable market capital. Immediate redemption is limited by:

- Idle USDC.
- The 20 percent minimum idle reserve while allocations are active.
- The current one-hour withdrawal window.
- The 10 percent window limit.
- The current conservative withdrawal NAV.

A share note can therefore be economically valuable while exceeding immediate exit capacity. The owner keeps the note and may redeem a smaller amount, retry after the window resets, or use the supported exit mechanisms. Rate limits are liquidity controls, not a loss of ownership.

6.6 Terminal harvesting

When a market resolves or voids, its isolated liquidity cell determines terminal assets. The pool harvests that cell, removes deployed principal and risk-group exposure, increases idle assets by returned capital, and marks the allocation terminal.

The difference between returned terminal assets and principal, together with vested LP fees, changes pool share value. Loss is isolated to the capital assigned to that market but still affects all pool-share owners economically through aggregate NAV.

7. Shielded USDC note system

7.1 Public boundary and private interior

Circle USDC enters and leaves Moros through public Stellar Asset Contract transfers.

On deposit, the public ledger reveals:

- The authorizing Stellar account.
- The shared vault.
- The USDC amount.
- The transaction time.

Inside Moros, value is represented by notes. The contract stores commitments, encrypted output envelopes, commitment roots, and spent nullifiers. It does not store a public per-wallet private balance.

On withdrawal, the public ledger reveals the recipient, amount, vault, and time. Moros does not claim to hide either public boundary.

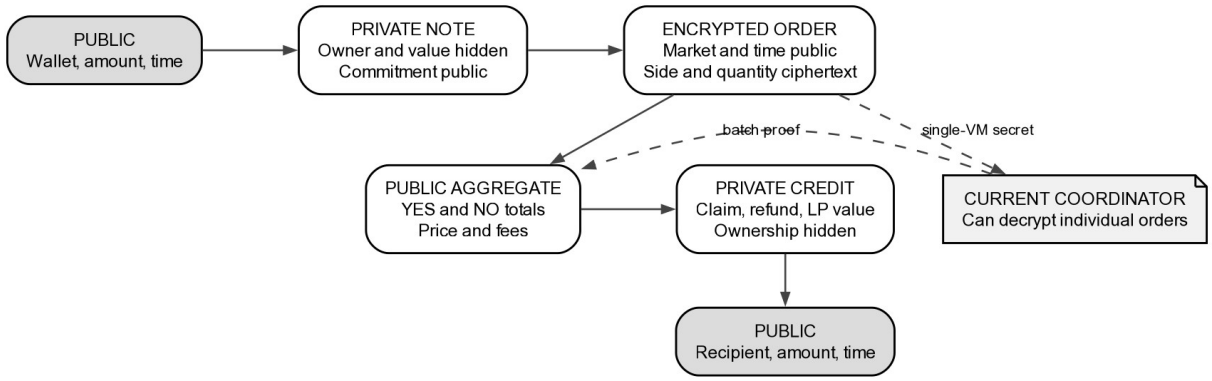

Figure 3. Public USDC boundaries and the shielded Moros interior.

7.2 Notes

A conceptual private note contains:

- A note type or purpose.
- An owner spending or recovery key.
- An amount or share quantity.
- A unique identifier.
- A blinding value.
- A network and contract domain.
- Optional market, epoch, or action context.

The plaintext note remains in the user's recoverable private state. The public commitment is a domain-separated Poseidon2 hash of the required fields [5].

7.3 Commitment tree

The shared vault appends output commitments to a 20-level Merkle tree, following the authenticated-tree membership pattern used by shielded note protocols [9][12]. Its capacity is:

$$2^{20} = 1{,}048{,}576 \text{ commitments}$$

The mainnet vault retains 64 recent roots and accepts proofs no older than 2,048 ledgers under the deployed policy. A spend proof demonstrates membership in an accepted root without revealing which public commitment belongs to the user.

7.4 Nullifiers

A nullifier is a deterministic, domain-separated value derived from a spendable note secret. Publishing a nullifier proves that the hidden note has been consumed without revealing the note commitment. The vault rejects duplicate nullifiers. This commitment and nullifier separation follows the established shielded-note model, with Moros-specific domains and transition rules [9].

This gives the note model its double-spend rule:

- Commitment proves that an eligible note existed.
- Nullifier proves that the same hidden note cannot be spent twice.
- The zero-knowledge circuit proves ownership and value conservation.

7.5 Fixed transition shapes

Most user transitions accept up to two input-nullifier slots and two output commitments. A one-input action places a real value in one slot and uses padding in the other. Fixed shapes reduce obvious metadata differences between action instances.

The common 15 public signals are:

Index class	Meaning
Action	Typed circuit and operation domain
Context digest	Hash binding network, vault, token, action, and operation data
Membership root	Accepted pre-state commitment root
Append root	Root immediately before outputs are appended
New root	Root after the two outputs

Index class	Meaning
Nullifiers	Count and two fixed slots
Output commitments	Two fixed slots
Envelope hashes	Hashes of two encrypted output envelopes
First leaf index	Append position
Public amount	Explicit sign and magnitude for public deposit or withdrawal boundary

The exit-match circuit uses 20 public inputs, and the batch circuit uses 49 because they bind additional parties and aggregate market state.

7.6 Reusable balance

A deposit creates a positive private balance note and a padding output. Bet and LP actions consume a sufficient private balance note and produce:

- An action-specific position or share note.
- A private change note for unused value.

If available value is fragmented, a private transfer can consolidate notes. Claims, refunds, LP exits, and LP redemptions produce reusable private USDC notes. A wallet is required for public deposits, public withdrawals, and archive unlock, not for each relayed internal action.

8. Encrypted private orders

8.1 Order contents

A private order selects:

- One market.
- One current epoch.
- Side `s`, where `s` is 0 or 1.
- Whole quantity `q`, where $1 \leq q \leq 1{,}000$.
- A maximum private budget.
- A terminal note owner.

The browser creates ElGamal ciphertexts over Baby Jubjub for YES and NO quantities [6][7]. Exactly one side carries the positive quantity:

$$\text{yesAmount} = s\ q$$

$$\text{noAmount} = (1 - s)\ q$$

The order proof establishes that `s` is binary, quantity is in range, ciphertexts use the registered committee public key, the position budget is sufficient, and the output notes conserve value.

8.2 Public order record

The accepted on-chain record includes:

- Market and epoch.
- Sequence number.
- Position commitment.
- Typed action identifier.
- Encrypted order points.
- Proof-bound output commitments and envelopes.
- Accepted-tree state.

It does not include a public trader wallet, plaintext side, or plaintext quantity. The target market and order timing are public.

8.3 Epoch ordering

Accepted orders receive contiguous sequence numbers in the current epoch. The epoch stores its first and last sequence, accepted count, accepted root, market state version, cutoff, and refund time. Batch construction must use the complete accepted sequence range. Reordering or omitting an accepted record changes the statement and fails verification.

8.4 Current coordinator model

The mainnet manifest records `committeeMode: single_vm`. The production private service derives or loads the combined committee secret, decrypts every accepted order, builds aggregate totals, and generates the batch proof.

Consequences:

- A public ledger observer does not see plaintext individual sides or quantities.
- The current Moros coordinator can see individual sides and quantities.
- Compromise of the coordinator can breach order confidentiality and interrupt batch availability.
- The mainnet deployment does not provide independently operated threshold privacy.

The repository contains earlier committee prototypes and future threshold designs, but they are not the active shared-vault runtime and are not represented as deployed security.

9. Aggregate batch proof

9.1 Purpose

Encryption alone would let a coordinator submit false totals. The batch proof connects the stored encrypted orders to the aggregate LMSR transition.

The service constructs a witness for up to eight real orders and pads unused slots. The 49 public inputs bind:

- Network domain, vault, market, and epoch.
- Accepted root, accepted count, and sequence range.
- Committee epoch, public key, and configuration hash.
- Aggregate ciphertext.
- Decryption and committee-statement hashes.
- Allocation and inclusion roots.
- Lot size and market state version.
- Aggregate YES and NO quantities.
- Pre-state and post-state prices.
- Per-side execution prices and charges.
- Aggregate market charge and rounding contribution.
- Fee escrow, LP fee, and protocol fee.

9.2 Proved statement

At a high level, a valid batch proof demonstrates:

1. Every active slot corresponds to one stored accepted ciphertext.
2. Every padded slot carries the required neutral values.
3. Each accepted ciphertext decrypts under the committed committee secret.
4. Each plaintext order has a valid side and bounded quantity.
5. YES and NO aggregate totals equal the sum of the decrypted orders.
6. Allocation packages map the authoritative side charge, fee, and change to each accepted position.
7. Public roots and hashes correspond to the complete ordered epoch.
8. The quoted market transition matches the bound aggregate values.

The contract verifies the Groth16 proof through the deployed verifier, checks the market quote again, transfers the exact aggregate charge, updates the epoch, and appends or authorizes the resulting private outputs.

9.3 State versioning

Each LMSR state change increments a market state version. An epoch records the version it expects. The batch quote, proof, and apply call all bind that version. If a conflicting transition changes the market first, the stale batch fails instead of applying against a different curve state.

9.4 Refund deadline

Once sealed, an epoch has a bounded execution opportunity. If a valid batch is not submitted before its refund time, a transaction can mark it refundable. Users then prove order ownership and recover the locked budget. The 600-second grace is an availability window, not the collection interval.

10. Claims, refunds, and value conservation

10.1 Execution change

An order locks a worst-case private budget. After a batch executes, the actual charge and fee are known. The user proves membership in the batch allocation and creates a private change note:

$$change = budget - executionCharge - executionFee$$

The proof binds the order ciphertext, allocation package, accepted position, and the user's hidden position data.

10.2 Winning claim

After a normal resolution, a winning position can create a private USDC credit equal to the base payout times its quantity, subject to exact atomic-unit rules:

$$winningCredit = lotPayout\ q$$

The shared vault has already redeemed aggregate winning shares from the LMSR market. A claim proof establishes position ownership, hidden winning side, quantity, eligible outcome, and terminal nullifier.

10.3 Losing position

A losing position does not receive a winning credit. Its budget change from execution remains reusable. The terminal state prevents later use of a conflicting claim or refund path.

10.4 Unexecuted refund

If the accepted epoch becomes refundable without execution, the order can recover the full locked budget into a new private balance note. It does not pay an execution fee because no market trade occurred.

10.5 Void refund

On void, aggregate market collateral and fee treatment follow the void accounting path. A position proves its own encrypted data and terminal eligibility to recover the value defined by the original budget and execution allocation. Conditional LP and protocol fees do not vest as ordinary execution revenue.

10.6 Pull-based settlement

All user settlement is pull-based. Moros may surface an action-required item in Portfolio and services may relay a proof-bound transaction, but funds do not move merely because a database row says a market is resolved. This avoids requiring a contract to enumerate private owners and preserves the note model.

11. Resolution and oracle routing

11.1 Resolver registry

The resolver registry maps an asset symbol to:

- Resolver contract.
- Risk group.
- Registration requirement.
- Enabled state.
- Route revision.

A proposal pins the exact route and revision. Activation revalidates them. Later registry changes do not mutate an already active market's resolver.

Governance changes use a 300-second delay in the current deployment. The registry permits new resolver contracts or routes without replacing the shared vault or pooled liquidity vault. New markets can use a new route after it is enabled; existing markets keep their pinned resolver.

11.2 Reflector price feeds

The current resolver reads two public mainnet Reflector contracts:

- A CEX price-feed contract for supported crypto assets.
- A fiat and metals contract for supported currencies and XAU.

Reflector public feeds implement the SEP-40 interface, use a uniform decimal representation, and update at a configured cadence [22][23][24]. Consumer contracts must check timestamps because old observations can remain available.

Moros normalizes supported observations to its resolver precision, verifies quote direction and freshness, compares the settlement value with the market strike, and invokes the market's resolver-only resolution path.

11.3 Current route coverage

The mainnet deployment enables 40 routes:

Group	Assets
Crypto against USD	BTC, ETH, USDT, XRP, SOL, USDC, ADA, AVAX, DOT, MATIC, LINK, DAI, ATOM, XLM, UNI, EURC
Foreign exchange against USD	EUR, GBP, CAD, BRL, JPY, CNY, MXN, KRW, TRY, ARS, PEN, VES, CLP, CRC, CDF, COP, HKD, INR, NGN, PHP, RUB, ZAR, KES
Metals against USD	XAU

The two feed contracts belong to one oracle provider family. They are not independent-provider redundancy.

11.4 Pyth and other adapters

The resolver source contains a Pyth Pro path and the keeper can request a paid Pyth Lazer payload when configured. Production mainnet has no configured Pyth resolver ID or access token, so the path is disabled.

Pyth's publisher and aggregation model, UMA's optimistic-oracle model, and Augur's dispute systems are useful comparative designs for future resolution work [27][28][29]. None is an active Stellar mainnet dependency of Moros. Adapter code or conceptual compatibility is not the same as an enabled resolver.

11.5 Event markets

Sports, politics, weather, economics, entertainment, and custom event markets are disabled. The repository contains an event-resolver foundation, but the production application does not expose those categories as tradeable.

Enabling one event category requires more than adding a UI card. It requires an exact evidence schema, source adapter, observation rules, challenge period, dispute or arbitration path, postponement and correction rules, timeout, void behavior, monitoring, and operator availability. A broad category may need separate resolver templates for each league, release series, station, or official source.

12. Encrypted private sync

12.1 Recovery problem

Private notes and position labels cannot depend only on browser local storage. Clearing storage or using another domain would otherwise make funds hard to discover even when the chain state remains valid.

Moros uses a wallet-authenticated encrypted archive for portfolio recovery. The wallet produces a deterministic, network-scoped signature. The browser derives archive keys through HKDF and encrypts fixed-size pages with AES-256-GCM [10][11].

12.2 Stored data

Supabase stores:

- A 43-character opaque bucket identifier.
- A public verification key for archive capability requests.
- Schema and generation counters.
- Fixed-size ciphertext pages.
- Per-page opaque identifiers.
- Nonces and ciphertext hashes.
- Short-lived request nonces.

The plaintext portfolio, note value, side, quantity, market participation, claim, refund, and LP ownership are inside the encrypted pages.

12.3 Request authorization

Read, register, and write operations use signed, expiring requests. The API verifies:

- Canonical payload hash.
- Public verification key.

- Request nonce.
- Expiry.
- Schema and fixed-size constraints.
- Compare-and-swap generation on updates.

Replay is blocked by consumed request nonces. AES-GCM associated data binds an encrypted page to its archive context and page identifier [10].

12.4 Metadata limits

Client-side encryption does not hide:

- IP address.
- Request time.
- Opaque bucket reuse.
- Number of pages within the allowed maximum.
- Fixed size class.
- Hosting and network metadata.

Supabase is not the source of authority for value. Chain commitments, nullifiers, and contract state determine whether an action is valid. Archive rollback or deletion can harm convenience and recovery availability but cannot authorize a forged spend.

12.5 Public social data

Wallet-authenticated comments and optional images are intentionally public and stored separately. They are not mixed with the encrypted financial archive. Users should assume anything placed in a market comment is public.

13. Zero-knowledge system

13.1 Proof system

Moros expresses its proof statements in Circom, uses version-pinned circomlib primitives, and produces Groth16 proofs with snarkjs on BN254 [4][8][30][31][32]. Stellar Protocol 25 introduced BN254 host operations through CAP-0074 and Poseidon or Poseidon2 permutation primitives through CAP-0075 [14][15][16][17]. Stellar mainnet currently runs Protocol 26. The verifier uses the Protocol 25 host primitives rather than implementing pairing arithmetic entirely in contract WASM.

Groth16 provides short proofs and efficient verification, but each circuit requires a circuit-specific proving key and verification key [4]. The proving key is intentionally public. Soundness depends on correct circuits, correct verification keys, a secure setup transcript, and uncompromised witness generation.

13.2 Circuit inventory

The mainnet proving manifest records 15 circuits:

Code	Circuit	Prover	Public inputs	Constraints
0	deposit	Browser	15	96,966
1	transfer	Browser	15	132,291
2	withdraw	Browser	15	114,627
3	order	Browser	15	143,989
4	claim	Browser	15	129,456
5	refund	Browser	15	129,473
6	liquidity_fund	Browser	15	115,463
7	liquidity_exit	Browser	15	115,470
8	liquidity_redeem	Browser	15	115,470
9	execution_change	Browser	15	129,449
10	treasury	Service	15	99,039
11	exit_request	Browser	15	120,400
12	exit_cancel	Browser	15	117,527
13	exit_match	Browser	20	218,827
14	batch	Service	49	233,310

13.3 Typed verification keys

The verifier registers each circuit under a numeric code and expected public-input count. Keys become immutable after finalization. A proof submitted under the wrong code, wrong signal length, wrong verifier domain, or wrong serialized form fails.

13.4 Domain separation

Moros binds proofs to:

- Stellar network domain.
- Shared vault contract.
- Circle USDC SAC.
- Verifier domain.
- Circuit action.
- Operation identifier.
- Market and epoch where applicable.
- Recipient or public account where applicable.
- Amount and expiry where applicable.

This prevents a proof generated for one network, vault, token, market, or action from being replayed as another.

13.5 Public artifacts

Browser circuits require downloadable WASM and proving keys. Public availability is necessary for permissionless proving and independent verification. Each artifact is recorded by SHA-256 in the proving manifest, together with:

- R1CS hash.
- Circuit source hash.
- Public-signal schema hash.
- Proving-key hash.
- Verification-key hash.
- Toolchain versions.
- Source bundle hash.
- Powers-of-tau hash.

Artifact integrity does not make the setup ceremony multi-party. The accepted mainnet setup is currently a single-contributor setup.

14. Stellar settlement

14.1 Why Stellar

Moros uses Stellar for final settlement, Circle USDC asset support, wallet authorization, and Soroban smart-contract execution. Stellar consensus is based on federated Byzantine agreement through the Stellar Consensus Protocol [13]. Moros does not alter consensus and inherits network-level safety and liveness assumptions.

14.2 Circle USDC and the SAC

USDC is a classic Stellar asset issued by Circle. Circle's official address registry identifies the Stellar mainnet issuer used by Moros [25][26]. The Stellar Asset Contract exposes the same classic asset to Soroban contracts through the CAP-46-6 and SEP-41 token interfaces without a separate bridged token [18][19][20]. The Moros mainnet manifest binds the exact USDC SAC and 7-decimal precision.

XLM pays transaction fees and supports account reserve requirements. It is not prediction-market collateral.

Moros cannot override issuer controls. Freeze, authorization, clawback, or network-level restrictions on the underlying asset may affect deposits, transfers, or withdrawals.

14.3 Soroban authorization

Public deposits and withdrawals require the appropriate account authorization. Internal relayed actions are accepted only when their zero-knowledge proof and operation context authorize the exact transition. The relayer pays XLM for submission but cannot alter the proof-bound outputs.

Cross-contract calls connect:

- Shared vault to Circle USDC.
- Shared vault to LMSR market.
- LMSR market to isolated liquidity vault.
- Factory to resolver registry, pool, vault, and market templates.

- Resolver to Reflector and market.

Nested failures revert the invocation.

14.4 Resource model

Stellar meters CPU instructions, memory, ledger reads, ledger writes, events, transaction size, and storage rent. Moros release builds use size optimization, link-time optimization, one codegen unit, stripped symbols, panic abort, and overflow checks.

Hot paths have bounded loops:

- At most 8 private orders per batch.
- 20 commitment-tree levels.
- At most 8 active pooled allocations.
- Fixed verification-key input counts.
- Deployment-controlled resolver routes.

The most expensive operations are Groth16 verification, Merkle state updates, encrypted output persistence, and cross-contract settlement. Performance is assessed through Soroban simulation and network resource limits rather than EVM gas.

14.5 State archival and TTL

Soroban contract instances, code, and persistent entries have time-to-live behavior [21]. Moros contracts expose bounded TTL extension paths. The keeper periodically extends active contracts and registry state. Nullifier and note correctness must remain safe across archival and restoration.

TTL maintenance is an operational liveness requirement. It does not change ownership or bypass proof checks.

15. Security model

15.1 Assets at risk

The security model protects:

- Circle USDC held by the shared vault and liquidity system.
- Private balance and share notes.
- Position side and quantity from public plaintext disclosure.
- Correct aggregate pricing.
- Claim and refund uniqueness.
- Resolver integrity.
- Recovery archive confidentiality and integrity.
- Contract and proving-artifact identity.

15.2 Adversaries

Moros considers:

- A public ledger analyst.
- A malicious user or proof submitter.
- A malicious relayer.
- A compromised private coordinator.
- A stale, unavailable, or manipulated oracle.
- A compromised browser or frontend origin.
- A malicious or curious Supabase operator.
- A compromised governance or service signer.
- RPC failure or inconsistent provider response.
- State archival and service restart.

15.3 Enforced protections

Risk	Protection
Double spend	Persistent nullifier uniqueness
Forged balance	Groth16 ownership, membership, and conservation constraints
Cross-network replay	Network and verifier domain binding
Wrong market action	Typed operation context and market binding
Stale market batch	Market state version

Risk	Protection
Selective order omission	Complete accepted sequence range and root binding
False aggregate	Batch proof over stored ciphertexts and aggregate totals
Unbounded LMSR loss	Activation subsidy at or above `b ln 2`
Insolvent transition	Scenario-liability check before batch application
Unsafe LP concentration	Deployed, idle, market, group, and count caps
Stale oracle	Resolver freshness and timeout checks
Route mutation	Pinned route revision per market
Archive tampering	AES-GCM, ciphertext hash, signed requests, and generation control
Relayer modification	Proof-bound outputs, recipient, context, and expiry
Artifact substitution	Deployment and proving manifest hashes

15.4 Privacy guarantees

Against an ordinary public ledger observer, Moros does not publish individual order side, quantity, private balance, position ownership, LP note ownership, or claim history as plaintext.

Privacy is conditional on usage. A unique deposit immediately followed by a unique order can be correlated by time. A unique public withdrawal amount can correlate with earlier activity. A sparse batch reveals aggregate totals that may narrow possibilities. The relayer sees the target market and request timing.

15.5 Current trusted parties

The mainnet release trusts:

- The single-VM coordinator not to disclose individual decrypted orders.
- The coordinator and keeper infrastructure for timely automation.
- Governance and configured signers within their contract powers.
- Reflector as the sole active oracle provider family.
- The frontend delivered to the browser not to steal secrets before proving.
- Circle and Stellar for the underlying asset and network.

These are not removed by zero-knowledge proofs.

15.6 Trusted setup

Groth16 security assumes toxic setup material was not retained. The accepted current setup uses one contributor. A malicious setup participant retaining trapdoor material may threaten proof soundness. Artifact hashes prove which setup is deployed but do not prove that toxic waste was destroyed.

An independently coordinated multi-party ceremony would reduce this trust assumption because soundness survives if at least one participant destroys its contribution.

15.7 Review status

The contracts, circuits, services, and deployment wiring have completed the team's internal review and test suites. They have not received an independent external audit. This document does not use internal review as a substitute for independent security assurance.

16. Failure modes and recovery

16.1 Insufficient batch activity

- Empty epoch: no price change.
- Nonempty one-sided epoch: may execute after cutoff and move the price.
- Epoch not executed before refund deadline: becomes refundable.
- Terminal market with zero YES or zero NO inventory: voids instead of normal resolution.

16.2 Coordinator outage

Accepted orders remain on-chain. If the batch is not produced in time, users use the refundable path after the configured deadline. New orders and automatic progress may be delayed while the primary coordinator is unavailable.

16.3 Oracle outage

Resolution waits through the recovery interval. If no valid observation becomes available before timeout, the resolver's stale-market path can void the market. Users then use proof-bound refund paths.

16.4 RPC failure

The service profile supports a selected RPC, a configured fallback, and a public network endpoint. Provider failover improves availability but does not make the coordinator redundant. RPC responses are still checked against the selected network passphrase and deployment manifest.

16.5 Public registry failure

Supabase stores discovery metadata, not market authority. If public listing fails after contracts are deployed and funded, the application or service can retry registration without redeploying the market or charging the liquidity target again.

16.6 Private archive loss

The encrypted archive improves cross-browser recovery. A local export provides an additional backup. Chain output history and wallet-derived recovery material remain the financial source of truth, but recovery tooling and retained history are operationally important. Supabase deletion cannot make an invalid spend valid, but it can reduce convenience and delay note discovery.

16.7 LP withdrawal pressure

Immediate redemption may be smaller than a user's total share value because active markets hold allocated capital. The pool enforces idle-reserve and window limits. Terminal harvest increases idle capacity after markets settle. The protocol does not promise instant redemption of all pool shares.

16.8 Contract archival

Keeper TTL transactions maintain active state. Stellar restoration mechanisms and permissionless extension paths reduce dependence on one periodic process. Operators must monitor both instance and persistent state because archival can interrupt availability even when ownership remains logically valid.

17. Mainnet deployment

17.1 Shared contracts

The canonical deployment manifest records the addresses, initialization transactions, runtime configuration, and code hashes below [34]. Live Stellar Expert records for the six shared contracts are collected in [36].

Contract	Stellar mainnet contract ID
Groth16 verifier	CD4RRUKDBQ6IP6LQJYTBGF5OOVFHIJWUULJCV52LANOLQEGYG4C5JQWL
Price resolver	CAUDDAZVWDGWHDC6IHV66RAQ2CTLMXGXUEOT6VNU7OF4ACZPMXUEDLWA
Resolver registry	CCK6GQ7DQDWCBZEIGVGDZ3AF3STHHXNVFYDKMFU634YHA7QDIQY3YKEB
Shared shielded vault	CBXZUAUEFAXZFRL4J7DTLS3GLAEY5OMIBBAUI672KJJE7FGU5LQJGXXL
Pooled liquidity vault	CB45XJ65Y46J2KGLUI6ZGUQ6C5EN7KS2BGI636ISIKSSBHVDYICPWP3F
Market factory	CDJ44IRLMZFEA4XY3J2XJMFLD4XIB3OIMRTYWBQZVX5ESBXMHNTOO3O7

Per-market LMSR and isolated liquidity vault addresses are created dynamically by the factory.

17.2 Mainnet dependencies

Dependency	Mainnet identifier
Circle USDC issuer	GA5ZSEJYB37JRC5AVCIA5MOP4RHTM335X2KGX3IHOJAPP5RE34K4KZVN

Dependency	Mainnet identifier
Circle USDC SAC	`CCW67TSZV3SSS2HXMBQ5JFGCKJNXKZM7UQUWUZPUTHXSTZLEO7SJMI75`
Reflector CEX feed	`CAFJZQWSED6YAWZU3GWRTOCNPPCGBN32L7QV43XX5LZLFTK6JLN34DLN`
Reflector fiat and metals feed	`CBKGPWGKSKZF52CFHMTRR23TBWTPMRDIYZ4O2P5VS65BMHYH4DXMCJZC`

17.3 Provenance

The deployed implementation is tied to the recorded Moros source commit, while the proving release separately binds circuit sources, toolchain versions, and artifact hashes [33][35].

Field	Value
Deployment account	`GAEUBHV5QCM5GMTMR2EED2CL2KIHSZ37YAHIQATZXNZREMVCP6TVREUH`
Source commit	`b8108b94376f665000c361049021c9be5b0cf138`
Network domain	`7ac33997544e3175d266bd022439b22cdb16508c01163f26e5cb2a3e1045a979`
Verifier domain	`6d1cba0152f257e03168e2dc1dfe9e818b9662220cbf8e2cfb6973606ce0e041`
Proving-manifest SHA-256	`f3e4d2120f23bee892d915f28400ab26cbae58bca62561a8b604b4aae838a623`
Setup label	`moros-mainnet-accepted-current-setup`
Curve and proof system	BN254 Groth16

17.4 Shared contract WASM hashes

Artifact	SHA-256
Verifier	`e2f43bb189dac32c44870f96f0f4534803a5e6efb713cde78a884e1fb1f8c739`
Price resolver	`fa2feaedc7622d45729e39e30a37946789934340a83bdd778981e7442194c06c`
Resolver registry	`8802df0b43b7ee2832de9a67da87eab68911f0a056f9608eaa5a81776bfaa686`
Shared shielded vault	`397a7769a0a70780c3f66b49f58ba0f7e270ebeccf57f9fa3d04ff18fb483e19`
Pooled liquidity vault	`d2065836ca2628d5ba4a5578b964f39380a20e77315f7d72a487da086ae6411d`
Market factory	`df0a4a4c029f55648980e3f921f4482799d7df570b997a491b47b16cd6c316e5`
LMSR market template	`567e85fd61a4be34bd1f5a1a5be43958c1649c662db0a6493de54ec389151346`
Isolated liquidity template	`742695db871af29ec35c2d85e76dfcfbd40180a705cfb815feb9c29975dc8b03`

18. Limitations and hardening priorities

18.1 Coordinator decentralization

The largest privacy gap is the combined committee secret on one VM. A stronger design would distribute key shares across independent operators and prove only aggregate decryption. It must also preserve order completeness, handle member outage, rotate epochs safely, and keep old orders executable or refundable.

18.2 Independent trusted setup

The deployed setup should be replaced only through a carefully managed contract and artifact transition. A public multi-party ceremony should publish contributions, transcript hashes, toolchain versions, final zkeys, verification keys, and deployment linkage.

18.3 Independent security assessment

Review should cover:

- Soroban contracts and cross-contract authorization.
- LMSR fixed-point approximations and rounding.
- Pool NAV, allocation caps, terminal harvesting, and withdrawal limits.
- All 15 circuits and public-signal encodings.
- Browser note selection and witness generation.
- Coordinator encryption and batch proof construction.
- Resolver normalization, freshness, timeout, and route governance.
- Deployment and artifact reproduction.
- Encrypted archive cryptography and request authorization.
- Operational key management and incident response.

18.4 Oracle diversity

The resolver registry makes new adapters possible without replacing the shared vault. Production diversity still requires a second independent provider, an aggregation or quorum rule, clear disagreement behavior, and monitoring. Two contracts from the same provider family do not satisfy this goal.

18.5 Event resolution

Event categories should be enabled one template at a time. Each template needs immutable interpretation rules and tested behavior for:

- Delayed, postponed, or abandoned events.
- Corrected official results.
- Conflicting sources.
- Missing or rate-limited evidence.
- Challenge and arbitration.
- Operator or source outage.
- Final void and refund.

18.6 Privacy at network and metadata layers

Future work may reduce timing and traffic correlation through relayer diversity, request batching, cover traffic, standardized denominations, withdrawal delay choices, independently hosted indexers, and stronger private information retrieval. These measures cannot protect a compromised browser that sees secrets before encryption.

19. Conclusion

Moros combines an automated prediction-market maker with a shielded USDC note system and encrypted batch execution. Its central technical separation is between public market truth and private participant intent. Stellar records the market definition, aggregate liquidity, proof-verified state transitions, batch totals, probability changes, oracle result, and terminal accounting. It does not receive a plaintext mapping from public wallets to individual sides, quantities, LP shares, claims, or refunds.

The system is designed as a complete lifecycle. Permissionless proposals activate only after resolver and liquidity checks. LMSR loss is bounded and funded. Nonempty one-sided batches can execute. Stale batches become refundable. Unsupported oracle categories remain disabled. Winners claim through proofs, and LP capital returns through isolated terminal settlement.

The current deployment is also deliberately bounded. Public deposits and withdrawals remain visible. Reflector is one provider family. The coordinator can decrypt individual orders. The setup has one contributor, and review is internal. These limitations define the present trust model and the work required to strengthen it.

Moros is live on Stellar mainnet at `https://moros.fun`.

Appendix A. Protocol parameters

Parameter	Mainnet value
Collateral precision	7 decimals
Note tree depth	20
Note tree capacity	1,048,576 commitments
Root history	64
Maximum root age	2,048 ledgers
Maximum proof bytes	512
Accepted-order tree depth	6
Maximum orders per epoch	8
Quantity per order	1 to 1,000
Epoch collection duration	60 seconds
Batch grace	600 seconds
Refund delay	600 seconds
Maximum probability movement	0.25
Liquidity tiers	20, 50, 100 USDC
Maximum market fee	10 percent
Default application fee	2 percent
LP share of distributable fees	80 percent
Protocol share of distributable fees	20 percent
Minimum funding window	240 seconds
Minimum open window	600 seconds
Maximum market duration	90 days
Resolver route change delay	300 seconds
Pool deposit cap	100,000 USDC
Maximum active allocations	8
Maximum deployed pool assets	80 percent
Minimum idle pool assets	20 percent
Maximum single-market exposure	80 percent
Maximum single-group exposure	80 percent
Withdrawal window	1 hour
Immediate withdrawal cap	10 percent per window

Appendix B. Visibility matrix

Data	Public ledger	Private service	Supabase	User browser
Deposit wallet and amount	Visible	Observable through chain or request	Not required in private page plaintext	Visible
Private note owner and value	Commitment only	Encrypted output; recovery service metadata	Ciphertext only	Plaintext while unlocked
Target market and order time	Visible	Visible	Inside encrypted portfolio page	Visible
Individual side and quantity	Ciphertext only	Visible to current coordinator	Ciphertext only	Visible
Aggregate batch totals	Visible after execution	Visible	May be public market metadata	Visible
LP pool assets and allocations	Visible	Visible	Public market metadata	Visible
Individual LP note ownership	Commitment only	Encrypted note flow	Ciphertext only	Visible
Claim or refund owner and value	Commitment transition; public amount absent until withdrawal	Proof request metadata	Ciphertext only	Visible
Public withdrawal recipient and amount	Visible	Observable through relay or chain	Not required in private page plaintext	Visible
Comment and image	Not on-chain	Visible	Plaintext public social data	Visible
Archive access metadata	Not applicable	API can observe request metadata	Observable	Visible locally

Appendix C. Contract state summary

C.1 Market proposal states

1. Proposed.
2. Waiting for pool capacity.
3. Isolated liquidity funded.
4. Ready for activation.
5. Active.
6. Cancelled or expired before activation.

C.2 Epoch states

1. Collecting.
2. Sealed.
3. Executed.
4. Refundable.

After an executed or refundable epoch and before market expiry, the vault may open the next epoch.

C.3 Market terminal states

1. YES resolved.
2. NO resolved.
3. Void.

C.4 Pool allocation states

1. Candidate pending.
2. Candidate skipped or expired.
3. Capital deployed.
4. Terminal capital harvested.

Appendix D. Reference implementation inventory

Layer	Repository location
ZK verifier	`contracts/zk-verifier`
Shared shielded vault	`contracts/shielded-collateral-vault`
Privacy types	`contracts/privacy-types`
LMSR market	`contracts/lmsr-market`
Pooled liquidity	`contracts/pooled-liquidity-vault`
Isolated liquidity	`contracts/market-liquidity-vault`
Market factory	`contracts/market-factory`
Price resolver	`contracts/resolver`
Resolver registry	`contracts/resolver-registry`
Circuits	`contracts/shielded-collateral-vault/circuits`
Mainnet manifest	`deployments/private-mainnet.json`
Proving manifest	`deployments/private-mainnet-proving.json`
Private service	`services/private-server.mjs`
Batch coordinator	`services/private-batch-coordinator.mjs`
Resolution keeper	`services/resolve-keeper.mjs`
Browser private actions	`web/lib/private/actions.ts`
Encrypted archive	`web/lib/private-sync`
Archive schema	`web/supabase/migrations/20260723000000_opaque_private_activity_sync.sql`

References

Academic foundations and market design

[1] Justin Wolfers and Eric Zitzewitz. "Prediction Markets." Journal of Economic Perspectives, 18(2), 2004, pp. 107-126. DOI

[2] Robin Hanson. "Logarithmic Market Scoring Rules for Modular Combinatorial Information Aggregation." Journal of Prediction Markets, 1(1), 2007, pp. 3-15. Paper

[3] Constantin Burgi, Wanying Deng, and Karl Whelan. "Makers and Takers: The Economics of the Kalshi Prediction Market." CEPR Discussion Paper 20631, revised January 2026. Paper

Privacy and cryptography

[4] Jens Groth. "On the Size of Pairing-based Non-interactive Arguments." Advances in Cryptology, EUROCRYPT 2016, pp. 305-326. Paper

[5] Lorenzo Grassi, Dmitry Khovratovich, Christian Rechberger, Arnab Roy, and Markus Schofnegger. "Poseidon2: A Faster Version of the Poseidon Hash Function." IACR Transactions on Cryptographic Hardware and Embedded Systems, 2023(3), pp. 472-523. Paper

[6] Taher ElGamal. "A Public Key Cryptosystem and a Signature Scheme Based on Discrete Logarithms." IEEE Transactions on Information Theory, 31(4), 1985, pp. 469-472. DOI

[7] Barry WhiteHat, Marta Bellés-Muñoz, and Jordi Baylina. "EIP-2494: Baby Jubjub Elliptic Curve." Ethereum Improvement Proposals, 2020. Standard

[8] Marta Bellés-Muñoz, Miguel Isabel, Jose Luis Muñoz-Tapia, Alberto Rubio, and Jordi Baylina. "Circom: A Circuit Description Language for Building Zero-Knowledge Applications." IEEE Transactions on Dependable and Secure Computing, 20(6), 2023, pp. 4733-4751. DOI

[9] Daira-Emma Hopwood, Sean Bowe, Taylor Hornby, and Nathan Wilcox. "Zcash Protocol Specification." July 2026. Specification

[10] Morris Dworkin. "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode and GMAC." NIST Special Publication 800-38D, 2007. Standard

[11] Hugo Krawczyk and Pasi Eronen. "HMAC-based Extract-and-Expand Key Derivation Function." RFC 5869, 2010. Standard

[12] Ralph C. Merkle. "A Digital Signature Based on a Conventional Encryption Function." Advances in Cryptology, CRYPTO 1987, pp. 369-378. DOI

Stellar protocol and standards

[13] David Mazieres. "The Stellar Consensus Protocol: A Federated Model for Internet-level Consensus." 2016. Paper

[14] Stellar Development Foundation. "Stellar Software Versions." Mainnet Protocol 26, accessed 26 July 2026. Documentation

[15] Stellar Development Foundation. "CAP-0074: Host Functions for BN254." Final, Protocol 25. CAP-0074

[16] Stellar Development Foundation. "CAP-0075: Host Functions for Poseidon and Poseidon2 Permutations." Final, Protocol 25. CAP-0075

[17] Stellar Development Foundation. "ZK Proofs on Stellar." Accessed 26 July 2026. Documentation

[18] Stellar Development Foundation. "CAP-0046-06: Smart Contract Standardized Asset." CAP-0046-06

[19] Stellar Development Foundation. "SEP-0041: Soroban Token Interface." Draft version 0.4.1. SEP-0041

[20] Stellar Development Foundation. "Stellar Asset Contract." Accessed 26 July 2026. Documentation

[21] Stellar Development Foundation. "Smart Contract State Archival." Accessed 26 July 2026. Documentation

[22] Stellar Development Foundation. "SEP-0040: Oracle Consumer Interface." Draft version 0.1.0. SEP-0040

Oracle and asset dependencies

[23] Reflector Network. "Reflector Oracle Documentation." Accessed 26 July 2026. Documentation

[24] Reflector Network. "Price Feed Interface." Accessed 26 July 2026. Interface

[25] Circle. "USDC Contract Addresses." Stellar mainnet issuer registry, accessed 26 July 2026. Address registry

[26] Circle. "Transfer USDC on Stellar." Accessed 26 July 2026. Documentation

[27] Pyth Data Association. "Pyth Network Whitepaper." 2023. Whitepaper

[28] UMA. "How Does UMA's Oracle Work?" Accessed 26 July 2026. Documentation

[29] Ryan Garner and Philip Monastirsky. "A Bribery-Resistant Group-Strategyproof Oracle." Augur Lituus Whitepaper, January 2026. Whitepaper

Tooling, implementation, and deployment records

[30] iden3. "Circom: zkSNARK Circuit Compiler." Source

[31] iden3. "snarkjs: zkSNARK Implementation in JavaScript and WebAssembly." Source

[32] iden3. "circomlib: Library of Basic Circuits for Circom." Source

[33] Moros. "Protocol Source at the Mainnet Deployment Commit." Commit `b8108b94376f665000c361049021c9be5b0cf138`. Source

[34] Moros. "Mainnet Deployment Manifest." Manifest

[35] Moros. "Mainnet Proving Artifact Manifest." Manifest

[36] Stellar Expert. "Moros Shared Mainnet Contract Records." Verifier, resolver, registry, shared vault, liquidity pool, and factory.