MORSOL Repository Overview

Overview of Functionality

MORSOL (Morpheus Solana contracts) is designed to bring the Morpheus **MOR** token into the Solana ecosystem as an **omnichain fungible token**. It implements LayerZero's Omnichain Fungible Token (OFT) standard on Solana, allowing MOR tokens to move between Solana and other chains without using wrapped assets . In practice, tokens are **burned on the source chain and minted on the destination** via LayerZero messages, maintaining a unified total supply across networks . MORSOL interacts with the Solana blockchain through an on-chain Anchor program (smart contract) that handles these burn/mint operations and communicates with LayerZero's cross-chain infrastructure. On the other side, it interfaces with LayerZero endpoints on EVM chains (like Ethereum, Arbitrum, Base, etc.), leveraging standard LayerZero **OFT contracts** (which are unchanged from LayerZero's official implementations) to send and receive MOR token transfers. This design makes MOR a **native multichain token**—deployed on multiple blockchains yet kept in sync. According to the future plans, the project will use **LayerZero OFT** alongside **Wormhole's NTT** bridging standards to achieve seamless multichain deployment of the MOR token . In summary, MORSOL's core functionality is enabling **omnichain token transfers** for MOR, by **burning, messaging, and minting** tokens across Solana and EVM chains, all while preserving the token's total supply and fungibility across all networks.

High-Level Architecture

MORSOL's architecture consists of both Solana and EVM components, orchestrated by a set of scripts and configurations. On the Solana side, the main component is an Anchor-based program (located under programs/moroft/) which implements the Solana OFT logic. This program defines instructions to send/receive tokens across chains (burning or minting MOR on Solana when cross-chain transfers occur) and stores state (e.g. an OFT store account tracking the token settings). Notably, MORSOL uses LayerZero's official contracts and libraries without modification – the LayerZero endpoint and messaging protocol on Solana and the standard OFTCore.sol contract on EVM are used as-is , ensuring it adheres to audited LayerZero standards. On the EVM side, the repository includes a contracts/ directory with the LayerZero OFT Solidity contract custom implementation that represents MOR on chains like Ethereum or Arbitrum. This EVM contract is an extension of a normal ERC-20 token that can burn/mint tokens in response to LayerZero messages .

Surrounding these core contracts, the repository's **structure** includes configuration and deployment scripts (in deploy/ and tasks/ directories) which use Hardhat and Anchor CLI to manage cross-chain setup. For example, a **LayerZero config file** (layerzero.config.ts) defines chain IDs and contract addresses for each network, linking the Solana program with its EVM "peer" contracts. Hardhat tasks (scripts in tasks/) automate the process of deploying the Solana program, creating the SPL token mint, deploying the EVM contract, and then **wiring them together** by registering the chain endpoints and trusted remote addresses. The utils/ folder provides helper scripts (e.g. for key conversion or rent calculations).

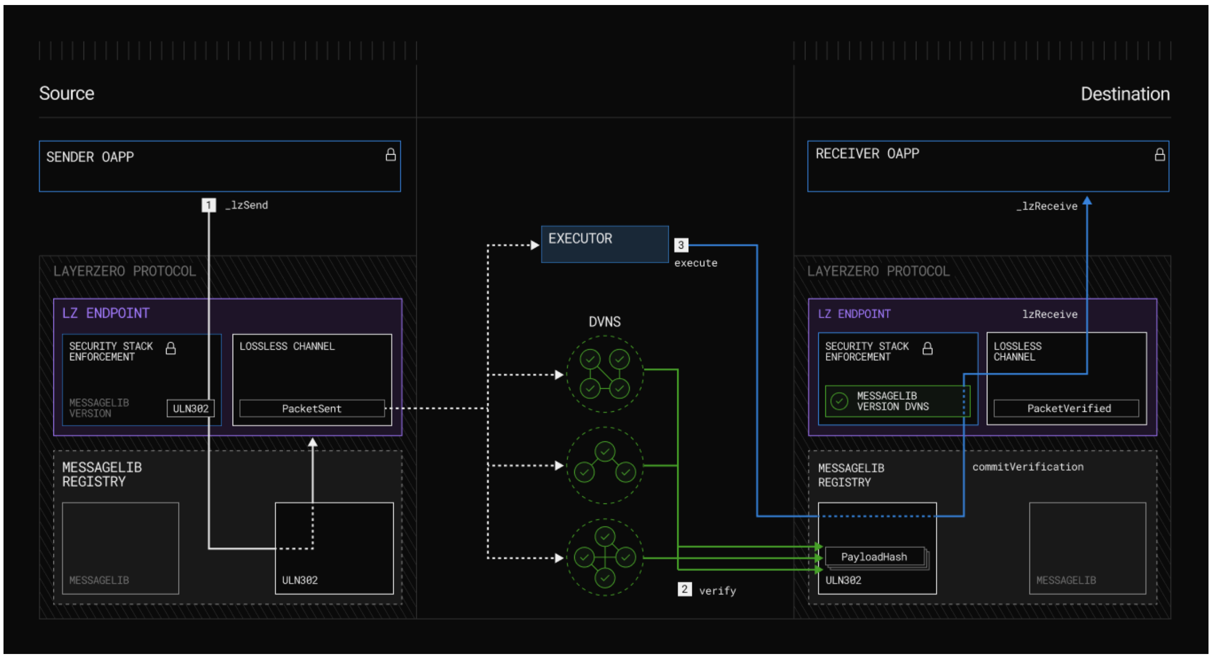


Figure: LayerZero OFT architecture.

The **overall flow** is that the Solana program and the EVM contract act as peers: when a user on Solana sends MOR to an EVM chain, the Solana program burns the tokens and sends a LayerZero message; the LayerZero endpoint delivers it to the EVM contract, which then mints the equivalent tokens (and vice-versa for EVM to Solana). All LayerZero **security mechanisms** (validation of proof/Oracle data and Relayer delivery) remain in effect, since MORSOL does not alter the LayerZero contracts. In essence, the repository sets up a **bi-directional token bridge**: Solana's **MOR-OFT program** on one side, and the custom LayerZero OFT contract on the other, configured to trust each other. This achieves **omnichain MOR transfers** while relying on LayerZero's established protocol for message passing and finality.

Multisig Infrastructure (Solana Side Architecture)

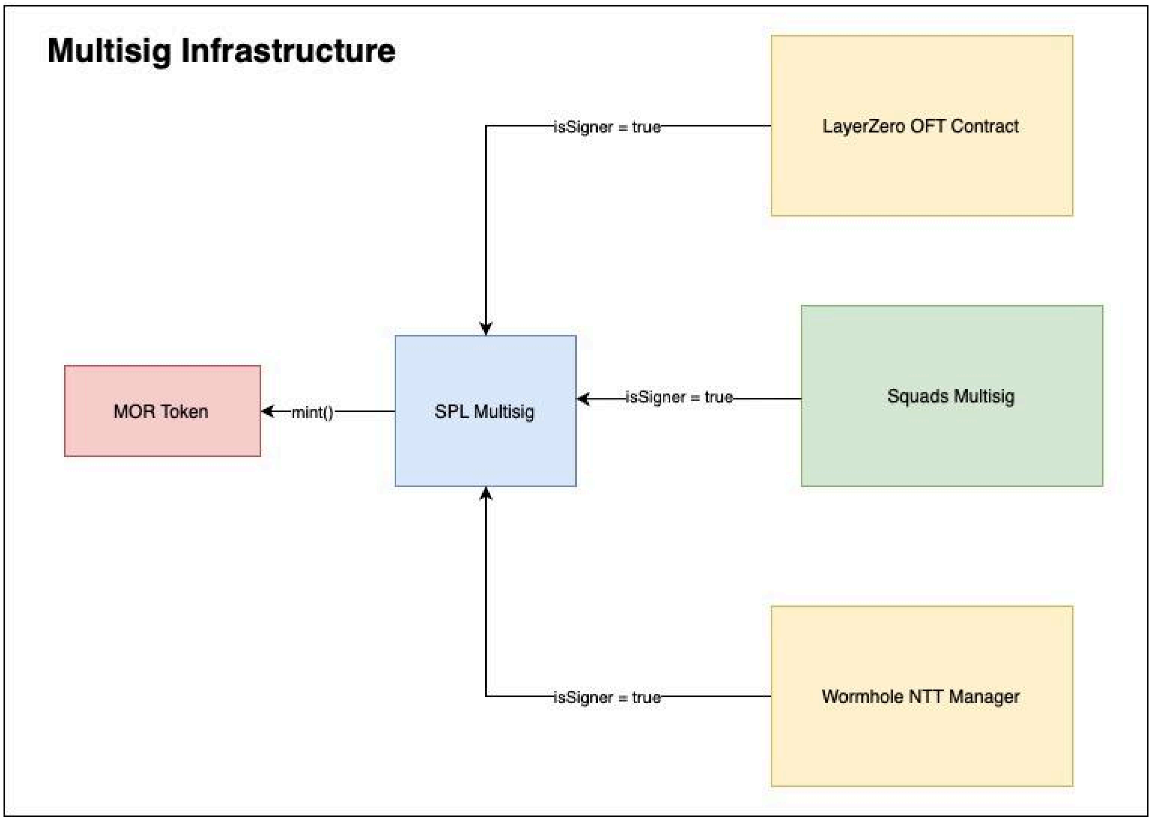


Figure: High-level architecture diagram of MOR omnichain token on Solana. A Squads multisig controls the MOR SPL token mint on Solana, acting as a gatekeeper for minting. The Solana OFT program (LayerZero integration) and a Wormhole NTT manager (will be implemented in the future) are both configured as authorized minters, requiring multisig approval for cross-chain token flows.

To enhance security, the **MOR token mint on Solana is governed by a multisig structure** rather than a single key. In Solana's SPL token standard, a mint's authority can itself be a special account that requires multiple signatures (1-of-N) for any mint or freeze action . MORSOL leverages this by setting the token's mint authority to a **SPL multisig** account which is a Solana-native multisig program that allows a group of keys to collectively control assets and sign transactions through proposals. In this design, the Squads multisig (represented by its **vault PDA** address) is the external controller of MOR token issuance on Solana along with NTT and OFT. This means no single entity can mint new MOR tokens on Solana without the required approvals, adding a layer of decentralization and safety, Despite of authorized contracts like NTT and OFT that are added to the SPL Multisig.

The multisig's role is central in **controlling token minting and bridging interactions**. When the Solana OFT program needs to mint MOR in response to an incoming LayerZero message (e.g. tokens arriving from Ethereum), it cannot freely mint on its own — it must be authorized by the multisig. During the deployment process, the MORSOL setup script creates the MOR token with the SPL multisig as the mint authority, but also configures the OFT program's own derived account and Squads multisig VaultPDA as an **"additional minters"**. In practice, this means the OFT program is allowed to mint or burn tokens *only* in collaboration with the multisig. The Squads multisig would have to pre-approve or co-sign the mint instruction (for example, the program might invoke the token mint instruction and one of the required signatures is provided by the Squads account). This ensures that **omnichain transfers are under multisig oversight** – any cross-chain mint operation on Solana requires the multisig's consent, aligning with Morpheus's goal of community-governed contracts . The Squads multisig may include Morpheus team members or community members as signers, and is set with a threshold (e.g. 5 of 9 signatures) as described in Morpheus's governance documents .

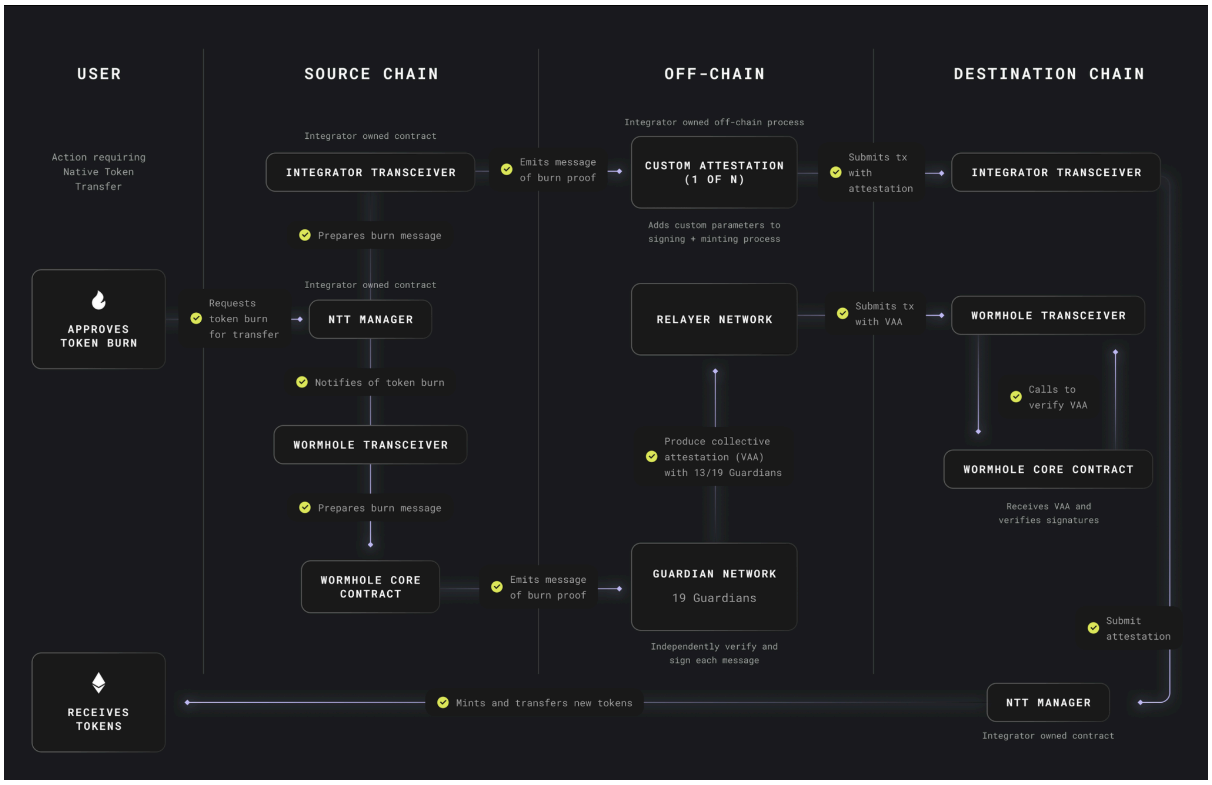
By using Squads, the project benefits from a tested on-chain wallet for managing the token. Squads introduces advanced features like time-locks and role-based signers, which could be used to automate certain actions within set limits. In the current architecture. It can also be the authority to configure or upgrade the Solana OFT program if needed (ensuring any program updates are multisig-approved). Additionally, the SPL multisig could interact with the LayerZero or Wormhole configurations – for example, approving registration of new "trusted remotes" (new chain contracts) or pausing token transfers in an emergency. Overall, the multisig acts as a **security gatekeeper** for token control on Solana: it holds the keys to minting and thus can enforce that cross-chain mint/burn operations match the intended logic. This design addresses risk scenarios like bridge exploits by requiring human (or governance) intervention for unexpected mint events. *In summary, the Squads multisig controls the MOR token mint on Solana and coordinates with both the LayerZero OFT program and (proposed) Wormhole integration, ensuring that no single contract or party can unilaterally mint tokens.* Squads provides an easy interface and robust security for this purpose .

Proposal: Wormhole Integration Through Multisig (*can be redesigned after Wormhole Multisig support audit*)

In addition to LayerZero, Morpheus plans to integrate **Wormhole's Native Token Transfer (NTT) framework** for bridging MOR, using the same multisig architecture as a safeguard. Wormhole NTT is Wormhole's approach to make a token **natively**

multichain – tokens aren’t “wrapped” but rather minted and burned on each chain, similar to the OFT model . To connect Solana via Wormhole, we propose deploying a Wormhole **NTT Manager** contract/program for MOR that interfaces with Wormhole’s core (guardian) network. On Solana, this would likely involve either using the existing Wormhole Token Bridge program in a special mode or deploying a custom Wormhole integration program that can **verify guardian VAAs** (signed messages) and instruct the MOR mint. The key idea is to align this with the existing multisig control:

1. **NTT Manager as a Multisig Signer:**



USER
SOURCE CHAIN
OFF-CHAIN
DESTINATION CHAIN
Action requiring Native Token Transfer
APPROVES TOKEN BURN
Requests token burn for transfer
Integrator owned contract
INTEGRATOR TRANSCEIVER
Emits message of burn proof
Prepares burn message
Integrator owned contract
NTT MANAGER
Notifies of token burn
WORMHOLE TRANSCEIVER
Prepares burn message
WORMHOLE CORE CONTRACT
Emits message of burn proof
Integrator owned off-chain process
CUSTOM ATTESTATION (1 OF N)
Adds custom parameters to signing + minting process
RELAYER NETWORK
Produce collective attestation (VAA) with 13/19 Guardians
GUARDIAN NETWORK
19 Guardians
Independently verify and sign each message
Submits tx with attestation
INTEGRATOR TRANSCEIVER
Submits tx with VAA
WORMHOLE TRANSCEIVER
Calls to verify VAA
WORMHOLE CORE CONTRACT
Receives VAA and verifies signatures
Submit attestation
NTT MANAGER
Integrator owned contract
Mints and transfers new tokens
RECEIVES TOKENS

Figure: Wormhole NTT architecture.

We configure the Wormhole NTT Manager such that it cannot mint MOR on Solana outright; instead, it acts as one signer in the Squads multisig. Practically, this could mean the multisig adds a designated key (or PDA) controlled by the NTT Manager as one of its members. When a Wormhole transfer from another chain is verified (guardian VAA confirmed), the NTT Manager will cosign a mint instruction for the appropriate amount of MOR to the user's address. The multisig's policy can require, say, **1-out-of-3 signatures** (for example: one signature from the NTT Manager upon VAA) to execute the mint. This way, Wormhole's message **must be approved by the multisig** consensus before new tokens are minted, preventing any single-point failure. Wormhole's NTT framework supports custom attestation and additional verification layers , so Morpheus can introduce their own signer(s) (the multisig keys) as an extra requirement on top of guardian consensus. This approach aligns with Wormhole's best practices for NTT, which encourage integrators to retain ownership and set up governance controls over minting .

2. **Cross-Chain Setup:** On each chain where MOR exists, deploy an NTT Manager (or use Wormhole's provided contracts) and have them **register each other as peers** (establish trust between the contracts across chains). The Squads multisig (or

Morpheus governance) would be the **owner** of these NTT contracts, as well as the MOR token contracts, to manage roles and settings . The MOR token on Ethereum and other EVM chains can also be retrofitted with Wormhole NTT by deploying an NTT Manager that has permission to burn/mint MOR (the Ethereum MOR contract would grant that contract a minter/burner role). Because MOR is already live on multiple chains, we'd use the **burn-and-mint mode**of NTT : e.g. burning MOR on Ethereum via the NTT Manager will emit a Wormhole message, which the Solana NTT Manager can verify and then request the multisig to mint the same amount on Solana.

3. **Governance and Safety:** The SPL multisig remains at the center of control. It will hold the **Owner role** of the Solana MOR token mint and perhaps the Pauser role of the NTT Manager . This means the multisig can pause cross-chain transfers if something goes wrong (using Wormhole's pausable feature) and can upgrade the contracts if necessary. The Wormhole **Global Accountant** mechanism will also ensure that the total circulating MOR across all chains never exceeds the intended supply (guardians enforce this rule by refusing invalid attestations). Thus, with Wormhole integration, we get an additional bridging pathway that is highly secure: it relies on Wormhole's 19 guardians plus Morpheus's multisig signers. Even if Wormhole's guardians were compromised, the attacker could not mint new MOR on Solana without the multisig's approval, and vice versa – this "two-factor" model significantly mitigates risk .

Implementation Proposal: Deploy the Wormhole NTT Manager on Solana and other chains, and set the MOR token's authorities such that: (a) the SPL multisig is the owner of the token and has ultimate control; (b) the NTT Manager contract is granted a **restricted minting right along with OFT contract** (via multisig or as an authorized minter) on Solana. On Solana, processing a Wormhole transfer would involve minting the tokens by the SPL multisig to the user's account. All of this can be designed to be **user-triggered** under the hood.

In summary, integrating Wormhole via the multisig involves **adding a Wormhole NTT path to the MOR token**, where the **SPL multisig and OFT contract co-signs all mint/burn events**. This preserves the invariant that **no new MOR can be minted on any chain without multisig oversight**, even while allowing automated cross-chain transfers. This proposal aligns with the current multisig-centric design and leverages Wormhole's robust infrastructure for additional chain support, all while ensuring compliance with Wormhole's recommended best practices for governance, rate-limits, and security checks .

References

1. LayerZero Docs – *Omnichain Fungible Token (OFT) Standard*
2. Morpheus Documentation – *MOR20 Value Proposition (Multichain Token)*
3. Solana SPL Token Program – *Multisig Authority Usage*
4. Wormhole Documentation – *Native Token Transfers Overview*
5. Wormhole Blog – *Deep Dive: Wormhole NTT (Roles and Security)*
6. LayerZero Documentation – *Solana OFT Program (Deployment guidance)*