Solayer Bridge: Canonical Bridge Between SVMs

Solayer Team

Abstract

The Solayer Bridge (sBridge) is a canonical, SVM-native cross-chain bridge designed to securely transfer assets and execute cross-chain transactions between SVM environments, starting from Solana and Solayer. Unlike generic multichain bridges, sBridge aligns tightly with SVM execution semantics, combining on-chain programs with a decentralized guardian quorum coordinated through a leader–follower model and threshold signature aggregation. Key innovations include poll-based transaction indexing, PDA-based bridge proofs, dynamic finality (fast-path and slow-path), and cross-chain call execution. Its modular architecture supports wrapped assets, rotating validators, and adaptive risk controls, providing a fast, secure, and extensible foundation for interchain interoperability.

1 Bidirectional Bridge Between Solayer and Solana

The bidirectional bridge between any SVM environment is a generalized cross-chain communication and asset interoperability layer designed to enable secure asset transfers and preauthorized transaction execution across the heterogeneous SVM execution environments, starting with Solana and Solayer. Unlike generic bridges such as Wormhole [Fou22], LayerZero [PZB23], or Hyperlane [Pro23], which prioritize multichain generality, the Solayer Bridge (sBridge) is designed to maximize capital efficiency and interchain operability by tightly coupling with SVM execution semantics.

We chose to build this bridge in-house because existing solutions lack the flexibility, latency guarantees, and protocol-level control needed for our domain. By integrating on-chain duplicate prevention, poll-based transaction subscription, guardian auto-failover mechanism, and a dynamic-finality model (fast optimistic + slow finalized), we ensure low-latency and highly available bridging without compromising security. The cross-chain call feature unlocks the new ear of interchain operability, as now you can trade assets on two chains by bundling the trade in one transaction on one chain. The design is also extensible for future upgrades, such as dynamic validation sets, new message types, and programmable token minting, and draws inspiration from research on cross-chain security, modular interoperability [BVGC21], and production bridge architectures.

2 sBridge Architecture

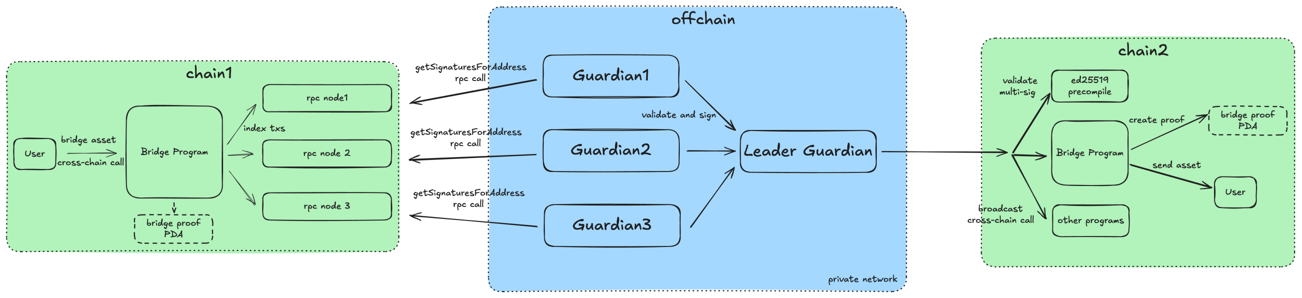

Figure 1: Solayer Bridge Architecture

The bi-directional bridge consists of three core layers: chain1 (origin chain), offchain (guardian multi-sig network), and chain2 (destination chain). A user initiates a bridge operation—either asset transfer or cross-chain execution—on chain1 by interacting with the Bridge Program, which generates a deterministic PDA as a cryptographic proof of intent. This transaction is indexed by multiple RPC providers and consumed by a decentralized guardian network.

Each guardian independently fetches transactions from rpc, verifies them and signs a canonical hash (covering chain, sender, recipient, mint, amount, nonce, and source tx ID). Once the quorum threshold is reached, the leader guardian aggregates signatures and relays them to chain2. There, the Bridge Program uses Solana's ed25519 precompile to verify the multi-sig, enforces replay protection via the bridge proof PDA, and then dispatches the token. For cross-chain call, leader guardian would simply decode and broadcast the transaction on chain2.

3 sBridge On-chain Program

The sBridge on-chain program is deployed on both chains and is responsible for securely executing asset bridging and arbitrary cross-chain call based on validated multi-sig proofs produced by a decentralized guardian quorum. Through bridge proof PDA, the program ensures double-handling prevention, replay protection, and verifiability on-chain. The program also implements an auto-incremental nonce mechanism to increase uniqueness for each user transaction. The nonce can also be used as a bridge handling cutoff point, for example, a bridge transaction with a nonce lower than N could be safely and permanently discarded by the system when the last processed transaction is over 5N, such that the rent paid to related on-chain PDAs could be recycled.

3.1 Asset Bridge

The asset bridge module is responsible for handling canonical and wrapped token transfers between Solana and Solayer. It exposes two core entry points:

- `bridgeAssetSourceChain`: This instruction is invoked on the origin chain (e.g., Solana). It deducts the bridged token amount from the user, splitting it into a fee (transferred to the handler fee vault) and a locked portion (stored in the bridge vault). A nonce is auto-incremented and a PDA is derived from the tuple `(sender, recipient, mint, amount, nonce, txId)` and stored as a cryptographic proof of the request.
- `bridgeAssetTargetChain`: This instruction is executed on the destination chain (e.g., Solayer) after guardians validate the original transaction and submit their aggregated signatures. The program verifies the multi-sig using Solana's precompiled `ed25519_program`, checks the bridge proof PDA for existence on destination chain (to prevent double-processing), and then transfers the bridged asset to the recipient. If the asset is not natively available on the destination chain, a wrapped token is minted under the authority of brdige handler.

The bridge program enforces threshold signature validation, configurable guardian sets, and precise accounting to guarantee safety and liveness across the bridge.

3.2 Cross-Chain Call

The cross-chain call module enables authenticated, cross-chain execution of pre-authorized instructions on the destination chain. This mechanism is particularly useful for multi-chain transactions with specific order requirements.

- On the source chain, the user serializes a transaction for the destination chain and encodes it into base58 format. This payload is then submitted as payload during invocation of the crossChainCall method on thethod on the source chain.
- The Off-chain leader guardian then decodes the base58 transaction, simulates it, and then directly dispatches it to the destination chain. After successful execution, the leader guardian invokes

the bridge program to create a bridge proof PDA derived from the tuple `(sender, calldata, nonce, sourceTxId)` to mark completion.

4 sBridge Off-chain Guardian

The sBridge off-chain guardian network functions as the verification layer for all cross-chain bridge activities. Guardians independently fetch source chain transactions from different sources, validate and produce cryptographic attestations, and coordinate to reach a threshold multi-signature that authorizes execution on the destination chain. This layer is designed for decentralization, fault tolerance, and deterministic correctness, ensuring trust-minimized interoperability.

4.1 Leader and Follower Network Model

The sBridge guardian network adopts a **Leader–Follower network model** to efficiently coordinate signature aggregation and relaying under partial synchrony.

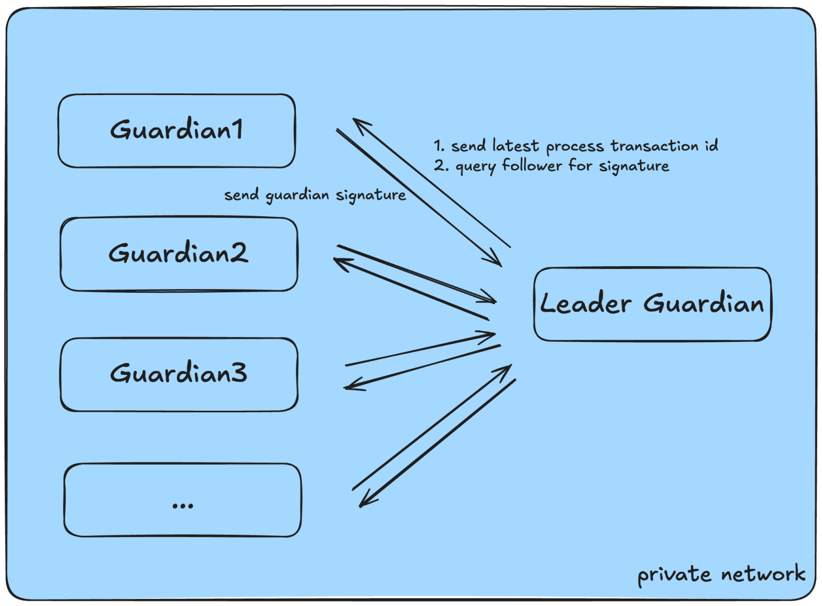

Figure 2: Solayer Bridge Guardian Network

All guardians gossip with each other through a private channel, configured through internal DERP (Designated Encrypted Relay for Packets) server. Each guardian operates independently to fetch, validate, and attest transactions from the source chain. Upon successful validation, guardians generate and locally sign a canonical hash of the transaction payload. This cryptographic attestation represents the guardian's commitment to the transaction's correctness.

To streamline multi-signature coordination, a single leader guardian is deterministically selected at the beginning and rotates after processing **N** transactions, based on a round-robin rotation. The leader is responsible for collecting individual guardian signatures until a pre-configured threshold (e.g., t of n) is met. If the leader waits for follower signatures over a certain time threshold, it would reach out to the followers and ask for signatures. Once quorum is reached, the leader constructs the aggregated multi-signature payload and submit to the destination chain.

Follower Guardians do not coordinate directly with each other but continuously gossip their signed attestations to the leader and optional fallback peers. This gossip model enables redundancy in the face of leader failure and supports rapid recovery.

After successfully aggregating and relaying a batch of transactions, the leader guardian broadcasts a commit acknowledgment to all guardians, instructing them to persist the latest processed transaction signature as their local high-water mark. This ensures consistent fault recovery checkpoints across the network and enables seamless leader rotation, allowing any follower to deterministically resume leadership without reprocessing already-committed transactions.

4.2 Transaction Subscription

Each guardian operates against a dedicated RPC endpoint and is responsible for continuously polling new transactions of the bridge program on the source chain.

For both **Solana** and **Solayer** chain, each guardian periodically invokes the `getSignaturesForAddress` RPC method on the bridge program ID, starting from the most recently processed transaction signature. This position is stored locally in a durable file. Upon retrieval, signatures are fetched in reverse chronological order and then resolved to full transactions using `getTransaction`. This polling mechanism also works well during guardian crash or new guardian catch up cases as long as the local file is stored properly.

To ensure exactly-once semantics, guardians maintain a local de-duplication set in memory and persistently store the highest processed transaction signature. This eliminates reprocessing due to retries, even in cases of partial crashes or concurrent recovery.

4.3 Multi-Sig Aggregation

After validating a transaction, each guardian generates a deterministic hash over the bridge payload:

$$BridgeAsset : H = \texttt{hash(sender, recipient, mint, amount, nonce, sourceTxId)}$$

$$CrossChainCall : H = hash(sender, calldata, nonce, sourceTxId)$$

Each guardian signs this hash by invoking hardware security module (HSM), which never exposes the private key in the memory. The designated leader guardian is responsible for collecting signatures until the threshold t out of n is reached (e.g., $t = 5, n = 7$). Once quorum is achieved, the leader packages the multi-sig data into a canonical payload and then submit it to the destination chain.

The multi-sig scheme supports both static validator sets and dynamic reconfiguration, with public keys stored on-chain for signature verification using Solana's `ed25519_program`.

4.4 Failure Recovery

Failure Retry

Bridge execution failures (e.g., transient networking failures) are handled via a fault-tolerant redrive mechanism. Lead guardian continuously scans the cached transaction sessions and process each session based on it's current status. Each retry operation verifies that the transaction has not already been executed on-chain by checking the existence of its PDA proof. This ensures at-least-once delivery semantics without compromising determinism or duplicity protection from the off-chain module perspective.

Guardian Leader Failover

Leader failure (e.g., network partition, crash) is mitigated through a rotating leader election strategy. Guardians periodically exchange heartbeats, and if the current leader becomes unresponsive beyond a timeout window, a new leader is elected deterministically (e.g., round-robin by guardian ID).

Any follower should be able to quickly rotate as the new leader as they should have up-to-date fault recovery checkpoint locally stored. During transaction processing, leader leader would broadcast highest processed transaction signature to followers after finishing a batch. Even if the checkpoint transaction is slightly delayed from the actual one, the new leader can still quickly catch up as it would check bridge proof PDA on destination chain for each transaction and quickly identifies already committed ones.

5 Double-Handling Prevention

To ensure safe and deterministic cross-chain execution, the sBridge protocol enforces strict double-handling prevention guarantees across both on-chain and off-chain components. These properties are critical for maintaining correctness in the presence of retries, partial failures, and network reordering—common in asynchronous, adversarial environments.

On-chain Prevention

Each bridge operation is uniquely identified by a deterministic `Program Derived Address` (PDA) derived from a tuple of immutable parameters, as discussed in section 4.3.

Before executing any bridge instruction, the target chain's Bridge Program performs a `PDA existence check`. If the PDA already exists, the instruction is aborted, thereby preventing double-execution of the same logical transaction.

Off-chain De-duplication

Off-chain, each guardian maintains a persistent log of all observed and signed source transactions, indexed by their `sourceTxId`. Before processing a new transaction, a guardian checks this log to determine whether it has already produced a signature or witnessed the transaction being finalized. This prevents unnecessary signature generation and ensures consistency across leader rotation or recovery.

In addition, each guardian tracks the **highest processed nonce** per bridge direction (Solana $\rightarrow$ Solayer and Solayer $\rightarrow$ Solana). Any incoming transaction with a nonce lower than the stored watermark is rejected as stale or previously handled. This mechanism also serves as a compact garbage collection strategy when purging old transaction logs or reclaiming PDA rent.

Recovery Safety

During recovery (e.g., after crash or restart), guardians restore their high-water mark from durable storage and resume transaction polling from the last acknowledged signature. Since all critical execution paths are gated by PDA existence (on-chain) and signature logs (off-chain), the system tolerates retries and message duplication without compromising safety or introducing non-deterministic behavior.

These mechanisms align with formal principles of *exactly-once semantics* in distributed systems and are inspired by transaction replay resistance models used in production systems like Cosmos IBC and LayerZero Ultra Light Nodes.

6 Dynamic Finality

The sBridge protocol adopts a **Dynamic Finality** model to optimize the trade-off between bridging latency and economic security. Instead of relying solely on chain-level consensus finality (e.g., slot confirmation on Solana or block finalization on Solayer), the system introduces a dual-mode confirmation mechanism: *fast-path optimistic execution* and *slow-path finality-enforced execution*, determined by the value and context of the transaction.

Fast-Path Finality (Optimistic Confirmation)

For low-value transactions, below a predefined risk threshold estimated from dollar value, guardians are permitted to sign and relay a transaction after verifying its presence in the source chain's recent confirmed blocks (e.g., ‘confirmationStatus = confirmed‘ in Solana RPC). This optimistic path assumes probabilistic finality and economic boundedness: if a reorg occurs, the potential loss is limited and will be covered by the bridge fee.

Slow-Path Finality (Deterministic Confirmation)

For high-value transactions—above the configured threshold—the guardians defer signing until the transaction reaches irreversible finality on the source chain. This ensures that high-risk operations cannot be reverted or front-run, aligning with formal definitions of deterministic safety.

Guardian Behavior Enforcement

Each guardian enforces finality rules locally and independently. If a guardian signs a fast-path transaction that later fails finalization (e.g., due to a reorg), the system may mark it as misbehavior. Depending on the deployment, this can trigger reputation loss, exclusion from quorum, or economic penalties through bonded collateral.

To prevent inconsistent views, all guardians must track and persist the finality status of each transaction they process, and the leader guardian must enforce uniform behavior during multi-sig aggregation.

References

[BVGC21] Rafael Belchior, André Vasconcelos, Sérgio Guerreiro, and Miguel Correia. A survey on blockchain interoperability: Past, present, and future trends. *Acm Computing Surveys (CSUR)*, 54(8):1–41, 2021.

[Fou22] Wormhole Foundation. Wormhole documentation: Core protocol overview, 2022. Accessed: 2025-07-01.

[Pro23] Hyperlane Protocol. Hyperlane protocol documentation: Validator and agent architecture, 2023. Accessed: 2025-07-01.

[PZB23] Bryan Pellegrino, Ryan Zarick, and Caleb Banister. Layerzero v2: Generalized interoperability, 2023. Accessed: 2025-07-01.