

Kora Rent Reclaimer: Technical Deep-Dive

Executive Summary

This document provides an in-depth technical analysis of the Kora Rent Reclaim Bot - an automated system for recovering rent-locked SOL from Solana accounts sponsored by Kora node operators. We explore the underlying Solana economics, the technical implementation, and real-world use scenarios.

Table of Contents

1. [Understanding Solana Rent Economics](#)
 2. [The Kora Paymaster Model](#)
 3. [Why Rent Gets Locked \(The Problem\)](#)
 4. [Authority vs Fee Payer \(Critical Distinction\)](#)
 5. [Technical Architecture](#)
 6. [Use Case Scenarios](#)
 7. [Implementation Deep-Dive](#)
 8. [Safety Mechanisms](#)
 9. [Operational Considerations](#)
-

Understanding Solana Rent Economics

What is Rent on Solana?

Solana uses a **rent-based storage model** where every account must maintain a minimum SOL balance to cover the cost of storing its data on-chain. This is called "rent exemption."

Rent Exemption Formula:

$$\text{rent_exempt_minimum} = (\text{account_size_bytes} + 128) * \text{rent_per_byte} * 2_years$$

For a standard Token Account (165 bytes):

$$\text{rent} = (165 + 128) * 0.00000348 * 730 = \sim 0.00203 \text{ SOL}$$

Rent Collection vs Rent Exemption

Solana offers two modes:

1. **Rent-Paying:** Account pays rent each epoch, eventually depletes
2. **Rent-Exempt:** Account holds enough SOL to be exempt forever (2 years worth)

Most accounts are rent-exempt, meaning the rent SOL is **locked** until the account is closed.

Rent Recovery

When an account is **closed** on Solana:

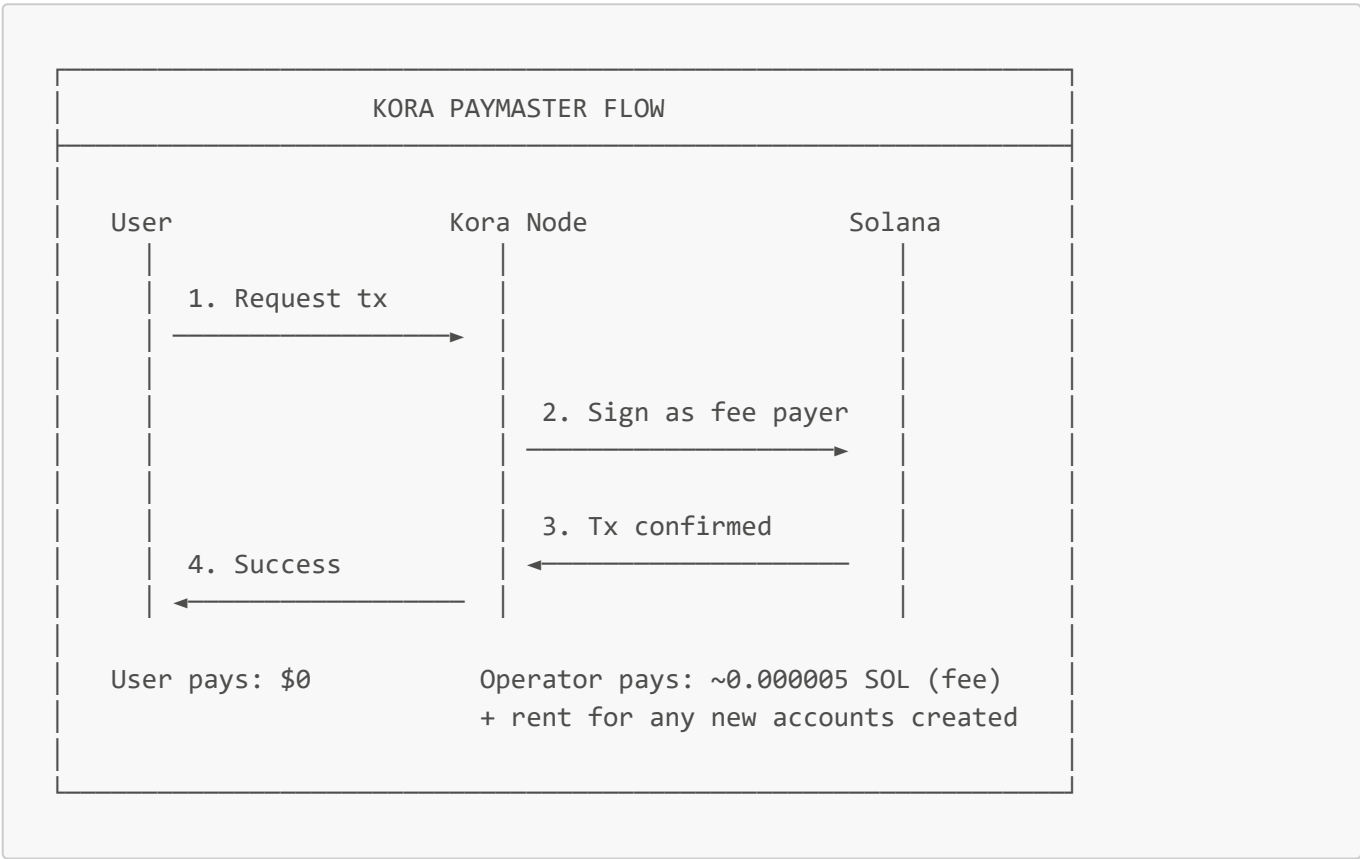
- 1. All account data is zeroed
- 2. The rent-exempt lamports are transferred to a destination address
- 3. The account ceases to exist

This is the mechanism we exploit for rent reclamation.

The Kora Paymaster Model

What is Kora?

Kora is a **paymaster service** for Solana. It allows applications to sponsor transaction fees for their users, enabling gasless experiences.



The Hidden Cost: Rent

When a Kora-sponsored transaction creates new accounts, the **fee payer (operator)** pays for:

- 1. Transaction fee (~0.000005 SOL) - consumed, cannot be recovered
- 2. **Account rent** (~0.002 SOL per token account) - locked but recoverable

Why Rent Gets Locked

Common Rent-Locking Scenarios

Scenario	Account Type	Rent Cost	Created When
Token Transfer	Associated Token Account (ATA)	~0.00203 SOL	First transfer to a wallet

Scenario	Account Type	Rent Cost	Created When
NFT Mint	Mint Account	~0.00145 SOL	New token/NFT creation
NFT Metadata	Metadata Account	~0.0056 SOL	Metaplex metadata
Program Data	PDA	Variable	App-specific accounts

The Silent Capital Drain

Consider an operator sponsoring 10,000 transactions over 6 months:

Scenario: 10,000 sponsored transactions

- 5,000 created new ATAs
 - $5,000 \times 0.00203 \text{ SOL} = 10.15 \text{ SOL}$ locked
- 1,000 created NFT metadata
 - $1,000 \times 0.0056 \text{ SOL} = 5.60 \text{ SOL}$ locked
- Total Rent Locked: ~15.75 SOL

After 6 months:

- 60% of ATAs still in use (users have tokens)
- 30% of ATAs are empty (users moved tokens out)
- 10% of accounts are closed

Recoverable: $3,000 \text{ empty ATAs} \times 0.00203 = 6.09 \text{ SOL}$

This is capital that can be recovered - if you have authority to close the accounts.

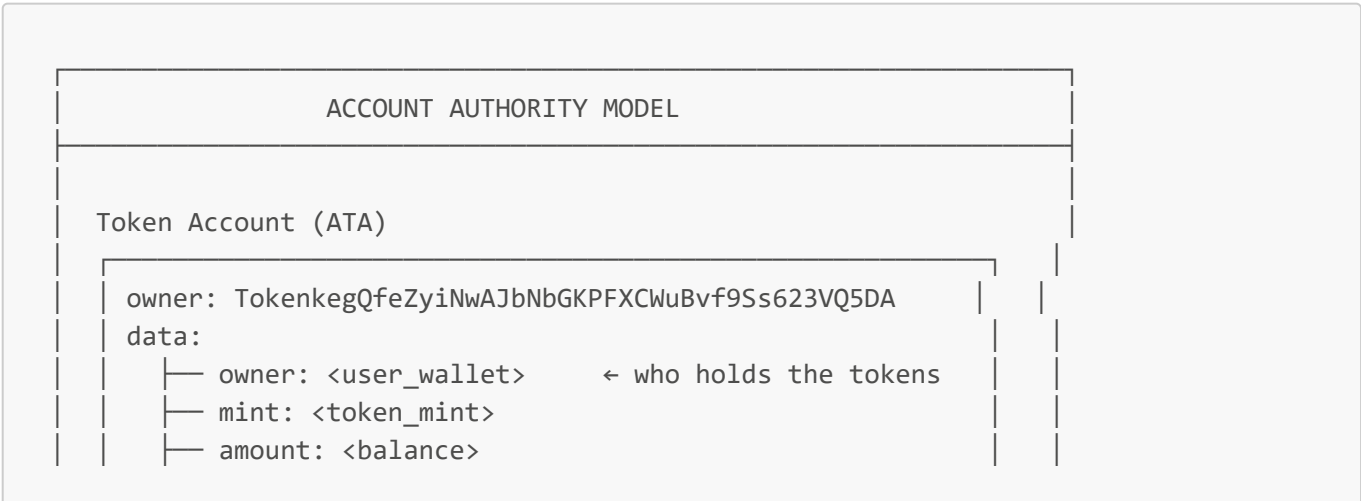
Authority vs Fee Payer (Critical Distinction)

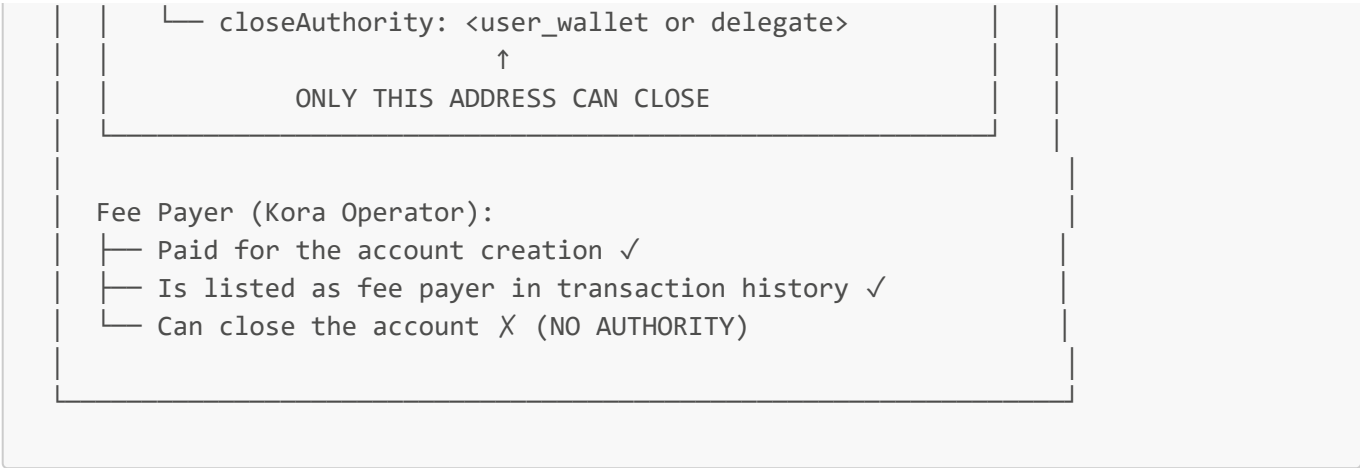
The Core Insight

Paying for an account's creation does NOT grant you the right to close it.

This is the most important concept to understand.

Solana's Permission Model



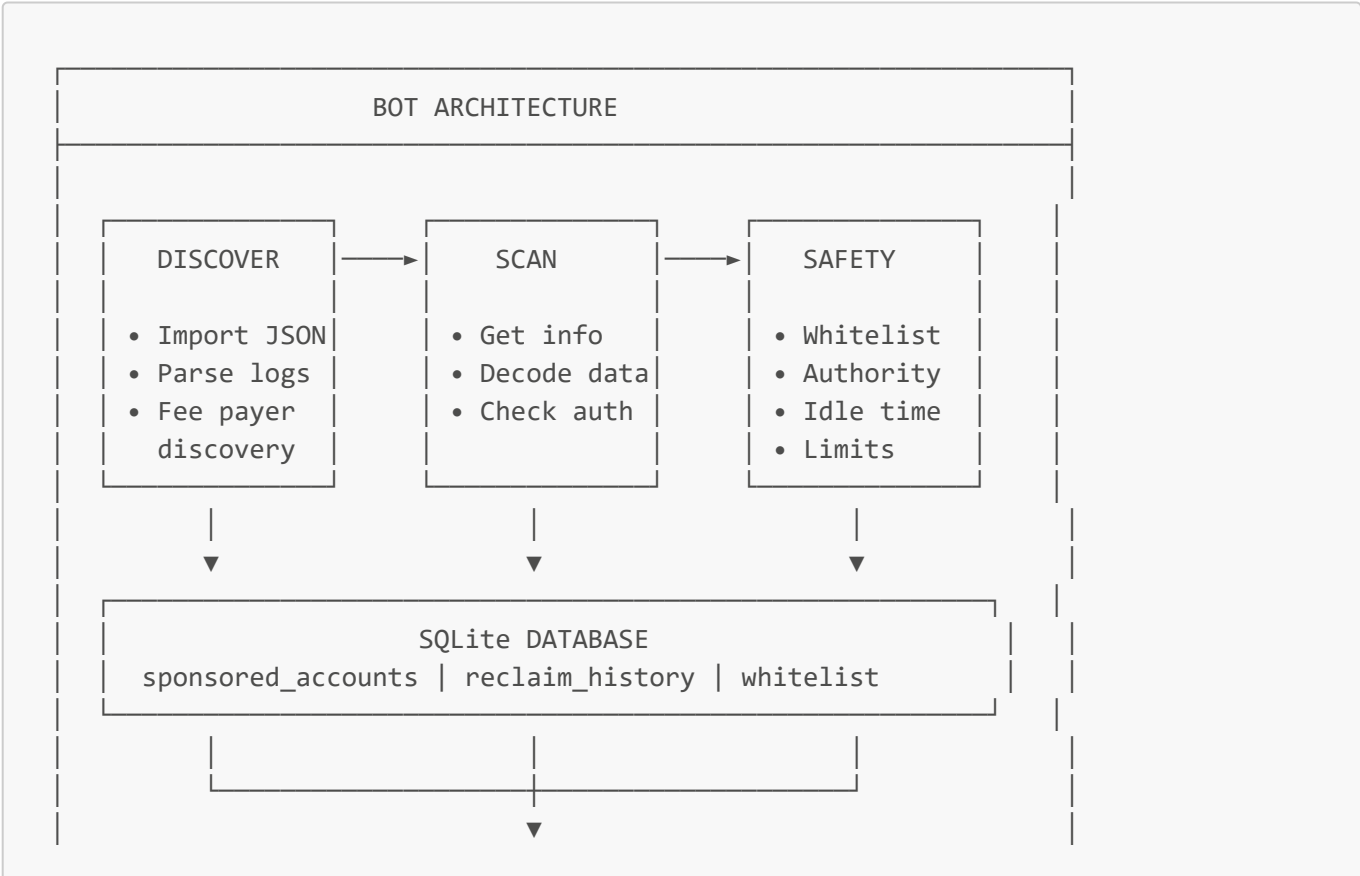


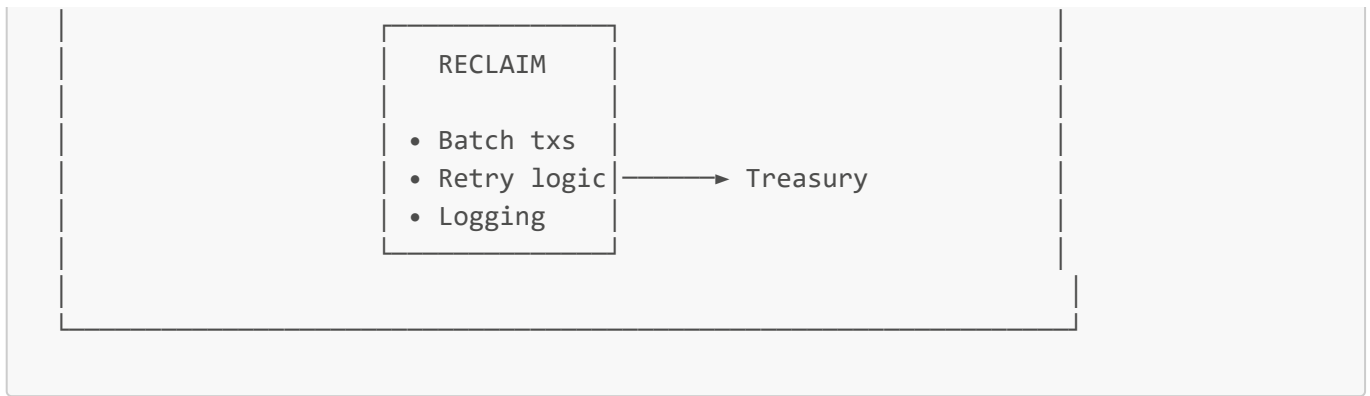
What Can Actually Be Reclaimed?

Account Type	Authority	Can Operator Reclaim?
User's ATA	User wallet	✗ NO
Operator-created ATA (closeAuthority = operator)	Operator	☑ YES
User's System Account	User	✗ NO
Already Closed Account	N/A	📊 Tracking only
Program PDA (operator controls program)	Program/Operator	☑ MAYBE

Technical Architecture

System Overview





Data Flow

1. DISCOVERY PHASE

- └ Import sponsored accounts from JSON
- └ Parse Kora operator logs
- └ Scan fee payer transaction history

2. SCANNING PHASE

- └ Batch fetch account info via RPC
- └ For each token account:
 - └ Decode with AccountLayout
 - └ Extract closeAuthority
 - └ Compare with operator pubkey
- └ Set operatorCanClose = true/false
- └ Update database with authority status

3. SAFETY FILTERING

- └ Check operatorCanClose (MUST be true)
- └ Check whitelist
- └ Check idle duration (reclaimableSince)
- └ Check budget limits

4. RECLAIM EXECUTION

- └ Create closeAccountInstruction for each
- └ Batch into transactions (10 max)
- └ Send with retry logic
- └ Log results to database

Use Case Scenarios

Scenario 1: NFT Marketplace Operator

Context: An NFT marketplace uses Kora to sponsor minting fees. When users mint NFTs, the operator pays for metadata account creation.

- Operator sponsors 5,000 NFT mints over 3 months
 - └ 5,000 metadata accounts created (~0.0056 SOL each)
 - └ Total rent locked: 28 SOL

```
|
After 3 months:
|— 2,000 NFTs burned (metadata accounts still exist but empty)
|— 1,500 NFTs transferred to secondary market
|— 1,500 NFTs still held by original minters
|
IF operator set closeAuthority during mint:
|— Can reclaim  $2,000 \times 0.0056 = 11.2$  SOL
```

Bot Action: Scan all metadata accounts, verify closeAuthority matches operator, close empty ones.

Scenario 2: Token Airdrop Campaign

Context: A project uses Kora to airdrop tokens to 10,000 users. Each user gets an ATA created.

```
10,000 ATAs created
|— Rent locked:  $10,000 \times 0.00203 = 20.3$  SOL
|
After 6 months:
|— 3,000 users claimed and moved tokens
|— 4,000 users claimed and hold tokens
|— 3,000 users never claimed (tokens still there)
|
Standard ATAs (closeAuthority = user):
|— Recoverable by operator: 0 SOL (no authority)
```

Key Insight: For standard user ATAs, the operator CANNOT reclaim rent. This bot correctly identifies and skips those accounts.

Scenario 3: Gaming Platform (Operator-Owned Accounts)

Context: A gaming platform creates temporary token accounts for in-game items. The platform (operator) creates these as THEIR accounts with themselves as authority.

```
Platform creates 1,000 temporary game item ATAs
|— closeAuthority = platform wallet
|— Rent locked: 2.03 SOL
|
After game session:
|— Items consumed, accounts empty
|
Since closeAuthority = operator:
|— Can reclaim 100% = 2.03 SOL
```

Bot Action: Scan, verify authority, close all empty accounts.

Implementation Deep-Dive

Authority Detection (scanner.ts)

```
// 1. Load operator keypair
var operatorPubkey = configManager.loadOperatorKeypair().publicKey;

// 2. For token accounts, decode the data
var decoded = AccountLayout.decode(info.data);

// 3. Determine close authority
// If closeAuthorityOption is set, use that address
// Otherwise, the owner is the default close authority
var closeAuthority = decoded.closeAuthorityOption === 1
  ? new PublicKey(decoded.closeAuthority).toBase58()
  : new PublicKey(decoded.owner).toBase58();

// 4. Check if operator has authority
if (closeAuthority === operatorPubkey.toBase58()) {
  status.operatorCanClose = true;
  status.isReclaimable = true;
}
```

Idle Time Tracking (safety.ts)

```
// Problem: Using lastChecked resets on every scan
// var daysSinceCheck = now - lastChecked; // BAD!

// Solution: Track when account FIRST became reclaimable
// var daysReclaimable = now - reclaimableSince; // GOOD!

// This ensures:
// - Timer starts when account becomes empty
// - Timer resets if account becomes active again
// - Scanning frequently doesn't reset the idle timer
```

Close Instruction (reclaim.ts)

```
// CRITICAL: Verify authority before attempting close
if (!account.operatorCanClose) {
  logger.warn(`Skipping: Operator is not close authority`);
  return null;
}

// For token accounts with zero balance
createCloseAccountInstruction(
  accountPubkey,    // Account to close
  treasury,         // Where rent goes
  authority          // Operator signs (verified to be closeAuthority)
)
```

Safety Mechanisms

Multi-Layer Protection

SAFETY LAYERS

Layer 1: Authority Verification

└─ Cannot close without being closeAuthority

Layer 2: Whitelist

└─ Explicitly protected accounts never touched

Layer 3: Executable Check

└─ Never attempt to close program accounts

Layer 4: Idle Duration

└─ Accounts must be empty for N days (default 7)

Layer 5: Budget Limit

└─ Max SOL per run prevents runaway reclaims

Layer 6: Mainnet Confirmation

└─ Requires explicit "yes" on mainnet

Layer 7: Dry Run Mode

└─ Always test before executing

Operational Considerations

RPC Rate Limiting

Batch Size: 100 accounts per getMultipleAccountsInfo call

RPC Delay: 100ms between batches (configurable)

Retry Logic: 3 attempts with exponential backoff

Database Considerations

Uses sql.js (SQLite in JavaScript) for:

- No external database dependencies

- Portable database file
- Atomic operations

Recommended Workflow

```
# 1. Discover accounts
node dist/index.js discover <fee-payer> --limit 1000 -n devnet

# 2. Scan (read-only)
node dist/index.js scan -n devnet -v

# 3. Dry run reclaim
node dist/index.js reclaim --dry-run -n devnet -v

# 4. Actual reclaim (devnet first!)
node dist/index.js reclaim -n devnet

# 5. Monitor status
node dist/index.js status
```

Cron Automation

```
# Run every 6 hours
node dist/index.js cron --schedule "0 */6 * * *"
```

Conclusion

The Kora Rent Reclaim Bot provides a **safe, automated approach** to recovering rent-locked SOL from Solana accounts. By respecting Solana's permission model and only attempting to close accounts where the operator has rightful authority, it:

1. **Maximizes capital efficiency** for operators who correctly set up their accounts
2. **Provides visibility** into rent expenditure and recovery
3. **Never attempts unauthorized actions** that would fail or cause issues

The key insight for operators: **If you want to reclaim rent, ensure you set yourself as the closeAuthority when creating accounts.** For standard user-owned accounts, the best approach is either incentivizing users to close accounts themselves, or accepting rent as a cost of user acquisition.

Appendix: Key Solana Concepts

PublicKey

A 32-byte identifier for accounts, wallets, and programs on Solana.

Lamport

The smallest unit of SOL. 1 SOL = 1,000,000,000 lamports.

Associated Token Account (ATA)

A deterministically derived account that holds tokens for a specific wallet/mint combination.

CloseAuthority

The address authorized to close a token account and receive its rent.

Program Derived Address (PDA)

An account whose address is derived from a program and seeds, controlled by that program.