# ZK-Stables: USDC/USDT Non Custodial Bridge Architectural Planning – Feasibility Report & System Architecture Blueprint

## Feasibility Assessment

### End-to-End Cross-Chain Workflow

The bridge supports **non-custodial, ZK-verified cross-chain transfers** using a lock–prove–mint / burn–prove–unlock model.

### Lock → Proof → Mint / Release

1. User locks assets on the source chain via a smart contract.
2. Lock event is emitted with nonce and metadata.
3. Relayer observes finality and generates a ZK proof.
4. Proof is verified by the destination chain verifier.
5. Equivalent assets are minted or released to the recipient.

### Burn → Proof → Unlock

1. User burns wrapped assets on the destination chain.
2. Burn event reaches finality.
3. Relayer generates ZK proof of burn finality.
4. Source chain verifier validates the proof.
5. Original locked assets are unlocked and transferred.

---

### Contract Interfaces & Integration Surface

## Smart Contracts

- Lock / Unlock contracts (Cardano, EVM)
- Mint / Burn contracts (destination chains)
- Light-client ZK verifiers
- Liquidity pool contracts

## SDK Surface (Illustrative)

```
lock(amount, destChain, recipient)
burn(amount, sourceChain, recipient)
getStatus(txHash)
estimateFee(amount, chains)
subscribeEvents()
```

## Wallet Integration Points

- Transaction signing (no custody)
- Status tracking and notifications
- Fee estimation and confirmation

## Relayer Endpoints

- Event ingestion
- Proof generation requests
- Proof submission and retry logic

---

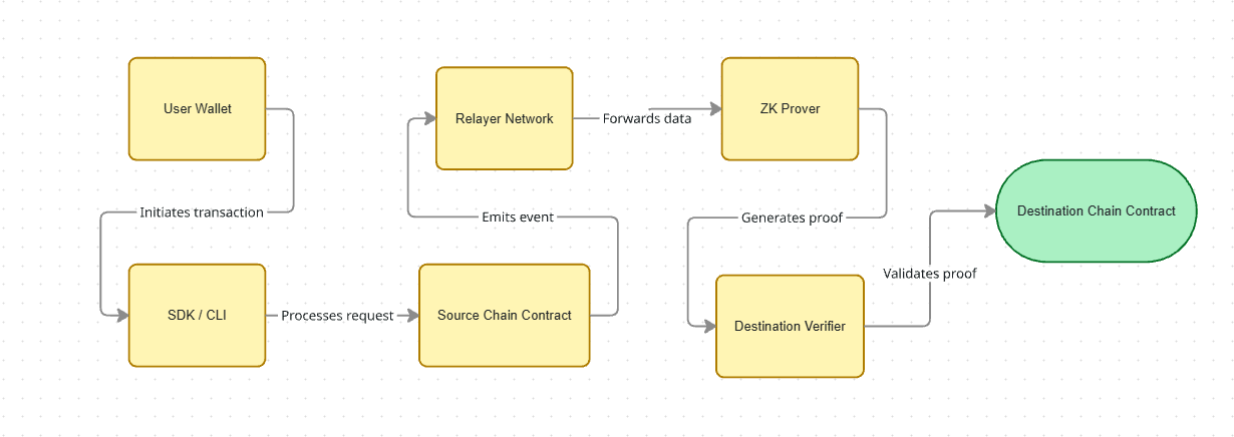
## Risk Evaluation & Mitigations

| Risk | Description | Mitigation |
|---|---|---|
| Chain Reorgs | Source chain reorganizations | Finality thresholds per chain |
| Chain Halts | Network downtime | Fallback relayers, pause logic |
| Proving Latency | ZK proof delays | Parallel proving, batching |

| Risk | Description | Mitigation |
| --- | --- | --- |
| Privacy Leaks | Metadata exposure | ZK selective disclosure |
| Relayer Failure | Single-point outages | Active-active relayer setup |

# 3. System Architecture Blueprint

## 3.1 High-Level Architecture




## 3.2 Component Responsibilities

### Source Chain Smart Contracts

- Lock assets
- Emit verifiable events
- Enforce nonce-based replay protection

### Relayer Network

- Monitor events
- Enforce finality rules
- Generate and submit proofs

### ZK Circuits

- Header finality verification
- Event inclusion proofs
- Optional privacy predicates

### Destination Chain Contracts

- Verify proofs
- Mint / release assets
- Enforce state machine rules

---

## 3.3 ZK Circuit Modules

### Private Inputs

- `validator_signatures`: Aggregated or individual validator signatures
- `validator_pubkeys`: Validator public keys (or aggregated key)
- `event_payload`: Lock or burn event data (amount, asset ID, sender, recipient)
- `nonce_secret`: Secret nonce preimage

### Public Inputs

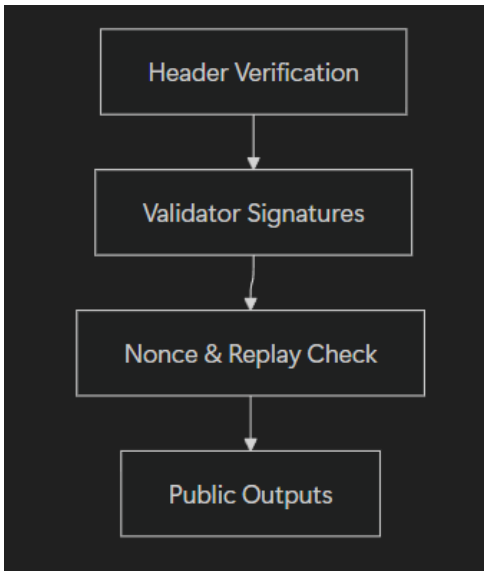
- `header_hash`: Hash of the finalized block header
- `nonce_commitment`: Hash(nonce_secret)
- `destination_chain_id`: Target chain identifier
- `operation_type`: Enum (LOCK_MINT or BURN_UNLOCK)

### Public Outputs

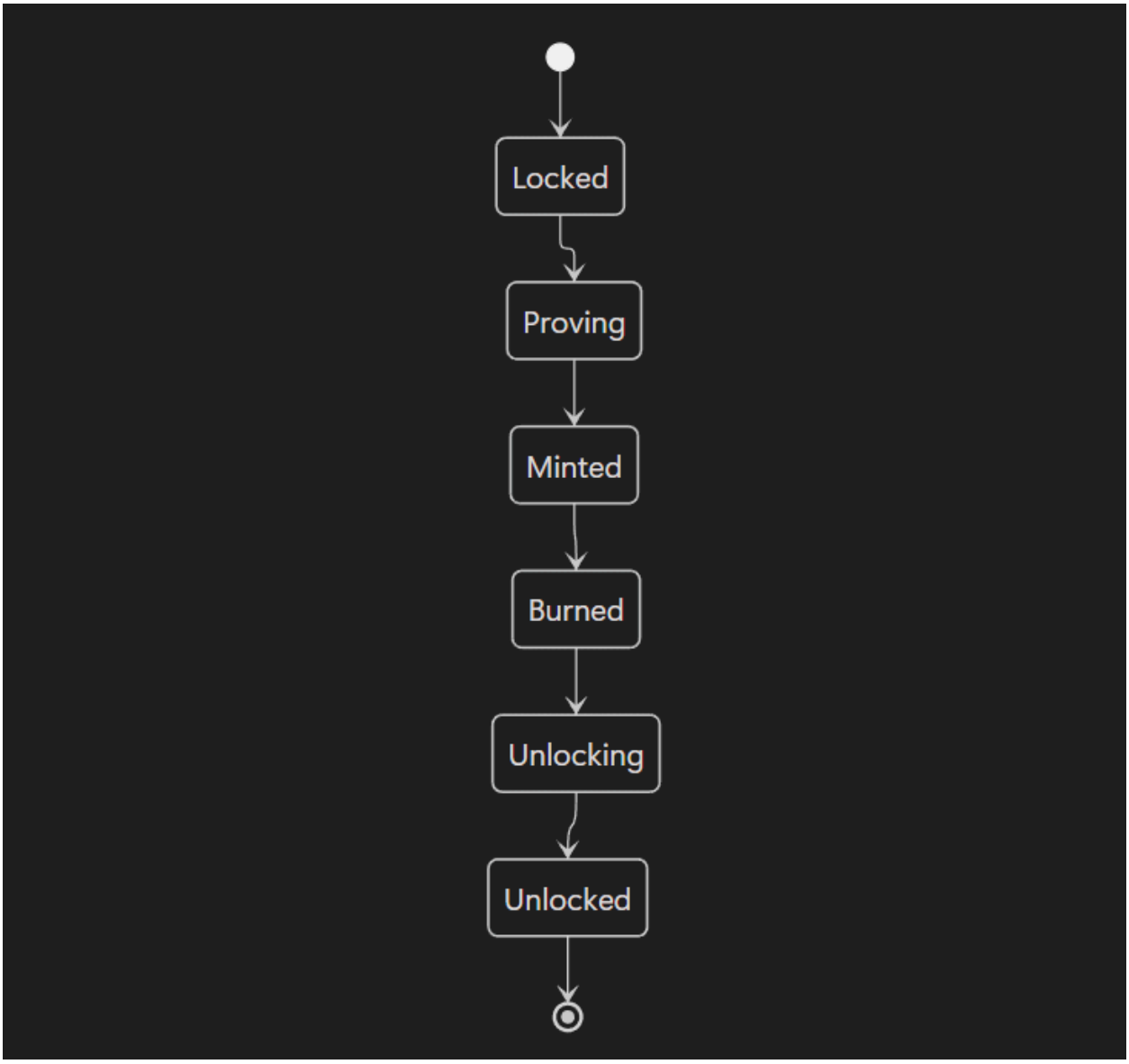
- `is_finalized`: Boolean finality flag
- `is_event_valid`: Boolean inclusion result
- `is_nonce_unused`: Boolean replay-protection result

## Circuit Constraints

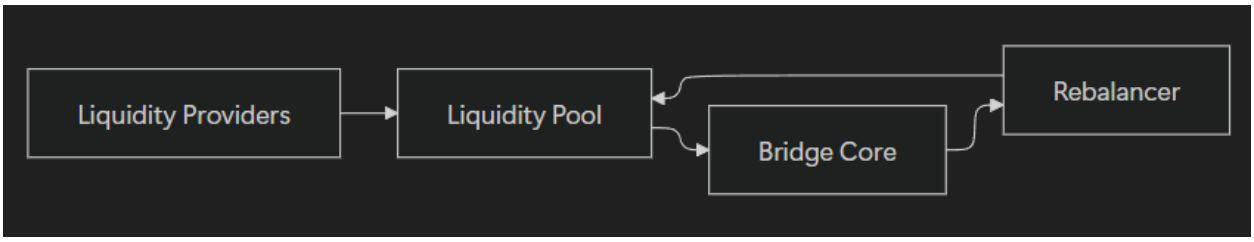
- Header validity constraint
- Nonce uniqueness constraint
- Optional privacy predicates (range proofs, address masking)




## 3.4 State Machine




## 3.5 Liquidity Pool Architecture



# 4. Robustness, Scalability & Fallbacks

## Robustness

- Non-custodial asset handling
- Deterministic state transitions
- Replay protection via nonces

## Scalability

- Stateless SDK services
- Horizontally scalable relayers
- Batched proof verification

## Fallback Strategies

- Pause / resume bridge operations
- Governance-controlled emergency overrides
- Manual proof submission paths
- Validator set recovery procedures