

---

# Rumi Protocol

A Decentralized CDP Stablecoin on the Internet Computer

---

**Version 2.0** · February 2026

**Rumi Labs LLC**

[rumiprotocol.com](https://rumiprotocol.com) · [github.com/RumiLabsXYZ/rumi-protocol-v2](https://github.com/RumiLabsXYZ/rumi-protocol-v2)

*This document describes the design, architecture, and economic model of Rumi Protocol. It is not financial advice and does not constitute an offer or solicitation of any kind. Participation in DeFi protocols carries inherent risk including potential loss of funds.*

# Contents

---

## **I. Abstract**

## **II. Introduction**

The Stablecoin Market

Why Build on the Internet Computer

## **III. Protocol Architecture**

System Overview

The Vault System

Price Oracle

icUSD Token

Stability Pool

Treasury

## **IV. Economic Model**

Collateralization and Borrowing

Fee Structure

Liquidation Mechanics

Redemption Mechanism

Stablecoin Repayment and Peg Defense Reserve

Protocol Modes

## **V. Security**

Smart Contract Security

Oracle Security

Economic Security

## **VI. Governance and Decentralization**

Current Model

Path to SNS

Immutable Vault Terms

## **VII. Roadmap and Future Directions**

## **VIII. Conclusion**

## **IX. Glossary**

# I. Abstract

---

Rumi Protocol is a collateralized debt position (CDP) system deployed on the Internet Computer Protocol (ICP). Users lock ICP as collateral in vaults and mint icUSD, a stablecoin pegged to the US dollar, implemented as an ICRC-1 / ICRC-2 compliant token.

The system maintains icUSD's peg through overcollateralization (minimum 133% collateral ratio), an incentivized liquidation system backed by a stability pool, a direct redemption mechanism that creates a hard price floor, and a stablecoin reserve built from ckUSDT and ckUSDC repayments that provides additional peg defense during market stress. Protocol modes automatically adjust parameters during systemic stress, entering Recovery Mode when the system-wide collateral ratio falls below 150%.

The protocol uses the Internet Computer's on-chain Exchange Rate Canister (XRC) for price feeds, eliminating the need for third-party oracle networks. The backend is written in Rust and the frontend in Svelte/TypeScript, with all components deployed as Internet Computer canisters. The protocol supports authentication via Internet Identity, Plug Wallet, and Oisy Wallet.

## II. Introduction

---

### The Stablecoin Market

Total stablecoin market capitalization surged to over \$311 billion by the end of 2025, a 49% increase year-over-year. Regulatory clarity from the GENIUS Act in the United States, the rollout of MiCA in Europe, and accelerating institutional adoption have driven this growth. Monthly transaction volumes exceeded \$1 trillion for the first time in September 2025. Stablecoin issuers are now the seventh-largest purchasers of US government debt.

The market remains heavily centralized. Tether (USDT) and Circle (USDC) together control over 83% of total stablecoin supply. Decentralized, crypto-collateralized stablecoins represent a smaller but important segment. MakerDAO's DAI, the largest decentralized stablecoin at roughly \$5.4 billion in market capitalization, demonstrates that sustained demand exists for the CDP model—borrowing a stable asset against volatile crypto collateral without relying on a centralized issuer.

### Why Build on the Internet Computer

The Internet Computer is a general-purpose blockchain that extends the public internet with serverless compute. It excels at hosting full-stack web applications, managing digital identity, and running autonomous services. As adoption of the Internet Computer grows across these use cases, holders of ICP accumulate a native asset with no native stablecoin to pair it against. Every functional blockchain ecosystem needs DeFi primitives, and a CDP stablecoin is foundational among them. Rumi Protocol exists to fill that gap.

Building DeFi on the Internet Computer comes with trade-offs. The execution model is different from EVM chains: there are no atomic composable transactions across canisters, async inter-canister calls require careful state management, and the ecosystem's DeFi tooling is less mature than Ethereum's. These are real engineering constraints that the protocol's design accounts for.

At the same time, the Internet Computer offers properties that are genuinely useful for a CDP system:

- **On-chain price oracle.** The Exchange Rate Canister (XRC) is a system-level canister that queries major cryptocurrency exchanges via HTTPS outcalls and returns median prices. This eliminates dependence on third-party oracle networks like Chainlink, removing a layer of trust and cost.
- **Reverse gas model.** Smart contracts pay for their own computation using cycles. Users do not pay gas fees per transaction, which means small vault positions remain economical regardless of network congestion.
- **On-chain web hosting.** The protocol's entire frontend is served from canisters, making the application as decentralized as the backend logic. There are no centralized servers or CDN dependencies.
- **Chain-key tokens.** ICP's chain-key cryptography enables trustless representations of Bitcoin (ckBTC), Ethereum (ckETH), and stablecoins (ckUSDT, ckUSDC) without bridge risk, providing a path to multi-collateral support and the stablecoin reserve mechanism described in Section IV.

# III. Protocol Architecture

## System Overview

Rumi Protocol consists of multiple Internet Computer canisters. The core flow: a user deposits ICP collateral into a vault, borrows icUSD against it, and can later repay the debt to retrieve their collateral. The protocol continuously monitors vault health using price data from the Exchange Rate Canister and facilitates liquidation of undercollateralized positions through the stability pool.

Component	Canister	Function
Protocol Backend	<code>rumi_protocol_backend</code>	Vault management, liquidation logic, redemptions, mode control, price feed integration
icUSD Ledger	<code>icusd_ledger</code>	ICRC-1/ICRC-2 token ledger for icUSD supply
Stability Pool	<code>rumi_stability_pool</code>	Depositor pool for automated liquidation; distributes collateral rewards
Treasury	<code>rumi_treasury</code>	Protocol fee collection and reserve management
Vault Frontend	<code>vault_frontend</code>	On-chain web application for vault management
Homepage	<code>rumi_homepage</code>	Public-facing informational site, fully on-chain

Table 1: Canister architecture

## The Vault System

A vault is the fundamental unit of the protocol. Each vault tracks one user's ICP collateral and icUSD debt. Users may open multiple vaults under the same principal.

```
struct Vault {
    owner: Principal,
    borrowed_icusd_amount: ICUSD,
    icp_margin_amount: ICP,
    vault_id: u64,
}
```

Vault operations include opening (depositing ICP), borrowing (minting icUSD), repaying (burning icUSD to reduce debt), adding margin (depositing additional ICP), and closing (repaying all debt and withdrawing collateral). Each operation is protected by per-principal guards that prevent concurrent operations on the same vault, eliminating reentrancy-class vulnerabilities.

The collateral ratio of a vault is: **CR = (ICP Collateral × ICP/USD Price) / icUSD Debt**. A vault must maintain a CR above the Minimum Collateral Ratio (MCR) of 133% under normal conditions, or 150% in Recovery Mode. Vaults below the applicable threshold become eligible for liquidation.

## Price Oracle

Rumi Protocol obtains ICP/USD price data from the Internet Computer's Exchange Rate Canister (XRC), a system canister maintained by DFINITY running on the uzr34 system subnet. The XRC uses HTTPS outcalls to query major cryptocurrency exchanges, filters outliers, and returns the median rate. Because HTTPS outcalls are processed through ICP's consensus mechanism, every replica in the subnet independently fetches the data and must agree on the result.

The protocol fetches prices at configurable intervals and enforces a staleness check: if the most recent price is older than 10 minutes, all vault-modifying operations are paused until a fresh price is obtained. A circuit breaker transitions the system to Read-Only mode if the reported ICP price falls below \$0.01, guarding against oracle malfunction.

## icUSD Token

icUSD is implemented as an ICRC-1 and ICRC-2 compliant token. ICRC-1 provides the base fungible token standard; ICRC-2 adds approve-and-transfer-from functionality required for vault repayment and stability pool deposits. icUSD is minted when users borrow and burned when debt is repaid with icUSD. The total icUSD supply at any time reflects the total outstanding debt across all vaults.

## Stability Pool

The stability pool is a dedicated canister serving as the first line of defense during liquidations. Users deposit icUSD into the pool and receive a proportional share of ICP collateral from liquidated vaults, acquired at a discount to market price (typically 10%). The pool ensures liquidations execute immediately without requiring external liquidator bots to have capital ready.

## Treasury

The treasury canister manages protocol revenue. Borrowing fees and stablecoin repayment fees are directed to the treasury, which maintains custody with controlled withdrawal mechanisms.

## IV. Economic Model

### Collateralization and Borrowing

Parameter	Value	Description
Minimum Collateral Ratio	133%	Floor ratio in normal operation
Recovery Mode MCR	150%	Stricter ratio when system is stressed
Borrowing Fee	0.5%	One-time fee on minted icUSD
Liquidation Bonus	10%	Discount on collateral for liquidators/pool
Minimum Borrow	1 icUSD	Smallest borrowable amount
Redemption Fee	0.5% – 5%	Dynamic; decays over time, increases with volume

Table 2: Core protocol parameters

### Fee Structure

The **borrowing fee** is charged once when icUSD is minted, currently 0.5% of the borrowed amount. This fee is sent to the treasury. In Recovery Mode, the borrowing fee is waived to encourage new collateral deposits that help restore system health.

The **redemption fee** is dynamic, computed from the base rate, elapsed time since the last redemption, and the proportion of total debt being redeemed. It is bounded between 0.5% and 5%. The base rate decays when no redemptions occur (decay factor of 0.94 per hour) and increases with each redemption, discouraging rapid repeated redemptions while keeping the pathway accessible.

The **stablecoin repayment fee** (described below) is a configurable parameter applied when users repay vault debt with ckUSDT or ckUSDC instead of icUSD.

### Liquidation Mechanics

Liquidation resolves undercollateralized vaults to protect icUSD solvency. The protocol supports two liquidation paths:

- **Full Liquidation (Normal Mode).** When a vault's collateral ratio falls below 133%, the vault is liquidated. The stability pool's icUSD repays the debt, and pool depositors receive the ICP collateral at a 10% bonus (i.e., ICP worth 110% of the repaid debt).
- **Partial Liquidation (Recovery Mode).** When the system enters Recovery Mode (total CR below 150%), vaults with collateral ratios between 133% and 150% undergo partial liquidation. The vault's debt is cleared and collateral equal to 133% of the debt value is seized, but excess collateral remains in the vault. This is less punitive to borrowers while still restoring system health.

## Redemption Mechanism

icUSD holders can redeem tokens directly for ICP collateral at face value minus the redemption fee. The protocol targets vaults with the lowest collateral ratios first, deducting both ICP and icUSD debt proportionally. This creates a hard price floor: if icUSD trades below \$1, arbitrageurs can buy it cheaply and redeem for \$1 worth of ICP, profiting from the difference and pushing the price back toward peg.

## Stablecoin Repayment and Peg Defense Reserve

Rumi Protocol allows users to repay vault debt using ckUSDT or ckUSDC (chain-key stablecoins on ICP) instead of icUSD. This addresses a practical problem: in early stages with limited liquidity, users may not be able to acquire enough icUSD on the open market to repay their vaults. Accepting established stablecoins gives them an alternative exit path.

A configurable fee is applied to stablecoin repayments, surcharging the payment rather than discounting the debt reduction. For example, to repay 100 icUSD of debt at a 0.02% fee, the user pays 100.02 ckUSDT; the vault debt is reduced by exactly 100 icUSD; and the protocol retains all 100.02 ckUSDT. The fee rate can be adjusted by protocol administrators, serving as a dial: it can be kept low to encourage usage during normal conditions, or raised sharply if a ckstable begins to depeg, effectively acting as a soft kill-switch. Individual stablecoin types can also be disabled entirely.

Critically, when users repay with ckstables, no icUSD is burned. The vault debt is reduced, but the corresponding icUSD remains in circulation. The collected ckstables are held in reserve by the protocol. This reserve serves a dual purpose:

- **Peg defense.** If icUSD begins trading below \$1, the protocol can deploy ckstable reserves to meet redemptions directly, providing dollar-pegged value to redeemers without forcing ICP collateral sales during market volatility. This is particularly valuable during broad crypto downturns when selling ICP collateral for redemptions would add further downward price pressure.
- **Adjusted collateral ratio.** Because stablecoin repayments reduce vault debt without reducing collateral, the system-wide collateral ratio formula is adjusted: the denominator subtracts icUSD debt that has been effectively retired through stablecoin repayment, making the CR reflect only the debt that lacks corresponding stablecoin backing.

The stablecoin reserve balance is tracked on-chain and intended to be surfaced in the protocol's public dashboard for full transparency.

## Protocol Modes

Mode	Trigger	Behavior
General Availability	System CR $\geq$ 150%	Normal operations. 133% MCR. Standard borrowing fee.
Recovery	System CR < 150%	150% MCR. Zero borrowing fee. Partial liquidations enabled.



Read-Only	System CR < 100%, or ICP price < \$0.01	All state-changing operations paused.
-----------	--	---------------------------------------

*Table 3: Protocol operating modes*

## V. Security

---

### Smart Contract Security

The protocol backend is written in Rust, a memory-safe language that eliminates entire classes of vulnerabilities (buffer overflow, use-after-free, data races) at compile time.

All vault operations are protected by per-principal guards: only one state-changing operation per principal can be in flight at any time, eliminating reentrancy. Guard timestamps are tracked, and stale guards (older than 3 minutes) are automatically cleaned up to prevent permanent lockout.

Rate limiting is applied to sensitive operations. Close-vault operations are limited to 5 per minute per user, 60 per day per user, and 300 per minute globally, with a maximum of 200 concurrent close operations system-wide.

State is persisted in stable memory using ic-stable-structures, ensuring data survives canister upgrades. Invariant checks verify state consistency after each upgrade.

### Oracle Security

The XRC's security derives from ICP's consensus mechanism. Every replica in the subnet independently makes HTTPS outcalls and processes responses; the replicas must reach consensus, meaning a price can only be accepted if a supermajority agrees. Manipulating the feed would require compromising both multiple exchanges and a supermajority of subnet replicas.

The 10-minute staleness window prevents operations against outdated prices. The \$0.01 circuit breaker catches oracle malfunction or extreme market events. Together these ensure the protocol fails safe rather than operating on bad data.

### Economic Security

Overcollateralization ensures the value of locked ICP exceeds minted icUSD by at least 33% (normal mode) or 50% (Recovery Mode). The stability pool absorbs liquidations without market-selling collateral, reducing downward price pressure during stress. The stablecoin reserve provides a further buffer, allowing the protocol to meet redemptions with dollar-pegged assets rather than forcing ICP sales during broad market declines. And the redemption mechanism creates a hard arbitrage floor, ensuring icUSD cannot sustainably trade below peg.

## VI. Governance and Decentralization

---

## Current Model

Rumi Protocol currently operates under developer-controlled governance during its bootstrap phase. Canister controllers include the core development team, enabling rapid iteration and parameter adjustments. Key parameters (MCR, recovery ratio, borrowing fee, liquidation bonus) are currently compile-time constants. The roadmap includes migrating these to mutable state fields adjustable through governance.

## Path to SNS

The Internet Computer provides the Service Nervous System (SNS), a native framework for decentralizing canister control to a token-governed DAO. Rumi Protocol plans to transition to SNS governance as the protocol matures. Under SNS, upgrades and parameter changes would require proposal submission, deliberation, and token-weighted voting.

## Immutable Vault Terms

The transition to decentralized governance introduces a risk: governance could change parameters after users have opened positions, retroactively worsening their terms. Rumi Protocol has designed an Immutable Vault Terms system to address this.

When a vault is opened, a snapshot of the current protocol parameters is stored with the vault. These terms act as a user-favorable floor or ceiling: the protocol applies whichever value (global or vault-specific) is more beneficial to the vault owner. The captured terms include the MCR, liquidation penalty, borrowing fee, and a borrowing fee cap.

If governance raises the MCR from 133% to 150%, existing vaults retain their original 133% threshold. If governance lowers the MCR to 125%, all vaults benefit from the lower requirement. The mechanism is backward-compatible: legacy vaults created before the feature have no snapshot and follow global parameters.

# VII. Roadmap and Future Directions

---

- **Multi-collateral expansion.** The architecture accommodates chain-key tokens as additional collateral types, including ckBTC, ckETH, and others. Each would have independently calibrated risk parameters. Chain-key tokens are not bridged assets; they use threshold signatures across subnet nodes to hold underlying assets on their native chains, avoiding bridge risk.
- **icUSD Savings Rate.** A deposit facility where icUSD holders earn yield funded by protocol fees. This would serve as a monetary policy tool: raising the rate incentivizes holding icUSD (tightening supply), lowering it encourages spending. Implementation requires careful calibration to ensure sustainability.
- **Protocol bonds.** A mechanism for time-locked bonds denominated in icUSD, offering a premium at maturity. Bonds could serve as a liquidity management tool and long-term governance incentive. Viability depends on sustained protocol revenue and would follow operational maturity.

- **SNS governance token.** Token design, distribution, and voting power mechanics are under research, with a focus on productive participation incentives.
- **Ecosystem integrations.** As ICP's DeFi ecosystem grows, Rumi Protocol aims to integrate with DEXes, lending platforms, and other protocols to expand icUSD utility.

## VIII. Conclusion

---

Rumi Protocol provides the Internet Computer ecosystem with a fundamental DeFi primitive: a decentralized, overcollateralized stablecoin. The protocol's design prioritizes soundness over complexity. Full overcollateralization, a stability pool for immediate liquidation, a redemption mechanism for hard peg defense, and a stablecoin reserve for volatility periods together form a layered system for maintaining icUSD's dollar peg.

The Immutable Vault Terms design addresses an underexplored problem in DeFi: protecting users' positions as a protocol transitions from developer control to decentralized governance. As Rumi Protocol moves toward SNS control, vault holders can be confident that their terms will not be changed to their detriment.

Rumi Protocol is open source, deployed on the Internet Computer mainnet, and actively developed. The codebase is available at [github.com/RumiLabsXYZ/rumi-protocol-v2](https://github.com/RumiLabsXYZ/rumi-protocol-v2).

## IX. Glossary

---

**Canister** — A smart contract on the Internet Computer that bundles code and state and can serve web content.

**CDP** — Collateralized Debt Position. A loan secured by cryptocurrency collateral exceeding the loan value.

**Chain-Key Token** — A trustless representation of an asset from another blockchain (e.g., ckBTC for Bitcoin) using ICP's threshold signature scheme. Not a bridge.

**ckUSDT / ckUSDC** — Chain-key representations of Tether (USDT) and USD Coin (USDC) on the Internet Computer.

**Collateral Ratio** — The ratio of a vault's collateral value to its debt, expressed as a percentage.

**HTTPS Outcall** — ICP's mechanism for canisters to make direct HTTP requests to external web services, processed through consensus.

**icUSD** — The stablecoin minted by Rumi Protocol, pegged to the US dollar, implemented as an ICRC-1/ICRC-2 token.

**ICRC-1 / ICRC-2** — Token standards on ICP. ICRC-1 defines fungible token operations; ICRC-2 adds approve-and-transfer-from.

**Internet Identity** — ICP's passkey-based authentication framework.

**Liquidation** — Closing an undercollateralized vault to protect protocol solvency.

**MCR** — Minimum Collateral Ratio. The lowest CR a vault may have before becoming eligible for liquidation.

**Principal** — A unique identifier for a user or canister on the Internet Computer.

**Recovery Mode** — Protocol state triggered when system-wide CR falls below 150%.

**Redemption** — Direct exchange of icUSD for ICP collateral at face value, targeting the lowest-ratio vaults first.

**SNS** — Service Nervous System. ICP's framework for decentralizing canister governance.

**Stability Pool** — A reserve of deposited icUSD used to absorb liquidated vault debt in exchange for discounted collateral.

**Vault** — A record representing a user's ICP collateral deposit and icUSD debt.

**XRC** — Exchange Rate Canister. ICP's system-level on-chain oracle for exchange rates.