

Software Requirements Specification- ZK-Stables: USDC/USDT Non-Custodial Bridge

1. Introduction

1.1 Purpose

This Software Requirements Specification (SRS) defines the functional and non-functional requirements for the ZK-Stables non-custodial bridge. The document serves as the authoritative reference for development, testing, and acceptance criteria throughout the project lifecycle.

1.2 Scope

ZK-Stables is a non-custodial, zero-knowledge-verified bridge enabling secure transfer of USDC and USDT across EVM chains, Cardano, and Midnight. The system provides:

- **Cross-chain asset transfers** without custodial intermediaries
- **Zero-knowledge proof verification** of finality across chains
- **Developer SDKs and user-facing integrations** for wallets and dApps

1.3 Target Audience

- Development team (blockchain engineers, ZK specialists, smart contract developers)
- Ecosystem partners

2. Overall Description

2.1 Product Overview

ZK-Stables removes reliance on custodial bridges and multisig operators by using zero-knowledge proofs to verify chain finality. Users lock stablecoins on source chains; ZK circuits prove block headers reached finality, triggering minting on destination chains.

2.2 Problem Statement

Cardano currently lacks native, non-custodial USDC/USDT inflow. Existing solutions rely on:

- **Custodial bridges** (third-party risk, opaque operations)
- **Centralized exchanges** (delays, compliance burden, poor user experience)
- **Fragmented liquidity** (reduced capital efficiency, wider spreads)

This limits DeFi adoption, institutional participation, and Cardano's competitiveness.

2.3 Solution Summary

ZK-Stables provides a transparent, verifiable, and modular bridge architecture:

- Light client contracts validate succinct ZK proofs of finalized headers
- Smart contracts mint/release equivalent assets upon proven lock/burn events
- Open-source design enables ecosystem-wide reuse

2.4 Key Features

1. **Non-Custodial Design:** Users retain asset ownership; no trusted intermediaries
2. **ZK-Verified Finality:** Cryptographic proofs replace multisig signers
3. **Multi-Chain Support:** EVM, Cardano, and Midnight interoperability
4. **Developer-Friendly SDKs:** TypeScript/JavaScript libraries for integration
5. **One-Click Bridge Widget:** Minimal friction for end-users
6. **Privacy Preservation:** Selective disclosure and rate limiting via ZK
7. **Open Source:** MIT-licensed, fully transparent codebase

3. Functional Requirements

3.1 Core Bridge Operations

FR-3.1.1 Asset Lock

Requirement: Enable users to lock USDC/USDT on source chains without custodial intermediaries.

Functional Details:

- User initiates lock via dApp or SDK specifying:
- Asset type (USDC or USDT)
- Amount
- Source chain (EVM, Cardano, or Midnight)
- Destination chain
- Recipient address
- Smart contract receives tokens into non-custodial pool contract
- Event logged with asset amount, sender, recipient, and nonce
- Transaction confirmed with receipt provided to user

FR-3.1.2 Finality Proof Generation

Requirement: Generate zero-knowledge proofs demonstrating source chain finality.

Functional Details:

- Relayer monitors source chain for lock events
- Upon event detection, relayer waits for finality threshold
- Relayer fetches finalized block headers and validator signatures
- ZK circuit generates proof:
- Validates header structure and cryptographic signatures
- Verifies block hash and merkle tree inclusion of lock event
- Produces succinct proof
- Proof signed by relayer and submitted to destination chain verifier

FR-3.1.3 Asset Mint

Requirement: Mint equivalent assets on destination chain upon proof verification.

Functional Details:

- Bridge contract receives ZK proof from relayer
- Contract calls light-client verifier to validate proof
- Upon successful verification:
 - Contract checks nonce (prevents replay attacks)
 - Contract mints equivalent tokens to recipient address
 - Event logged with mint transaction hash
 - Recipient notified via SDK or dApp UI
 - Minted tokens are standard ERC-20 (Ethereum-compatible) or native Cardano tokens

FR-3.1.4 Asset Burn

Requirement: Enable users to burn stablecoins on destination chain to unlock assets on source chain.

Functional Details:

- User initiates burn via dApp specifying:
 - Amount to burn
 - Destination source chain
 - Recipient address on source chain
 - Smart contract burns tokens (removes from supply)
 - Event logged with burn amount, sender, recipient, and nonce
 - Transaction confirmed with receipt

FR-3.1.5 Asset Unlock

Requirement: Unlock originally locked assets upon successful burn proof.

Functional Details:

- Relayer detects burn event on destination chain
- Generates ZK proof of burn event finality
- Submits proof to source chain verifier
- Upon verification:
 - Smart contract checks nonce

- Contract releases equivalent locked tokens to recipient
- Event logged
- User notified

3.2 Liquidity Management

FR-3.2.1 Pooled Liquidity Model

Requirement: Maintain non-custodial liquidity pools for USDC and USDT on each supported chain.

Functional Details:

- Each chain maintains separate liquidity pools for USDC and USDT
- Pools are smart contracts holding user-locked assets

4. Non-Functional Requirements

4.1 Performance

NFR-4.1.1 Bridge Latency

Requirement: Bridge transfers complete within acceptable time frames.

Specifications:

- Lock to mint: 30 minutes (EVM), 8 hours (Cardano), 15 minutes (Midnight)
- Burn to unlock: 30 minutes (EVM), 8 hours (Cardano), 15 minutes (Midnight)
- Variability: ±20% due to network congestion (measured weekly)

NFR-4.1.2 Throughput

Requirement: Bridge sustains high transaction throughput.

Specifications:

- Peak throughput: 10 locks/burns per minute
- Proof generation: 1 proofs per minute

NFR-4.1.3 SDK Performance

Requirement: SDK operations respond quickly without blocking.

Specifications:

- Lock initiation: < 1000 ms
- Status query: < 500 ms
- Fee estimation: < 200 ms
- SDK memory footprint: < 10 MB

4.2 Security

NFR-4.2.1 Cryptographic Security

Requirement: All cryptographic operations meet modern security standards.

Specifications:

- Hash functions: SHA-256 or equivalent (256-bit)
- ZK proof system: Sound, complete, and zero-knowledge proofs

NFR-4.2.2 Smart Contract Security

Specifications:

- No critical or high-severity vulnerabilities (CVSS 7+)

NFR-4.2.3 Privacy

Requirement: User privacy is protected to extent feasible.

Specifications:

- Selective disclosure available (sender/recipient masking)
- ZK proofs do not leak transaction amounts (via optional privacy mode)