SolCircle

Trustless Group Trading Infrastructure on Solana

Technical White Paper

SolCircle Development Team
solcircleindia@gmail.com

Version 1.0
October 2025

Abstract

Telegram has become the primary coordination platform for cryptocurrency trading groups, yet these communities operate on fundamentally broken trust models where administrators control member funds directly. SolCircle introduces a trustless infrastructure that transforms Telegram groups into secure, on-chain trading collectives by leveraging Solana's Program Derived Addresses, decentralized oracles, and democratic governance mechanisms. This paper presents the technical architecture of SolCircle, detailing how blockchain primitives can solve coordination problems in decentralized group trading while maintaining the social dynamics that make these communities effective.

Contents

1 Introduction

Cryptocurrency trading has evolved into a highly social activity, with Telegram emerging as the dominant coordination platform for traders worldwide. Thousands of trading groups exist, ranging from small communities of friends pooling capital to large-scale signal providers coordinating trades for tens of thousands of members. These groups share trading signals, coordinate market movements, pool capital for larger positions, and collectively analyze market conditions. However, this social revolution in trading operates on a fundamentally broken trust model where every group assumes administrators and fund managers will act honestly, a premise that has proven catastrophically false in countless cases.

The current state of group trading presents several critical problems. Group administrators control member funds directly, creating single points of failure and custodial risk. Members must trust that administrators will not abscond with funds, make unauthorized trades, or misreport profits and losses. Trading decisions, fund movements, and profit distributions often occur off-chain with minimal transparency, making it impossible for members to verify that trades were executed as promised or that profits are distributed fairly. When losses occur, it becomes impossible to determine whether they resulted from poor trading decisions, market conditions, or malicious actions by administrators. This ambiguity erodes trust and prevents effective group coordination. The ease with which administrators can disappear with member funds has made exit scams commonplace, and even well-intentioned groups face reputational damage from association with fraudulent operators.

Existing solutions fall short of addressing these fundamental issues. Multi-signature wallets provide some protection against unilateral actions but still require trusting a small group of key holders and do not provide transparency into trading decisions or profit distribution logic. Copy trading platforms and social trading apps solve the trust problem by introducing a centralized intermediary, but this merely shifts trust from group administrators to platform operators while adding fees and reducing member autonomy. Traditional DAO frameworks are too heavyweight for trading groups, requiring complex governance tokens, voting periods unsuitable for rapid trading decisions, and technical expertise beyond most community members. Basic pooling contracts exist but lack integration with social coordination platforms, making them impractical for communities that organize primarily through Telegram.

SolCircle bridges the gap between social coordination on Telegram and trustless execution on Solana. Our architecture enables non-custodial fund management where members retain control of their funds through Program Derived Addresses, eliminating custodial risk entirely. All trades execute on-chain through verifiable smart contracts, creating an immutable audit trail. Integrated voting mechanisms allow groups to make collective decisions about trading strategies and fund management. Profits and losses are calculated and distributed algorithmically based on contribution ratios, eliminating disputes and manual processes. Decentralized oracles validate trading outcomes before funds are released, preventing manipulation. Integration with Telegram through an intuitive bot interface makes the system accessible to non-technical users while maintaining the security guarantees of blockchain execution.

2 System Architecture

SolCircle's architecture consists of six integrated layers that work together to provide seamless, trustless trading. The Telegram interface layer provides the user-facing bot that handles commands, displays information, and facilitates voting. The off-chain coordination layer comprises backend services that manage group metadata, voting logic, and oracle operations. The on-chain program layer consists of Solana smart contracts that control fund custody, trade execution, and profit distribution. The Arcium MPC layer provides privacy-preserving dark pool operations through multi-party computation. The oracle network layer provides decentralized verification

of trading outcomes and authorization of fund releases. The DeFi integration layer connects to DEXs, lending protocols, and other DeFi primitives to enable actual trading operations.

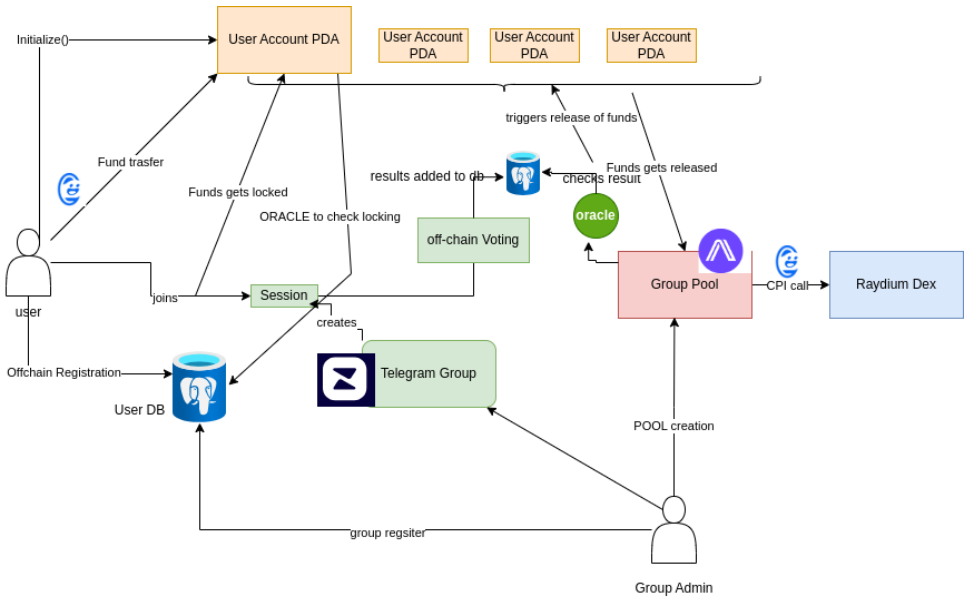

Figure 1: SolCircle System Architecture

2.1 Program Derived Addresses and User Accounts

Every user in the SolCircle ecosystem has a Program Derived Address that serves as their personal vault within the system. This PDA is deterministically derived from the user's Telegram ID and the SolCircle program address, ensuring uniqueness and recoverability across devices.

2.1.1 PDA Derivation

The user's PDA is derived using Solana's canonical PDA derivation:

$$\text{PDA}_{\text{user}} = \text{findProgramAddress}([\text{“user\_deposit”}, \text{userPubkey}], \text{programID}) \tag{1}$$

where the seeds consist of a static string identifier and the user's public key. The vault for holding SPL tokens uses a similar derivation:

$$\text{PDA}_{\text{vault}} = \text{findProgramAddress}([\text{“user\_vault”}, \text{userPubkey}], \text{programID}) \tag{2}$$

The non-custodial design means users maintain full control through their connected wallet signatures, with funds in one group pool completely isolated from positions in other pools. The account structure stores:

$$\text{UserDeposit} = \{\text{owner} : \text{Pubkey}, \tag{3}$$

$$\text{deposited\_amount} : u64, \tag{4}$$

$$\text{bump} : u8\} \tag{5}$$

This structure supports future feature additions through versioning, allowing the protocol to evolve without requiring users to migrate to new accounts.

2.1.2 Fund Transfer Mathematics

When a user deposits amount D_i to their PDA, the on-chain state transition is:

$$B_{\text{user}}^{\text{new}} = B_{\text{user}}^{\text{old}} + D_i \tag{6}$$

where B_{user} represents the user's deposited balance tracked in the UserDeposit account.

When joining a pool, funds transfer from the user vault to the group vault via a Cross-Program Invocation:

$$\begin{aligned} B_{\text{user}}^{\text{new}} &= B_{\text{user}}^{\text{old}} - A_i \\ B_{\text{pool}}^{\text{new}} &= B_{\text{pool}}^{\text{old}} + A_i \end{aligned} \tag{7}$$

where A_i is the amount allocated to the group pool by user i. This operation is atomic and requires oracle signature verification:

$$\text{Verify}(\sigma_{\text{oracle}}, H(m)) \to \{\text{true}, \text{false}\} \tag{8}$$

where $m = \text{concat}(\text{“release\_to\_group”}, \text{userID}, \text{groupID}, A_i, \lfloor t/300 \rfloor)$ and t is the current Unix timestamp. The time quantization to 300-second windows prevents replay attacks.

2.2 Group Pool Architecture

The Group Pool is the central smart contract that holds and manages collective funds for a trading group. The pool maintains a sophisticated balance between liquid assets available for immediate trading and positions deployed in DeFi protocols.

2.2.1 Pool State and Invariants

The pool maintains the following on-chain state:

$$\begin{aligned} \text{GroupPool} = \{&\text{admin} : \text{Pubkey}, && (9)\\ &\text{group\_id} : \text{String}, && (10)\\ &\text{total\_locked} : u64, && (11)\\ &\text{is\_active} : \text{bool}, && (12)\\ &\text{bump} : u8\} && (13) \end{aligned}$$

The fundamental pool invariant is:

$$\sum_{i=1}^{n} A_i = B_{\text{pool}} \tag{14}$$

where A_i is the contribution of user i and B_{pool} is the total pool balance. This invariant is enforced at every state transition.

2.2.2 Share Calculation

Each user's ownership share in the pool is computed as:

$$\phi_i = \frac{A_i}{\sum_{j=1}^{n} A_j} = \frac{A_i}{B_{\text{pool}}} \tag{15}$$

where $\phi_i \in [0, 1]$ and $\sum_{i=1}^{n} \phi_i = 1$. After a trade that changes pool value to B'_{pool}, user i's new balance is:

$$A'_i = \phi_i \cdot B'_{\text{pool}} \tag{16}$$

This proportional distribution ensures fairness: profits and losses are shared according to contribution ratios.

2.2.3 Profit and Loss Distribution

When a trade executes with profit/loss ΔP:

$$\Delta P = B_{\text{pool}}^{\text{after}} - B_{\text{pool}}^{\text{before}} \tag{17}$$

Each user's P&L is:

$$\Delta P_i = \phi_i \cdot \Delta P = \frac{A_i}{B_{\text{pool}}^{\text{before}}} \cdot \Delta P \tag{18}$$

The pool contract includes comprehensive protections such as slippage limits, maximum gas consumption constraints, and emergency halt mechanisms that can freeze operations if anomalous behavior is detected.

All open positions are tracked on-chain with detailed metadata including entry price, position size, target exit conditions, and stop-loss parameters. This enables automated position management and provides transparent performance reporting to all pool members. The pool automatically allocates portions of profits according to governance parameters set during pool creation, handling performance fees, protocol fees, and trader compensation without manual intervention. Position tracking occurs in real-time, with every trade execution updating the pool's state and triggering recalculation of member shares based on current valuations.

The pool implements sophisticated liquidity management to handle the competing demands of active trading and member withdrawals. A portion of funds always remains in liquid form to service withdrawal requests, while the remainder can be deployed in trading positions or yield-generating DeFi protocols. The system calculates optimal liquidity ratios based on historical withdrawal patterns, current market volatility, and pending trade proposals. When liquidity falls below safe thresholds, the pool automatically begins unwinding positions to restore adequate reserves, prioritizing positions with minimal slippage and those closest to target exit prices.

2.3 Oracle Network

SolCircle employs a decentralized oracle network to validate critical state transitions and prevent manipulation throughout the system.

2.3.1 Oracle Consensus Mechanism

Let $\mathcal{O} = \{o_1, o_2, \ldots, o_k\}$ be the set of k oracle nodes. For a data request r, each oracle o_i provides response d_i. The consensus value is determined by:

$$d_{\text{consensus}} = \text{median}(\{d_1, d_2, \ldots, d_k\}) \tag{19}$$

A response d_i is considered valid if:

$$|d_i - d_{\text{consensus}}| \leq \epsilon_{\text{tolerance}} \tag{20}$$

where $\epsilon_{\text{tolerance}}$ is the maximum allowed deviation. Consensus is reached when:

$$\frac{|\{i : |d_i - d_{\text{consensus}}| \leq \epsilon_{\text{tolerance}}\}|}{k} \geq \theta_{\text{oracle}} \tag{21}$$

where θ_{oracle} is the oracle consensus threshold (typically ≥ 0.66).

2.3.2 Stake-Based Security

Each oracle o_i must stake collateral S_i where:

$$S_i \geq S_{\min} \cdot \max(\text{value\_at\_risk}) \tag{22}$$

If oracle o_i provides a false attestation (detected when $|d_i - d_{\text{consensus}}| > \epsilon_{\text{tolerance}}$), a portion of their stake is slashed:

$$S_i' = S_i - \alpha \cdot S_i \tag{23}$$

where $\alpha \in [0.1, 1.0]$ is the slashing rate that increases with deviation magnitude.

2.3.3 Oracle Signature Verification

Fund release operations require a valid oracle signature. The verification function is:

$$\text{VerifyOracle}(\sigma, m, pk_{\text{oracle}}) = \begin{cases} \text{true} & \text{if Ed25519.verify}(\sigma, m, pk_{\text{oracle}}) = \text{true} \\ \text{false} & \text{otherwise} \end{cases} \tag{24}$$

where σ is the signature, m is the message, and pk_{oracle} is the oracle's public key stored in the OracleConfig account.

The oracle network consists of multiple independent nodes that reach consensus on various types of data before it can affect on-chain state. For price feeds, oracles provide real-time asset prices used for profit and loss calculations, particularly important for assets not traded directly on-chain or for positions held in external protocols. The oracles verify that executed trades achieved the reported outcomes, preventing administrators or malicious actors from falsifying trading results.

2.4 Arcium Confidential Dark Pools

SolCircle implements privacy-preserving dark pools using Arcium's Multi-Party Computation (MPC) network, enabling groups to trade with encrypted balances that remain hidden on-chain while maintaining full verifiability of operations.

2.4.1 Confidential Balance Representation

In contrast to public pools where balances are stored as plaintext $u64$ values, confidential pools store encrypted balance ciphertexts:

$$\widetilde{B}_i = \text{Enc}_{\text{MPC}}(B_i, r_i) \tag{25}$$

where B_i is user i's true balance, r_i is randomness, and Enc_{MPC} is Arcium's encryption scheme. The on-chain state stores:

$$\begin{aligned} \text{ConfidentialUserDeposit} = \{&\text{owner} : \text{Pubkey}, && (26)\\ &\text{encrypted\_balance} : [u8; 32], && (27)\\ &\text{last\_update\_slot} : u64, && (28)\\ &\text{is\_active} : \text{bool}, && (29)\\ &\text{bump} : u8\} && (30) \end{aligned}$$

2.4.2 MPC-Powered Operations

All arithmetic operations on encrypted balances occur off-chain in the Arcium MPC network, ensuring that no single party learns the plaintext values. The protocol supports four fundamental encrypted circuits:

Encrypted Deposit: Given encrypted balance $\widetilde{B}_i$ and plaintext deposit D, compute:

$$\widetilde{B}'_i = \text{MPC-Add}(\widetilde{B}_i, D) = \text{Enc}_{\text{MPC}}(B_i + D, r'_i) \tag{31}$$

The MPC network receives the encrypted input:

$$\text{input} = \{\text{current\_balance} : \widetilde{B}_i, \text{deposit\_amount} : D\} \tag{32}$$

and returns the encrypted result:

$$\text{result} = \{\text{new\_balance} : \widetilde{B}'_i, \text{success} : \text{bool}\} \tag{33}$$

Encrypted Withdrawal: For withdrawal amount W, the MPC performs:

$$\widetilde{B}'_i = \begin{cases} \text{Enc}_{\text{MPC}}(B_i - W, r'_i) & \text{if } B_i \geq W \\ \widetilde{B}_i & \text{otherwise} \end{cases} \tag{34}$$

The MPC verifies sufficient balance without revealing B_i:

$$\text{success} = \begin{cases} \text{true} & \text{if } \text{Dec}(\widetilde{B}_i) \geq W \\ \text{false} & \text{otherwise} \end{cases} \tag{35}$$

Encrypted Pool Transfer: When transferring amount T from user to confidential pool:

$$\widetilde{B}'_{\text{user}} = \text{Enc}_{\text{MPC}}(B_{\text{user}} - T, r'_1) \tag{36}$$

$$\widetilde{B}'_{\text{pool}} = \text{Enc}_{\text{MPC}}(B_{\text{pool}} + T, r'_2) \tag{37}$$

The MPC circuit implements:

$$\begin{aligned} \text{has\_sufficient} &= (\text{Dec}(\widetilde{B}_{\text{user}}) \geq T) \\ \text{result.transfer\_success} &= \text{has\_sufficient} \\ \text{result.new\_user\_balance} &= \begin{cases} \text{Enc}(B_{\text{user}} - T) & \text{if has\_sufficient} \\ \widetilde{B}_{\text{user}} & \text{otherwise} \end{cases} \end{aligned} \tag{38}$$

Encrypted Share Computation: For profit distribution, compute user i's share:

$$S_i = \left\lfloor \frac{C_i \cdot P}{T} \right\rfloor \tag{39}$$

where C_i is user i's contribution, $T = \sum_{j=1}^{n} C_j$ is total pool value, and P is profit to distribute. In the confidential setting:

$$\widetilde{S}_i = \text{MPC-Divide}(\text{MPC-Multiply}(\widetilde{C}_i, P), \widetilde{T}) \tag{40}$$

2.4.3 MPC Callback Protocol

The confidential operation flow follows a request-callback pattern:

1. User initiates operation, queueing MPC request on-chain
2. Arcium network receives encrypted inputs
3. MPC nodes collaboratively compute result without learning plaintexts
4. Consensus is reached on encrypted output
5. Callback instruction updates on-chain encrypted state

The callback must be signed by the Arcium MPC authority:

$$\text{require}(\text{signer} = \text{ARCIUM\_MPC\_AUTHORITY}) \tag{41}$$

This prevents unauthorized state updates and ensures only valid MPC results are accepted.

2.4.4 Privacy Guarantees

The security of confidential pools relies on the honest-majority assumption in the MPC network:

$$|\text{ByzantineNodes}| < \frac{|\text{TotalNodes}|}{2} \tag{42}$$

Under this assumption, the protocol provides:

- **Balance Privacy:** Individual balances remain encrypted on-chain, with decryption key known only to the owner
- **Transfer Privacy:** Transfer amounts are never revealed publicly
- **Correctness:** MPC computations are verifiably correct even if $< 50\%$ of nodes are malicious
- **Auditability:** Users can decrypt their own balances client-side to verify correctness

2.4.5 Dual-Mode Architecture

SolCircle's unique design allows users to choose between public and confidential pools:

Property	Public Pool	Confidential Pool
Balance Visibility	Fully visible	Encrypted
Gas Costs	Lower	Higher (MPC overhead)
Privacy	None	Strong
Latency	400ms	2-5s (MPC compute)
Audit Complexity	Trivial	Requires decryption

Table 1: Comparison of Public vs Confidential Pools

This dual-mode approach allows users to optimize for their specific requirements, trading off privacy for cost and latency when appropriate.

2.5 Voting and Governance

The SolCircle governance system balances the need for rapid trading decisions with democratic accountability through a flexible proposal and voting framework.

2.5.1 Weighted Voting Mechanism

Each user i with contribution A_i receives voting power:

$$V_i = \frac{A_i}{\sum_{j=1}^{n} A_j} \tag{43}$$

For a proposal p, let Y_p be the set of users who vote yes and N_p be the set who vote no. The proposal passes if:

$$\sum_{i \in Y_p} V_i > \theta_{\text{threshold}} \tag{44}$$

where $\theta_{\text{threshold}} \in [0.5, 1.0]$ is the approval threshold (typically 0.5 for simple majority or 0.66 for supermajority).

2.5.2 Quorum Requirements

To prevent low-participation votes, proposals also require minimum quorum:

$$\sum_{i \in Y_p \cup N_p} V_i \geq Q_{\min} \tag{45}$$

where $Q_{\min}$ is the minimum quorum parameter. A proposal only executes if:

$$\left(\sum_{i \in Y_p} V_i > \theta_{\text{threshold}} \right) \wedge \left(\sum_{i \in Y_p \cup N_p} V_i \geq Q_{\min} \right) \tag{46}$$

2.5.3 Time-Weighted Voting

For time-sensitive proposals, voting power may decay over time to incentivize rapid participation:

$$V_i(t) = V_i \cdot e^{-\lambda(t - t_{\text{proposal}})} \tag{47}$$

where λ is the decay parameter and t_{proposal} is the proposal creation time.

2.5.4 Vote Delegation

Users can delegate voting power to delegates. If user i delegates to delegate d, then:

$$V_d^{\text{effective}} = V_d + \sum_{i \in \text{Delegators}(d)} V_i \tag{48}$$

The delegation is revocable at any time, with revocation taking effect on the next vote.

Any pool member can create proposals through the Telegram bot interface, with proposal types including trade execution, parameter changes such as fee structures or voting thresholds, authorization changes for traders with special privileges, profit distribution events, and pool dissolution with return of funds to members. Each proposal includes detailed parameters, supporting rationale, and an expiration time after which it becomes invalid if not executed.

Different proposal types have different voting periods calibrated to their urgency and impact. Routine trade proposals may execute after short voting periods measured in minutes or hours, allowing groups to respond quickly to market opportunities. Parameter changes require longer deliberation periods measured in days to ensure all members have time to review and understand proposed changes. Major decisions such as pool dissolution or authorization changes for key roles require supermajority approval with extended voting periods and high quorum thresholds. Once a proposal passes all requirements, the system automatically executes it on-chain by

validating that pool liquidity is sufficient, constructing the transaction with appropriate slippage protections, executing through Cross-Program Invocation to the target protocol, recording the result on-chain and updating position tracking, and notifying pool members through Telegram of the outcome.

2.6 Sanctum Gateway Transaction Optimization

SolCircle leverages Sanctum Gateway to optimize transaction delivery to the Solana network, providing superior execution guarantees and lower failure rates compared to standard RPC submissions. Gateway abstracts the complexity of transaction routing and allows real-time optimization without code changes.

2.6.1 Transaction Builder Stage

The Gateway Builder stage constructs optimized transactions through the `buildGatewayTransaction` JSON-RPC method:

$$T_{\text{optimized}} = \text{buildGatewayTransaction}(T_{\text{raw}}, \theta) \tag{49}$$

where T_{raw} is the unsigned base transaction and θ represents optimization parameters:

$$\begin{aligned} \theta = \{&\text{cuPriceRange} : \{\text{low, medium, high}\}, && (50)\\ &\text{jitoTipRange} : \{\text{low, medium, high, max}\}, && (51)\\ &\text{deliveryMethodType} : \mathcal{D}, && (52)\\ &\text{expireInSlots} : n\} && (53) \end{aligned}$$

where $\mathcal{D} \in \{\text{rpc, jito, sanctum-sender, helius-sender}\}$.
The builder performs the following optimizations automatically:

Compute Unit Optimization: Gateway simulates the transaction to determine actual compute units consumed:

$$\text{CU}_{\text{limit}} = \text{CU}_{\text{simulated}} \cdot (1 + \epsilon_{\text{safety}}) \tag{54}$$

where $\epsilon_{\text{safety}} \approx 0.05$ provides a safety margin for execution variance.

Priority Fee Calculation: Gateway fetches real-time prioritization fees and sets:

$$\text{CU}_{\text{price}} = \text{percentile}(\{\text{fees}_{\text{recent}}\}, p) \tag{55}$$

where $p \in \{0.5, 0.75, 0.9\}$ for low, medium, high ranges respectively.

Tip Instruction Injection: For Jito delivery, Gateway adds tip instructions:

$$\text{Tip}_{\text{jito}} = \begin{cases} 0.0001 \text{ SOL} & \text{if range} = \text{low} \\ 0.001 \text{ SOL} & \text{if range} = \text{medium} \\ 0.01 \text{ SOL} & \text{if range} = \text{high} \\ \text{dynamic} & \text{if range} = \text{max} \end{cases} \tag{56}$$

2.6.2 Multi-Path Delivery

SolCircle configures multiple delivery methods to maximize transaction landing probability:

$$P(\text{land}) = 1 - \prod_{d \in \mathcal{D}} (1 - P_d(\text{land})) \tag{57}$$

where $P_d(\text{land})$ is the landing probability for delivery method d. With parallel submission to k independent paths:

$$P(\text{land})_{\text{multi}} \geq \max_{d \in \mathcal{D}} P_d(\text{land}) \tag{58}$$

The delivery methods used by SolCircle are:

Sanctum Sender (Primary): Combines SWQoS and Jito simultaneously:

$$P_{\text{sanctum}}(\text{land}) = P_{\text{SWQoS}}(\text{land}) + P_{\text{Jito}}(\text{land}) - P_{\text{SWQoS}}(\text{land}) \cdot P_{\text{Jito}}(\text{land}) \tag{59}$$

This provides dual-path submission with typical landing rates $P_{\text{sanctum}} > 0.95$.

Jito Bundles (Secondary): Direct submission to Jito Block Engine with configurable tips. Transactions participate in tip-based auctions:

$$\text{Priority}_{\text{bundle}} = \frac{\text{Tip}_{\text{jito}}}{\text{CU}_{\text{consumed}}} \tag{60}$$

Higher priority bundles have greater likelihood of inclusion in the next block.

RPC Fallback (Tertiary): Standard RPC submission to Triton/Quicknode for additional redundancy.

2.6.3 Transaction Lifecycle with Gateway

The complete flow for a SolCircle pool trade execution is:

1. Proposal passes governance vote with parameters $(T_{\text{in}}, T_{\text{out}}, s_{\max})$
2. Backend constructs unsigned transaction T_{raw} with DEX swap instruction
3. Call `buildGatewayTransaction`(T_{raw}, θ) to optimize
4. Gateway returns $T_{\text{optimized}}$ with:
   - Latest blockhash from Solana network
   - Optimal CU limit from simulation
   - Priority fee based on current network congestion
   - Jito tip instructions for bundle delivery
5. Pool PDA signs $T_{\text{optimized}}$ using PDA derivation
6. Submit via `sendTransaction` to Gateway endpoint
7. Gateway delivers via multiple paths simultaneously
8. Monitor transaction status until confirmation or timeout

2.6.4 Cost Optimization

Gateway's intelligent routing minimizes costs while maximizing success rates. The expected cost is:

$$E[\text{Cost}] = \text{BaseFee} + \text{CU}_{\text{consumed}} \cdot \text{CU}_{\text{price}} + E[\text{Tip}] \tag{61}$$

where:

$$E[\text{Tip}] = \sum_{d \in \mathcal{D}} w_d \cdot \text{Tip}_d \tag{62}$$

$$\sum_{d \in \mathcal{D}} w_d = 1 \tag{63}$$

Gateway's weighted sampling algorithm $\{w_d\}$ is configured per-project on the Sanctum Dashboard, allowing SolCircle to balance cost vs. success rate based on pool preferences.

2.6.5 Latency Guarantees

By leveraging Sanctum Sender, SolCircle achieves:

$$T_{\text{confirmation}} = T_{\text{build}} + T_{\text{sign}} + T_{\text{deliver}} + T_{\text{block}} \tag{64}$$

where:

- $T_{\text{build}} \approx 100\text{ms}$ (Gateway API latency)
- $T_{\text{sign}} \approx 50\text{ms}$ (PDA signature generation)
- $T_{\text{deliver}} \approx 50\text{ms}$ (Sanctum Sender propagation)
- $T_{\text{block}} \approx 400\text{ms}$ (Solana block time)

This results in typical confirmation latency:

$$T_{\text{confirmation}} \approx 600\text{ms} \tag{65}$$

significantly faster than standard RPC submission ($T_{\text{confirmation}} > 2000\text{ms}$).

2.7 DEX Integration

SolCircle integrates deeply with Raydium and other Solana DEXs to provide efficient trade execution for group pools.

2.7.1 Slippage-Protected Swaps

When executing a swap of amount Δ_{in} of token A for token B, the expected output is:

$$\Delta_{\text{out}}^{\text{expected}} = \frac{R_B \cdot \Delta_{\text{in}} \cdot (1 - f)}{R_A + \Delta_{\text{in}}} \tag{66}$$

where R_A and R_B are the AMM reserves and f is the fee rate. The actual output must satisfy:

$$\Delta_{\text{out}}^{\text{actual}} \geq \Delta_{\text{out}}^{\text{expected}} \cdot (1 - s_{\max}) \tag{67}$$

where $s_{\max}$ is the maximum allowed slippage parameter. If this condition fails, the transaction reverts.

2.7.2 Price Impact Calculation

The price impact of a trade is:

$$I = \frac{P_{\text{effective}} - P_{\text{mid}}}{P_{\text{mid}}} \tag{68}$$

where:

$$P_{\text{mid}} = \frac{R_B}{R_A} \tag{69}$$

$$P_{\text{effective}} = \frac{\Delta_{\text{out}}}{\Delta_{\text{in}}} \tag{70}$$

The pool contract enforces:

$$|I| \leq I_{\max} \tag{71}$$

to prevent trades from executing at unfavorable prices.

The integration includes intelligent liquidity routing across multiple DEX pools to minimize slippage and trading costs, with the system analyzing available liquidity across different venues before executing each trade. Limit order functionality allows groups to execute trades at target prices without requiring active monitoring, with orders remaining open until filled or cancelled by subsequent governance actions.

Integration with yield optimization protocols allows pools to earn returns on idle capital while maintaining availability for trading opportunities. The system automatically identifies yield opportunities that match the pool's risk parameters and liquidity requirements, deploying funds to earn interest, staking rewards, or liquidity provision fees during periods when trading activity is low. These yield strategies are transparent to pool members, who can see exactly where funds are deployed and what returns are being generated. Yield positions can be rapidly unwound when trading opportunities arise, ensuring that pursuit of passive returns does not interfere with active trading strategies.

Trade execution incorporates multiple layers of protection to ensure that trades execute as intended and that pools are not exploited through front-running or sandwich attacks. The system uses private transaction submission where available to prevent mempool observation by potential attackers. Randomized execution delays within acceptable ranges make timing-based attacks more difficult while still allowing timely trade execution. Comprehensive slippage protection aborts trades if market impact exceeds governance-defined thresholds, protecting the pool from executing trades at unfavorable prices. Gas price limits prevent trades from being executed at unreasonably high costs that would erode profits, with the system waiting for favorable gas conditions unless urgency parameters dictate immediate execution.

3 Security Model

SolCircle implements defense-in-depth security across multiple layers of the architecture, with each layer providing independent protections against different attack vectors. At the smart contract level, critical contract functions undergo formal verification to prove they satisfy specified safety properties, providing mathematical guarantees about contract behavior. Multiple independent security firms conduct comprehensive audits before each version deployment, with all findings addressed before mainnet release. A substantial bug bounty program incentivizes white-hat hackers to discover and report vulnerabilities before they can be exploited maliciously. Contracts use upgradeable proxy patterns with time-locked governance, allowing security fixes while preventing sudden malicious changes that could compromise user funds.

The oracle network employs multiple security mechanisms to ensure data integrity. Oracle operators must stake substantial collateral worth significantly more than potential gains from manipulation, making honest behavior the economically dominant strategy. Reputation systems track oracle reliability over time, with consistent accuracy required to maintain authorization to participate in the network. Multiple independent oracles must agree on critical state transitions, with outliers triggering investigation and potential exclusion from the network. Price feeds incorporate data from multiple external sources with automatic failover if primary sources become unavailable or begin reporting suspicious values.

The Telegram bot implements several protections against attacks targeting the user interface. Users must link their Telegram account to a Solana wallet through cryptographic signature verification, preventing impersonation attacks. Aggressive rate limiting prevents spam attacks and Sybil attacks on voting mechanisms. All bot commands undergo server-side validation before triggering any on-chain action, with malformed or suspicious commands rejected and logged for security analysis. The bot educates users about common phishing techniques and implements verification mechanisms for critical actions like fund withdrawal, requiring multiple confirmation steps for large transfers.

Economic attacks represent a significant threat vector that the protocol addresses through careful mechanism design. Governance capture attacks, where an attacker accumulates a controlling share of pool voting power to manipulate trades for personal gain, are mitigated through optional voting power caps per member, multi-signature requirements for trades exceeding certain size thresholds, and emergency pause mechanisms that can be activated by minority stakeholders when suspicious activity is detected. Oracle manipulation attacks are prevented through the stake-based security model where oracle operators face substantial slashing for providing false attestations, making coordinated manipulation economically irrational.

Front-running attacks where observers of pending pool trades attempt to profit by front-running them on the DEX are addressed through private mempools where available, randomized execution delays that make timing attacks more difficult, and slippage protection that aborts trades if market impact exceeds acceptable thresholds. The system monitors for patterns consistent with front-running and can blacklist suspicious addresses from interacting with pool contracts.

4 Data Architecture and Scalability

SolCircle uses a hybrid on-chain and off-chain data model optimized for cost, performance, and transparency. Critical state including account balances, open positions, active proposals, voting results, and fund movements is stored on-chain to ensure immutability and verifiability of essential data. Off-chain storage handles user preferences, chat history, analytics, and historical data too voluminous to store on-chain economically. This data is stored in distributed databases with cryptographic proofs anchoring it to on-chain state, allowing users to verify data integrity while avoiding prohibitive storage costs. A caching layer maintains frequently accessed data to minimize blockchain queries and ensure responsive user experience in the Telegram interface.

The architecture leverages Solana's high throughput and low latency to support thousands of simultaneous pools operating independently. Pool operations are designed to avoid shared state where possible, allowing Solana's parallel transaction processing to maximize throughput by executing multiple pool operations concurrently. State compression techniques reduce on-chain storage costs, making small pools with modest capital economically viable by minimizing the per-pool overhead required to maintain on-chain state. Where appropriate, operations like profit distribution are batched to minimize transaction count and associated costs, with the system aggregating multiple individual distributions into single transactions where possible.

The architecture is designed to integrate with Solana's evolving scaling solutions, including upcoming Layer 2 protocols and state compression improvements. The modular design allows

new scaling technologies to be incorporated without requiring changes to core pool logic or user-facing interfaces. As Solana's ecosystem matures and new scaling primitives become available, SolCircle can adopt them to further reduce costs and increase capacity while maintaining the security and transparency guarantees that make the system trustless.

5 Technical Implementation Details

The technical implementation of SolCircle involves careful attention to numerous details that ensure the system operates correctly under all conditions. Account structures use versioning to support protocol upgrades, with each account storing a version number that determines which instruction set can operate on it. When protocol upgrades introduce new account structures or functionality, existing accounts can be migrated through explicit upgrade instructions that users execute, maintaining backward compatibility while allowing the protocol to evolve.

Transaction construction requires careful attention to Solana's account model and transaction size limits. Complex operations are decomposed into multiple transactions when necessary, with the system ensuring atomic execution semantics through intermediate state locks and rollback mechanisms. Instructions are ordered to maximize parallelization potential, allowing the Solana runtime to execute multiple operations concurrently when they do not conflict on account access.

Error handling throughout the system ensures that failures are detected, logged, and handled gracefully without leaving the system in inconsistent states. All program instructions validate inputs comprehensively before making any state changes, preventing invalid data from corrupting on-chain state. Failed transactions revert all state changes, ensuring that partial execution cannot leave pools in inconsistent states. The system logs all errors with detailed context, enabling debugging and security analysis of failures.

Cross-program invocations to DEXs and other protocols include comprehensive error handling to deal with failures in external systems. When external calls fail, the pool contract reverts its state changes and notifies users of the failure reason. The system retries failed operations with exponential backoff for transient failures, while permanently failing operations that encounter non-recoverable errors. Users can query the status of pending operations through the Telegram bot, receiving detailed information about execution progress and any errors encountered.

6 Use Cases and Applications

The SolCircle infrastructure enables numerous use cases that were previously impractical or impossible in decentralized systems. Trading signal groups can operate transparently where experienced traders provide signals and members allocate capital to a pool that executes approved trades automatically. The signal provider earns performance fees without ever having custody of member funds, while members benefit from the provider's expertise without trust requirements. All trades are on-chain and verifiable, eliminating disputes about execution quality or profit attribution.

Community investment clubs can pool capital from friends, colleagues, or online communities for larger trades or to access investment opportunities unavailable to individual small investors. Groups create pools with custom governance rules, such as requiring supermajority votes for trades, setting maximum position sizes per trade, or restricting trades to specific asset classes. All members verify fund usage and performance in real-time through on-chain data, eliminating the need for trusted treasurers or fund managers.

Copy trading communities form around successful traders who want followers to replicate their strategies. Traders create public pools that automatically mirror their personal trading wallet, with followers depositing funds that move proportionally with the trader's positions. All trades are on-chain and verifiable, with the trader earning a percentage of follower profits

without custody of their funds. This eliminates the trust requirements and high fees associated with centralized copy trading platforms while providing superior transparency.

DAO treasury management becomes more efficient through SolCircle pools. DAOs often struggle with active treasury trading strategies because standard DAO governance is too slow for rapid market-responsive trading. DAOs can delegate specific treasury allocations to SolCircle pools managed by elected traders or trading committees, maintaining ultimate oversight through oracle-based approval mechanisms while enabling day-to-day trading flexibility. The DAO can observe all trading activity in real-time and revoke delegation if performance or behavior becomes unacceptable.

7 Future Technical Directions

Several technical enhancements are planned to expand SolCircle's capabilities while maintaining its security and decentralization properties. Cross-chain pool functionality will enable pools to hold and trade assets across multiple blockchains, using bridge protocols and cross-chain messaging to coordinate trades while maintaining security guarantees. Members will be able to contribute assets from different chains to a single pool, with the system managing cross-chain settlement and position tracking transparently.

Advanced trading strategies will be implemented as composable modules that pools can activate through governance. These include dollar-cost averaging strategies that automatically execute periodic purchases regardless of price, grid trading strategies that place multiple buy and sell orders at different price levels, and automated rebalancing that maintains target allocation ratios across multiple assets. Each strategy module is audited independently and can be enabled or disabled by pool governance.

Derivatives trading support will allow pools to take leveraged positions and implement hedging strategies. Integration with Solana derivatives protocols will enable pools to trade perpetual futures, options, and other derivative products while maintaining the same governance and transparency guarantees as spot trading. Derivative positions introduce additional complexity in risk management and liquidation, requiring careful protocol design to ensure pool solvency under all market conditions.

AI-powered analytics will provide pools with insights derived from on-chain data and market conditions. Machine learning models will analyze historical trading patterns to identify successful strategies, predict optimal entry and exit points, and detect potential market manipulation. These analytics will be available to all pools as optional tools that inform but do not replace human decision-making and governance processes.

8 Conclusion

SolCircle addresses a fundamental problem in cryptocurrency trading: the lack of trustless infrastructure for group coordination. By integrating Telegram's social features with Solana's high-performance blockchain and decentralized oracles, we enable trading groups to operate with unprecedented transparency and security. The technical architecture presented in this paper demonstrates how blockchain primitives can solve real coordination problems without sacrificing the social dynamics that make these communities effective.

The system's design balances multiple competing requirements: rapid trade execution with democratic accountability, capital efficiency with security, and ease of use with protocol decentralization. Each component of the architecture contributes to this balance, from Program Derived Addresses that provide non-custodial fund management to oracle networks that verify outcomes without centralized trust. The result is infrastructure that transforms how groups coordinate financial activity, replacing trust with cryptographic guarantees while maintaining the flexibility and responsiveness that trading requires.

SolCircle represents more than a useful tool for trading groups. It demonstrates how blockchain technology can enable new forms of coordination that were previously impossible, where groups of any size can coordinate effectively without surrendering control to centralized intermediaries. The same principles that make SolCircle work for trading could extend to investment clubs, charitable organizations, and any domain where collective action faces trust problems. This is the beginning of a new paradigm for group coordination in financial markets and beyond.

Technical Documentation:

GitHub: https://github.com/solcircle