Kora Rent Reclaimer - Deep Dive

Overview

This document explains the design, architecture, and reasoning behind the Kora Rent Reclaimer bot. It covers how the bot solves the rent-locking problem and the safety mechanisms that make it reliable.

The Problem: Silent Capital Loss

Why Rent Gets Locked

When a Kora node sponsors a transaction that creates accounts (e.g., token accounts for first-time token recipients), the operator pays for:

1. **Transaction fees** - Immediately consumed
2. **Rent** - Locked in the created accounts indefinitely

For example:

- Creating 1,000 Associated Token Accounts (ATAs) locks ~2.03 SOL in rent
- If 70% of those accounts are later closed or become inactive, ~1.4 SOL remains recoverable
- Most operators never realize this SOL is there — it's silent capital loss

Current Operator Pain Points

1. **No visibility** - Operators can't easily see how much rent is locked where
2. **Manual reclaim** - Reclaiming requires finding accounts, checking authority, and manually closing them
3. **Authority confusion** - Many operators don't understand that they can't close accounts they didn't create the `closeAuthority` for
4. **Risk management** - No way to safely experiment; one mistake could drain multiple accounts

---

The Solution: Automated Safe Reclaim

The Kora Rent Reclaimer bot automates the entire process while maintaining strict safety controls:

Core Flow

```
1. Account Discovery
   └─> Import from JSON, discover from fee payer history, or parse Kora logs

2. Authority Verification
   └─> Scan each account's closeAuthority field
   └─> Only flag accounts where operator is the authority

3. Eligibility Check
   └─> Confirm account is empty or closed
   └─> Apply safety filters (whitelist, idle time, budget limits)
```

```
4. Dry Run (Optional)
   └─> Show what would be reclaimed without executing

5. Execute Reclaim
   └─> Batch close instructions into efficient transactions
   └─> Sign and send to blockchain
   └─> Return rent SOL to treasury

6. Logging
   └─> Record every account (reclaimed, skipped, failed)
   └─> Generate audit trail for compliance
```

Architecture

High-Level Design

```
┌─────────────────────────────────────────────────────────────┐
│                     CLI Interface                           │
│             (Commander-based CLI framework)                  │
├─────────────────────────────────────────────────────────────┤
│                  Control Layer (index.ts)                    │
│ - Route commands (scan, reclaim, import, discover, etc)     │
│ - Orchestrate workflow                                       │
├─────────────────────────────────────────────────────────────┤
│             Business Logic (Core Modules)                    │
│                                                              │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐      │
│  │  Scanner     │  │  Reclaimer   │  │  Safety      │      │
│  │  ─────────   │  │  ──────────  │  │  ──────      │      │
│  │ - Fetch      │  │ - Create txs │  │ - Whitelist  │      │
│  │   accounts   │  │ - Batch ops  │  │ - Idle check │      │
│  │ - Verify     │  │ - Send & log │  │ - Budget cap │      │
│  │   authority  │  │              │  │              │      │
│  └──────────────┘  └──────────────┘  └──────────────┘      │
│                                                              │
│  ┌──────────────┐  ┌──────────────┐  ┌──────────────┐      │
│  │  Database    │  │  Logger      │  │  Alerts      │      │
│  │  ────────    │  │  ──────      │  │  ──────      │      │
│  │ - Track      │  │ - JSON logs  │  │ - Telegram   │      │
│  │   accounts   │  │ - Console    │  │ - Email      │      │
│  │ - History    │  │   output     │  │   (future)   │      │
│  └──────────────┘  └──────────────┘  └──────────────┘      │
├─────────────────────────────────────────────────────────────┤
│               Infrastructure Layer                           │
│                                                              │
│  ┌──────────────────┐          ┌──────────────────┐        │
│  │  Solana RPC      │          │  SQLite DB       │        │
│  │  ──────────────  │          │  ──────────────  │        │
│  │ - Account lookup │          │ - Account state  │        │
│  │ - TX submission  │          │ - Reclaim logs   │        │
│  │ - Confirmation   │          │ - Whitelist      │        │
│  └──────────────────┘          └──────────────────┘        │
└──────────────────────────────────────────────────────────────┘
```

Module Breakdown

`src/index.ts` - CLI Entry Point

- Uses Commander.js for command routing
- Implements commands: `scan`, `reclaim`, `status`, `import`, `discover`, `whitelist`, `cron`
- Handles global flags: `-n` (network), `-d` (dry-run), `-v` (verbose)

`src/config.ts` - Configuration Management

- Loads `.env` file (prefers local for testing)
- Validates required fields (RPC URL, wallet path, treasury)
- Exposes config as a singleton
- Handles different network endpoints (devnet vs mainnet-beta)

`src/scanner.ts` - Account Detection

Purpose: Identify reclaimable accounts and verify operator authority

Key Logic:

```
1. Fetch account state from RPC (in batches for efficiency)
2. Check if account owner is TOKEN_PROGRAM_ID (token account)
3. Decode token account to extract closeAuthority field
4. Compare closeAuthority with operator public key
5. Apply safety checks (whitelist, idle time)
6. Return reclaimable accounts
```

Why Authority Verification Matters:

- Solana prevents unauthorized account closure
- Only the `closeAuthority` can call `closeAccount`
- Operator can only reclaim accounts they have authority over
- Most sponsored accounts are user-owned (operator has no authority)

`src/safety.ts` - Risk Management

Purpose: Prevent accidental reclaims of active/important accounts

Controls:

1. **Whitelist** - Accounts that should never be touched
2. **Idle Time** - Only reclaim if account inactive for N days

3. **Budget Cap** - Max SOL to reclaim per run (default 10)
4. **Reclaimable State Tracking** - Only count idle duration from when account became empty

Example:

- Account created: Jan 1
- Account emptied: Jan 15 (reclaimable_since = Jan 15)
- MIN_IDLE_DAYS = 7
- Can reclaim starting: Jan 22 (not Jan 8)

`src/reclaim.ts` - Rent Recovery Execution

Purpose: Safely close accounts and recover rent SOL

Process:

```
1. Group accounts into batches (up to 10 per transaction)
2. Create closeAccount instructions for each account
3. Sign transaction with operator keypair
4. Send to Solana blockchain
5. Confirm transaction
6. Log result (success, failure, tx signature)
7. Update database with reclaim timestamp
```

Why Batching?

- Single transaction costs 5,000 lamports (~0.000005 SOL)
- Batching 10 closes = 1 tx instead of 10 (9 × 5,000 = 45,000 lamports saved)
- More efficient, lower fees

`src/database.ts` - State Persistence

Purpose: Track accounts and audit history

Tables:

- `sponsored_accounts` - Account state, authority info, timestamps
- `reclaim_history` - Every close attempt (success/failure/skip)
- `whitelist` - Protected accounts

Why SQLite?

- Lightweight (sql.js in JavaScript)
- Works offline (no external DB dependency)
- Suitable for CLI tools
- Full audit trail in single database file

`src/logger.ts` - Observability

Purpose: Record every action for audit and debugging

Outputs:

1. **Console** - Real-time feedback during execution
2. **JSON Logs** - `logs/reclaim-YYYY-MM-DD.json` with full details
3. **Telegram** (optional) - Alerts on large reclaims or errors

Why JSON Logs?

- Machine-parsable (can integrate with dashboards)
- Immutable audit trail
- Easy to search and analyze

`src/kora.ts` - Account Discovery

Purpose: Find sponsored accounts to reclaim

Methods:

1. **Import from file** - JSON array of pubkeys
2. **Discover from fee payer** - Query tx history for accounts created by operator
3. **Parse Kora logs** - Future: integrate with Kora endpoint APIs

---

Design Decisions

1. Authority-First Approach

Decision: Only close accounts where operator is `closeAuthority`

Rationale:

- Solana's permission model requires this for safety
- Prevents accidental/malicious closure of user accounts
- Aligns with Kora's design (operator sponsors but doesn't own accounts)

Trade-off:

- Can't recover rent from user-owned accounts
- But this is correct behavior — users own their accounts

2. Idle Time Before Reclaim

Decision: Require accounts to be inactive for N days before reclaim

Rationale:

- Catches accounts that are temporarily empty but will be reused
- Gives time to discover errors before irreversible closure
- Matches operator expectations (don't touch active accounts)

Configuration:

- Default: 7 days for production

- Devnet: 0 days for testing
- Configurable via `MIN_IDLE_DAYS` env var

3. Batched Transactions

Decision: Batch up to 10 close instructions per transaction

Rationale:

- Reduces on-chain footprint and fees
- Respects transaction size limits
- Balances efficiency vs risk (single large tx vs many small txs)

Limits:

- Max 10 per batch (conservative estimate)
- Transaction size ~1.2 KB (well below 1232 KB limit)
- Can adjust if needed

4. Dry-Run Mode

Decision: Always allow `--dry-run` before execution

Rationale:

- Critical safety feature
- Operators see exactly what will happen
- Zero risk of bugs affecting real transactions
- Builds confidence before mainnet

5. SQLite for Persistent State

Decision: Use sql.js (SQLite in JavaScript) instead of external DB or in-memory

Rationale:

- CLI tools need offline operation
- Single-file database easy to backup
- No external services required
- Full SQL queries for audit

Trade-off:

- Not suitable for multi-process (but CLI is single-threaded)
- Slower than relational DB for huge datasets (acceptable for operators)

Safety Mechanisms

Defense in Depth

```
┌─────────────────┐
│   User Input    │
└────────┬────────┘
         │
         v
┌─────────────────┐       ┌──────────────────┐
│  .env Validation│───────│ Throw if invalid │
└────────┬────────┘       └──────────────────┘
         │
         v
┌─────────────────┐       ┌──────────────────┐
│  Keypair Load   │───────│ Throw if missing │
└────────┬────────┘       └──────────────────┘
         │
         v
┌─────────────────────────────────────────────────┐
│            Scan & Verify Authority              │
│  - Fetch account from RPC                       │
│  - Decode token account                         │
│  - Extract closeAuthority                       │
│  - Compare with operator pubkey                 │
│  - Skip if no match (logged)                    │
└────────┬────────────────────────────────────────┘
         │
         v
┌─────────────────────────────────────────────────┐
│            Apply Safety Filters                 │
│  - Whitelist check (skip if listed)             │
│  - Idle time check (skip if too recent)         │
│  - Budget check (skip if would exceed limit)    │
│  - Account exists check (skip if already closed)│
└────────┬────────────────────────────────────────┘
         │
         v
┌─────────────────────────────────────────────────┐
│             Dry-Run Mode                        │
│  - Show what would happen                       │
│  - No transactions sent                         │
│  - Full logging of simulation                   │
└────────┬────────────────────────────────────────┘
         │
         v
┌─────────────────────────────────────────────────┐
│           Manual Confirmation                   │
│  - Mainnet: prompts "are you sure?"            │
│  - Devnet: proceeds automatically              │
│  - Flag: -y to skip (explicit opt-in)         │
└────────┬────────────────────────────────────────┘
         │
         v
┌─────────────────────────────────────────────────┐
│           Create Transaction                    │
│  - Batch instructions (max 10)                  │
│  - Sign with operator keypair                   │
│  - Include recent blockhash                     │
└────────┬────────────────────────────────────────┘
         │
         v
┌─────────────────────────────────────────────────┐
│           Submit & Confirm                      │
│  - Send to RPC                                  │
│  - Wait for confirmation                        │
│  - Retry logic with exponential backoff         │
│  - Log tx signature                             │
└────────┬────────────────────────────────────────┘
         │
         v
┌─────────────────────────────────────────────────┐
│           Update Database                       │
│  - Mark accounts as reclaimed                   │
│  - Record timestamp & tx signature              │
│  - Store outcome (success/failure)              │
└────────┬────────────────────────────────────────┘
         │
         v
┌─────────────────────────────────────────────────┐
│           Log & Alert                           │
│  - Write JSON log                               │
│  - Send Telegram notification (optional)        │
│  - Print summary to console                     │
└─────────────────────────────────────────────────┘
```

Edge Cases Handled

1. **Invalid Keypair File** - Throws error, doesn't proceed
2. **Zero-Balance Accounts** - Reclaims (empty accounts still hold rent)
3. **Already-Closed Accounts** - Skips (logged as closed)
4. **User-Owned Accounts** - Skips (operator not authority)
5. **Network Latency** - Retries with exponential backoff
6. **Insufficient Fees** - Fails gracefully, logged
7. **Whitelist Bypass** - Impossible (checked first)
8. **Budget Overflow** - Stops batching once limit reached

Testing Strategy

Unit Testing (Planned)

```
// Test authority detection
it('should skip accounts where operator is not closeAuthority')

// Test safety filters
it('should skip whitelisted accounts')
it('should skip accounts idle < MIN_IDLE_DAYS')

// Test batch logic
it('should batch max 10 accounts per transaction')

// Test database
it('should persist and recover reclaim history')
```

Integration Testing (Verified on Devnet)

1. **Account Creation** - Generate test ATAs with operator as closeAuthority
2. **Import** - Import test accounts into bot
3. **Scan** - Verify detection and authority check
4. **Dry-Run** - Confirm output without side effects
5. **Execute** - Close accounts and verify SOL returned

Test Results:

- Created 4 test ATAs (~0.008 SOL total rent)
- Scanned and detected as reclaimable
- Dry-run showed correct amounts
- Executed reclaim successfully
- Rent returned to treasury
- Database logged all transactions

Manual Testing Checklist

- ☑ Build on Windows, Linux, macOS
- ☑ Create test accounts on Devnet
- ☑ Verify authority detection works
- ☑ Test dry-run mode
- ☑ Execute real reclaim
- ☑ Verify logs and database
- ☑ Test with invalid keypair (error handling)
- ☑ Test with insufficient SOL (graceful failure)

Performance Considerations

Efficiency

Batch Scanning:

- Uses `getMultipleAccountsInfo` (RPC batch call)
- Requests 100 accounts per RPC call
- Much faster than sequential `getAccountInfo`

Transaction Batching:

- Up to 10 closeAccount instructions per transaction

- ~1.2 KB per transaction (well under limit)
- Saves ~45,000 lamports in fees per batch

Database Queries:

- Indexed by pubkey for O(1) lookups
- Indexed by status for bulk operations
- Suitable for operators with 10K+ accounts

Scalability Limits

Current Design:

- Tested with up to 100+ accounts per scan
- Database can handle millions of records
- RPC rate limits: respect 100ms delay between batches

Known Bottlenecks:

- Single-threaded (sequential processing)
- No caching of RPC responses
- Could optimize with parallel requests (future)

Security Considerations

Private Key Handling

Current:

- Keypair loaded from local JSON file
- Never transmitted over network
- Kept in memory only during tx signing
- Cleared after use (JS garbage collection)

Improvements:

- Support hardware wallets (Ledger)
- Support key derivation from seed phrase
- Optional encryption at rest

RPC Security

Current:

- Uses standard Solana RPC (public endpoint)
- No authentication required (standard practice)
- Assumes honest RPC (not validating every response)

Improvements:

- Support private RPC endpoints

- Validate RPC responses against network
- Fallback RPC endpoints

Transaction Signing

Current:

- Signs with operator keypair
- Includes recent blockhash
- Standard Solana security model

No Known Vulnerabilities:

- Not vulnerable to replay attacks (blockhash prevents)
- Not vulnerable to MitM (blockchain verifies sig)
- Requires operator's private key to sign

Lessons Learned

What Worked Well

1. **Authority-first design** - Prevents the vast majority of bugs
2. **Dry-run mode** - Gave confidence before real execution
3. **Detailed logging** - Made debugging easy
4. **SQLite persistence** - Reliable and simple
5. **Batch operations** - Efficient and clear

What Could Improve

1. **No unit tests yet** - Should add before production
2. **Limited error recovery** - Could retry transient failures
3. **No dashboard** - Would help visibility
4. **Single RPC** - Should support fallback endpoints
5. **Manual whitelisting** - Could auto-detect active accounts

---

Future Enhancements

Short Term

- ☐ Add Jest test suite
- ☐ Implement cron mode (runs periodically)
- ☐ Add email alerts
- ☐ Support Ledger hardware wallets
- ☐ Add metrics/stats endpoint

Medium Term

- ☐ Web dashboard for monitoring
- ☐ Multi-network support (all RPCs simultaneously)

- □ Account lifecycle tracking (age, balance history)
- □ Predictive analytics (estimate rent over time)

Long Term

- □ Kora API integration (pull sponsored account list)
- □ Rust rewrite for performance
- □ On-chain program for advanced reclaim logic
- □ Decentralized operator network

Conclusion

The Kora Rent Reclaimer bot solves a real operational problem for Kora operators by:

- **Automating** the tedious process of finding and closing accounts
- **Verifying** that only reclaimable accounts are touched
- **Logging** every action for transparency and compliance
- **Protecting** against accidental reclaim of active accounts

The design prioritizes safety over features, making it suitable for production use once tested thoroughly on real Kora-sponsored accounts.