HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Developing a Multi-chain Stablecoin Protocol via the Collateralized Debt Position (CDP) Mechanism

NGUYỄN ĐỨC THẮNG
thang.nd210778@sis.hust.edu.vn

Program: Cyber Security

Supervisor: Associate Professor Nguyễn Bình Minh

Department: Computer Science

School: School of Information and Communications Technology

HANOI, 01/2026

HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY

GRADUATION THESIS

Developing a Multi-chain Stablecoin Protocol via the Collateralized Debt Position (CDP) Mechanism

NGUYỄN ĐỨC THẮNG
thang.nd210778@sis.hust.edu.vn

Program: Cyber Security

Supervisor: Associate Professor Nguyễn Bình Minh _____

Signature

Department: Computer Science

School: School of Information and Communications Technology

HANOI, 01/2026

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to everyone who has supported me throughout the completion of this graduation thesis. My deepest thanks go to my family, whose constant encouragement and unconditional love have been my greatest motivation. I am also grateful to my friends for their companionship and for always being there during challenging moments. My heartfelt appreciation is extended to my supervisor and the faculty members, whose guidance, patience, and valuable insights have shaped both my academic progress and personal growth. Lastly, I would like to thank myself for the determination, persistence, and countless hours of effort devoted to finishing this work.

ABSTRACT

As blockchain ecosystems continue to expand, the lack of a unified mechanism for maintaining credit, collateralization, and stable value across heterogeneous networks has become a significant limitation for decentralized finance. Existing stablecoin models are predominantly single-chain or rely on centralized bridging infrastructures, resulting in fragmented liquidity, duplicated state, and increased security risks. Although several approaches have attempted to enable cross-chain value transfer, they often introduce trust assumptions, inconsistent state tracking, or high operational complexity. To address these challenges, this thesis adopts a multi-chain architecture built on a Solana-centered Collateralized Debt Position (CDP) system combined with decentralized message relaying from external chains. This approach ensures that all credit, collateral, and risk management logic is executed on a high-performance chain while allowing users to interact from any supported network.

The primary contributions of this thesis include a unified CDP state machine on Solana, a robust cross-chain request verification protocol, a multi-chain mint, burn mechanism without liquidity fragmentation. Experimental validation demonstrates reliable nonce synchronization, secure request execution, and consistent system solvency across chains. This work provides a practical and extensible foundation for interoperable multi-chain stablecoin protocols.

Student

(Signature and full name)

TABLE OF CONTENTS

CHAPTER 1. INTRODUCTION 1

1.1 Motivation 1

1.2 Objectives and scope of the graduation thesis 1

1.3 Tentative solution 2

1.4 Thesis organization 3

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS 5

2.1 Status survey 5

2.2 Functional Overview 7

2.2.1 General use case diagram 7

2.2.2 Detailed use case diagram 8

2.2.3 Business process 9

2.3 Functional Description 10

2.3.1 Use Case: Create Universal Wallet 10

2.3.2 Use Case: Deposit Collateral 11

2.3.3 Use Case: Mint Stablecoin 12

2.3.4 Use Case: Repay Debt 13

2.3.5 Use Case: Withdraw Collateral 13

2.4 Non-functional Requirements 14

2.4.1 Security and Safety 14

2.4.2 Performance and Efficiency 15

2.4.3 Reliability and Availability 15

2.5 Conclusion 15

CHAPTER 3. METHODOLOGY 17

3.1 Introduction 17

3.2 Blockchain Platforms and Architecture 17
3.2.1 Blockchain Fundamentals and Distributed Ledger Technology 17
3.2.2 The Hub: Solana Blockchain Architecture 18
3.2.3 The Spokes: Ethereum Virtual Machine (EVM) Networks 18
3.3 Smart Contract Development Frameworks 19
3.3.1 Solana Program Development: Rust and Anchor 19
3.3.2 EVM Smart Contracts: Solidity and Hardhat 20
3.4 Cryptography and Cross-chain Verification 21
3.4.1 Elliptic Curve Mismatch and Solution 21
3.4.2 Mathematical Formulation of ECDSA Verification 21
3.4.3 Implementation via Solana Native Program 22
3.5 Theoretical Foundation: The CDP Mechanism 22
3.5.1 Economic Model of Stability via Over-collateralization 22
3.5.2 Borrowing Capacity and LTV Formulation 22
3.5.3 The Health Factor Function 23
3.6 Off-chain Infrastructure 23
3.7 Off-chain Infrastructure and Guardian Network 24
3.7.1 Architectural Overview 24
3.7.2 Universal Wallet Initialization via Signature Verification 24
3.8 Conclusion 25
CHAPTER 4. DESIGN, IMPLEMENTATION, AND EVALUATION 26
4.1 Architecture Design 26
4.1.1 Overall design 26
4.1.2 Detailed Package Design 28
4.2 Detailed Design 32
4.2.1 Smart Contract Design 32

4.2.2 Server Application Design 37
4.2.3 Database Design 39
4.3 Application Client and Illustration of main functions 42
4.3.1 User Interface for Universal Wallet 42
4.3.2 User Interface for Cross-chain Borrow 44
4.3.3 User Interface for Portfolio 45
4.4 Application Building 47
4.4.1 Libraries and Tools 47
4.4.2 Achievement 51
4.5 Testing 52
4.5.1 Testing Methodology 52
4.5.2 Test Cases for Critical Functions 52
4.5.3 Evaluation and Results 55
4.6 Deployment 55
CHAPTER 5. SOLUTION AND CONTRIBUTION 57
5.1 Introduction 57
5.2 The State Orchestration Architecture via Hub-and-Spoke Model 57
5.2.1 Problem Identification 57
5.2.2 Proposed Solution 58
5.3 Cross-chain Identity Verification and Request Integrity 58
5.3.1 Problem Identification 58
5.3.2 Proposed Solution 59
5.4 Asynchronous State Consistency via Mutex and Saga Pattern 60
5.4.1 Problem Identification 60
5.4.2 Proposed Solution 61

5.5 Hub-Centric Liquidation Strategy with Liquidity Rebalancing 62

5.5.1 Problem Identification 62

5.5.2 Proposed Solution 62

CHAPTER 6. CONCLUSION AND FUTURE WORK 63

6.1 Conclusion 63

6.2 Future Work 64

REFERENCE 65

LIST OF FIGURES

Figure 2.1 General Use Case Diagram 7
Figure 2.2 Detailed Use Case Diagram 8
Figure 2.3 Business Process Activity Diagram 9

Figure 4.1 Detailed Architecture of the Multi-chain Stablecoin Protocol 27
Figure 4.2 Detailed Design of Solana Hub Package 29
Figure 4.3 Detailed Design of Guardian Middleware Package 30
Figure 4.4 Detailed Design of EVM Controller Package 31
Figure 4.5 Multi-chain Wallet Connectivity Interface 43
Figure 4.6 Universal Wallet Management Interface (Link and List View) 44
Figure 4.7 Cross-chain Borrowing and Position Management Interface . 45
Figure 4.8 User Portfolio and Position Management Interface 46

LIST OF TABLES

Table 2.1 Comparison of Lending Architectures 6

Table 4.1 Development Environment Tools 47
Table 4.2 Smart Contract Development Stack 48
Table 4.3 Frontend Development Libraries 49
Table 4.4 Backend Development Libraries 50
Table 4.5 Testing and Simulation Tools 50
Table 4.6 Source Code Statistics by Component 51
Table 4.7 Test Cases for Security and Verification 54
Table 4.8 Test Cases for Liquidation Logic 54
Table 4.9 Deployed Smart Contract Addresses 56

LIST OF ABBREVIATIONS

Abriviation	Full Expression
ABI	Application Binary Interface
API	Application Programming Interface
BSC	Binance Smart Chain
CDP	Collateralized Debt Position
CPI	Cross-Program Invocation
CPU	Central Processing Unit
DAO	Decentralized Autonomous Organization
DeFi	Decentralized Finance
ERC-20	Ethereum Request for Comments 20
ETH	Ethereum
EVM	Ethereum Virtual Machine
IDL	Interface Description Language
JSON	JavaScript Object Notation
LTV	Loan-to-Value
PDA	Program Derived Address
PoH	Proof of History
RPC	Remote Procedure Call
secp256k1	A specific elliptic curve over a finite field used in public-key cryptography, notably in Bitcoin and Ethereum for ECDSA signatures.
Solana BPF	Solana Berkeley Packet Filter
SVM	Solana Virtual Machine
Tx	Transaction
USDC	USD Cryptocurrency
WBTC	Wrapped Bitcoin
WETH	Wrapped Ether

CHAPTER 1. INTRODUCTION

1.1 Motivation

The rapid development of blockchain ecosystems has created a landscape of many independent networks, each operating with its own assets, smart contract environments, and liquidity pools. Although this diversity has contributed to innovation, it has also introduced a high level of fragmentation that complicates how users interact with digital value. Managing assets across chains remains cumbersome, and transferring liquidity often requires complex operational steps or reliance on intermediaries that weaken the trustless nature of decentralized finance.

Stablecoins, which have become a fundamental component of the digital economy, are still limited by their dependence on single-chain infrastructures or centralized issuers. These restrictions prevent stablecoins from functioning as a truly universal medium of exchange. Users who hold volatile assets on one network cannot easily unlock value on another without passing through bridging systems that may introduce delays, additional fees, and security risks. As a result, the current environment reduces capital efficiency and constrains the usefulness of decentralized financial applications.

Solving this fragmentation is vital to the long-term evolution of decentralized finance. A system that enables unified cross-chain asset management would not only improve how users mint or redeem stable assets but could also serve as a foundation for advanced applications such as multi-chain lending, derivatives, asset management tools, and liquidity optimization frameworks. A reliable mechanism for issuing and managing stable digital assets across chains would help unlock a more coherent, interoperable, and economically efficient blockchain ecosystem.

1.2 Objectives and scope of the graduation thesis

In recent years, several models have attempted to address the challenge of creating stable digital assets that operate across diverse blockchain networks. Custodial stablecoins offer strong usability but depend on centralized entities for asset backing. Over-collateralized decentralized models such as MakerDAO rely on a single-chain design that is difficult to extend to multiple networks without complex bridging layers. Algorithmic stabilization mechanisms explore endogenous supply control but often fail under extreme market conditions. Meanwhile, solutions that rely on existing cross-chain bridges encounter liquidity fragmentation, inconsistent state synchronization, and heightened security risks.

These approaches reveal persistent limitations: user positions cannot be maintained in a unified state across chains, collateral management requires fragmented infrastructure, and expansion to new networks often relies on external systems that reduce security and reliability.

In response to these challenges, this thesis focuses on designing an architecture that maintains a consistent, global collateralized debt position that can be accessed and updated from multiple blockchain networks. The aim is to propose and implement a protocol in which users can lock collateral on one chain while minting stable assets on any supported chain, with the canonical state stored on Solana. The thesis scope includes constructing a multi-chain wallet abstraction, developing a secure verification and message-passing mechanism, enabling updates to user positions originating from external chains, and establishing a stablecoin model capable of minting and burning natively across networks. By addressing existing constraints, the proposed design moves toward a more interoperable multi-chain system for stable asset issuance without dependence on centralized bridging counterparts.

1.3 Tentative solution

To approach the problem defined above, this thesis adopts a design centered on Solana as the authoritative settlement layer, combined with message verification components and smart contracts deployed across EVM networks that serve as user entry points for submitting signed requests. Solana maintains all universal collateralized debt positions through deterministic Program Derived Addresses, which define a unified wallet structure for each user regardless of the number of chains or external wallets they interact with.

User actions such as providing collateral, minting stable assets, repaying obligations, or redeeming collateral are represented as structured requests that include identifiers for the originating chain and sequence information for replay protection. These requests are verified on EVM-side contracts, then observed and relayed by off-chain guardians or backend processes to Solana. The Solana gateway contract validates the messages and forwards them to the main protocol contract, which processes state transitions in accordance with the CDP logic.

The essential contributions of this thesis are the design of a unified multi-chain CDP architecture, the introduction of a secure method for processing cross-chain messages, and the implementation of a mint-and-burn mechanism for stable assets that operates on multiple chains while maintaining a single source of truth on Solana. The expected outcome is a functional prototype that demonstrates secure and synchronized cross-chain stablecoin issuance, along with consistent solvency

and reliable state management.

1.4 Thesis organization

The structure of this thesis is designed to guide the reader through the full development process of the proposed multi-chain stablecoin system, beginning with foundational motivations and ending with practical implementation and evaluation. Each chapter plays a specific role in shaping the final solution and collectively ensures that the research narrative progresses logically from problem identification to system deployment.

Chapter 2 presents a comprehensive requirement survey and analysis. It begins by examining the current situation and the technological context in which multi-chain stablecoin solutions operate. This includes an exploration of existing systems, user needs, and the challenges posed by fragmented blockchain environments. The chapter then introduces the functional overview of the proposed system, incorporating both general and detailed use case diagrams that illustrate how users interact with the system across different contexts. The business processes are also discussed to provide a clear understanding of how information flows through the system. Following this, the chapter offers detailed descriptions of key use cases and concludes with an analysis of the system's non-functional requirements, such as security, performance, scalability, and reliability, which set the baseline for design constraints in later chapters.

Chapter 3 focuses on the methodology adopted in conducting the research and building the system. It outlines the reasoning behind selecting specific technologies, development frameworks, and verification models. The chapter explains the methodological steps taken to ensure scientific rigor, including how the multi-chain architecture was evaluated, how cross-chain communication assumptions were validated, and how the Solana-centered design philosophy influences the broader system. By defining the methodological foundation, the chapter ensures that subsequent design and implementation decisions are grounded in a consistent and justified approach.

Chapter 4 provides an extensive discussion of the system's design, implementation, and evaluation. It begins with the architecture design, explaining the rationale behind software architecture choices and presenting a detailed overview of the system's components and their interactions. The discussion continues with package-level and module-level design, followed by a thorough specification of the user interface layout, the layered backend design, and the on-chain data structures used in both Solana and EVM environments. The chapter also describes the database

schema where applicable, the tools and libraries used throughout development, and the milestones achieved during the building phase. It then illustrates the major functional flows of the system and explains how they were tested to ensure correctness, security, and performance. The chapter concludes with a discussion of the deployment process, including how the system is prepared for real-world execution across multiple blockchain networks.

Chapter 5 highlights the complete solution and the key contributions of the thesis. It synthesizes the work from previous chapters into a coherent model that demonstrates how the system solves the multi-chain stablecoin problem in a unified and scalable way. This chapter also emphasizes the technical and conceptual innovations introduced by the thesis, including unified wallet abstraction, cross-chain request verification, canonical CDP storage on Solana, and multi-chain mint–burn logic for stablecoin issuance. The contributions are contextualized within the broader landscape to clearly show how the proposed solution advances the state of the art.

Finally, Chapter 6 concludes the thesis by summarizing the main results achieved and discussing their implications for blockchain interoperability and decentralized finance. It reflects on the strengths and limitations of the system, and it provides several directions for future work, such as supporting additional chains, improving guardian decentralization, enhancing message throughput, or integrating advanced risk management mechanisms. By outlining these potential extensions, the thesis opens a path for continued development and academic exploration.

CHAPTER 2. REQUIREMENT SURVEY AND ANALYSIS

This chapter presents a comprehensive analysis of the requirements for the development of a Multi-chain Stablecoin Protocol based on the Collateralized Debt Position (CDP) mechanism. The chapter begins with a survey of the current Decentralized Finance (DeFi) landscape and analyzes existing solutions to identify limitations regarding cross-chain liquidity. Subsequently, it provides a functional overview of the proposed system through general and detailed use case diagrams. The core business processes, specifically the cross-chain state synchronization workflow, are illustrated to clarify the operation of the system. Finally, detailed functional descriptions of critical use cases and non-functional requirements regarding security, performance, and scalability are established to guide the subsequent design and implementation phases.

2.1 Status survey

The survey of the current technology landscape relies on three primary sources: (i) the needs of DeFi users seeking capital efficiency, (ii) existing single-chain stablecoin protocols, and (iii) current cross-chain infrastructure solutions.

Currently, the DeFi ecosystem exhibits severe fragmentation, particularly within lending protocols and CDP mechanisms. Under the prevailing model, users are restricted to establishing collateralized positions exclusively within a single blockchain network. Prominent platforms such as MakerDAO and Aave operate in isolated silos, preventing cross-chain collateral utility. Consequently, users face a complex and inefficient workflow when attempting to utilize capital across networks, often necessitating manual bridging procedures that incur high transaction costs and operational friction.

Analysis of Existing Systems

To identify the gap in the current market, this research analyzes two dominant categories of lending protocols.

Firstly, I want to talk about the isolated CDP protocols (e.g., MakerDAO). These represent the standard for decentralized stablecoins. They offer high security and proven economic models. However, they operate in strict isolation. A user with collateral on Ethereum cannot leverage this value to mint stablecoins on other networks (like Solana, Sui) and Layer 2 networks (like Arbitrum or Optimism) without physically bridging the underlying assets. This limitation forces users to choose between security (keeping assets on the one chain) and utility (using low-cost chains),

resulting in significant capital inefficiency.

Secondly, "Cross-Chain Money Markets" (e.g., Radiant Capital) have effectively streamlined the user workflow, enabling seamless cross-chain borrowing without manual bridging steps. However, their architecture heavily relies on specific third-party interoperability layers, such as LayerZero, for message passing and asset transfers. This dependency introduces a critical single point of failure: the protocol's security is inextricably linked to the third-party bridge. Consequently, users are exposed to external systemic risks, and the protocol lacks sovereignty over its own verification logic.

Proposed Solution Analysis

The proposed system addresses the dependency risks of existing cross-chain markets by implementing a State Orchestration Architecture centered on Solana. Unlike protocols that outsource security entirely to general-purpose messaging layers, this solution maintains sovereignty over verification logic. The "Universal Wallet" mechanism ensures that the state is unified, while the validity of cross-chain requests is cryptographically verified on-chain (via Solana's Secp256k1 program) rather than relying solely on the trust assumptions of third-party bridges.

Table 2.1 highlights the strategic advantages of this approach, specifically regarding security sovereignty and state management.

Feature	**Isolated CDPs (e.g., MakerDAO)**	**Cross-Chain Markets (e.g., Radiant)**	**Proposed System (Universal USD)**
State Model	Isolated	Fragmented Pools	Unified Global State
Verification	Native On-chain Verification	Trusted Third-Party	On-chain Verification (Solana)
3rd Party Risk	None	High (Bridge dependency)	Minimized (Cryptographic proof required)
Capital Effic	Low (Trapped on one chain)	High	High

Table 2.1: Comparison of Lending Architectures

2.2 Functional Overview

The system is designed to function as a decentralized lending protocol that orchestrates state across the Solana blockchain and various EVM-compatible chains.

2.2.1 General use case diagram

The system involves three primary actors:

(i) User: The borrower who holds collateral on EVM chains. They interact with the system by Lending Operations which include depositing assets, borrowing, repaying, and withdrawing, and manage their global debt position.

(ii) Guardian: A decentralized off-chain node responsible for listening to events on EVM chains, relaying signatures to Solana for verification, and synchronizing the execution results back to the EVM chains.

(iii) Liquidator: An actor who monitors the health factor of Universal Wallets on Solana. If a user's position becomes under-collateralized, the liquidator triggers the liquidation process.

Figure 2.1 illustrates the general use cases.

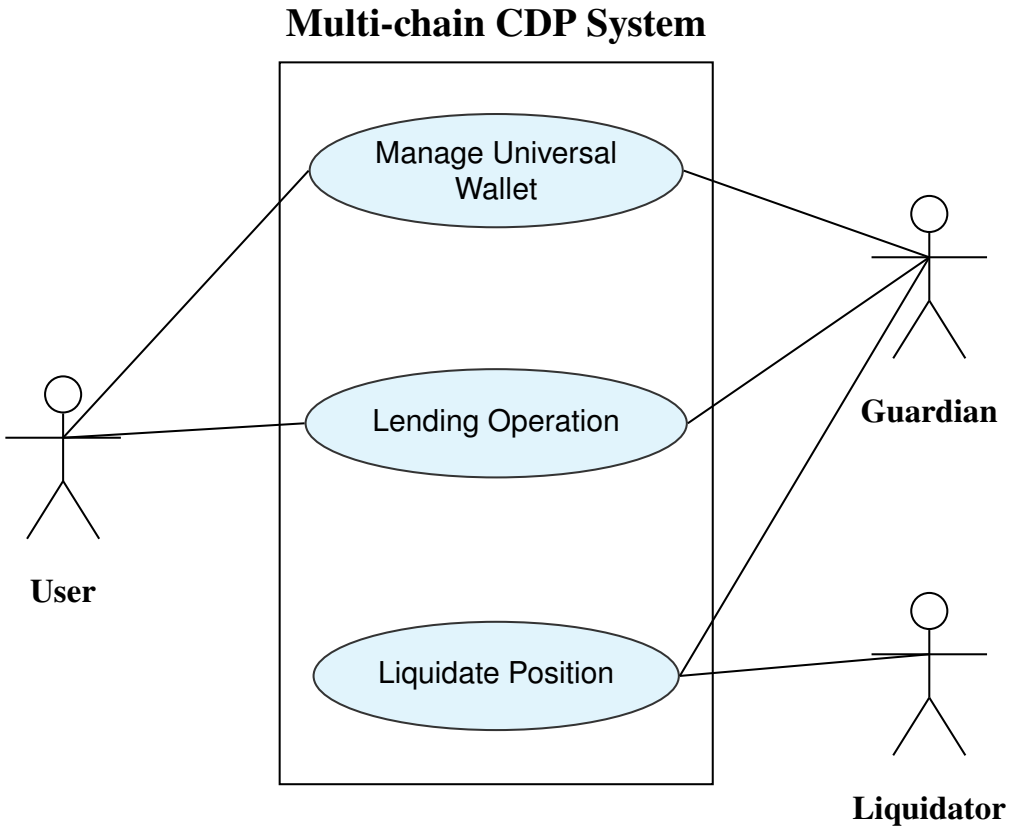

Figure 2.1: General Use Case Diagram

2.2.2 Detailed use case diagram

To provide a general view of the system's operation, Figure 2.2 illustrates the decomposition of the core cross-chain workflows. This diagram emphasizes the dependency relationships between user actions and the underlying system processes, specifically highlighting the "Sign-then-Relay" mechanism. The diagram

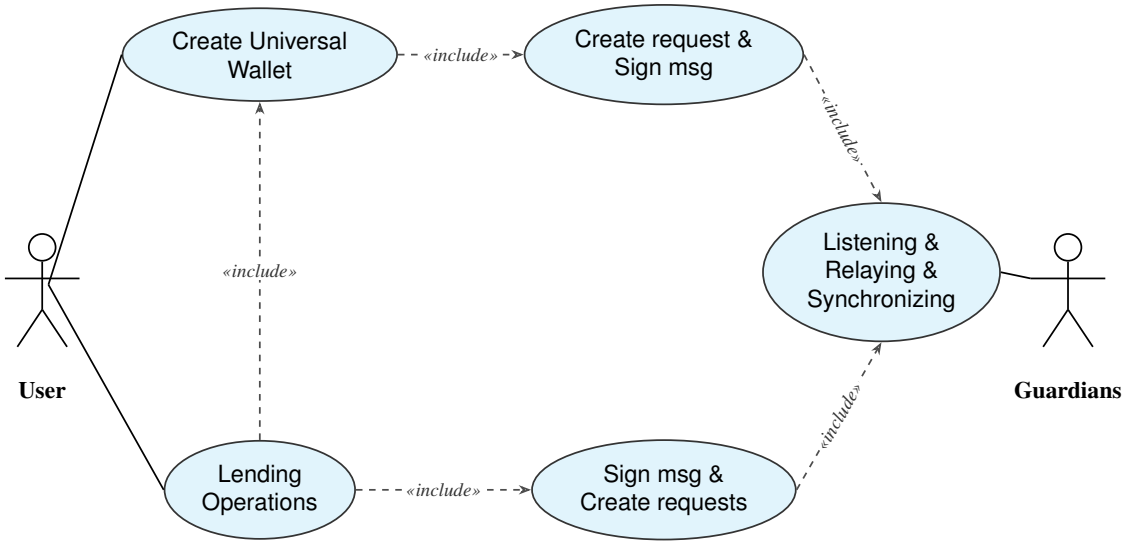

Figure 2.2: Detailed Use Case Diagram

delineates several critical logical dependencies that govern the system's operation. Foremost among these is the dependency on the Universal Wallet; the core financial use cases Lending Operations containing "Deposit, Borrow, Repay, and Withdraw" maintain an *include* relationship with the "Create Universal Wallet" use case. This structure enforces a strict precondition, mandating that a user must establish a valid identity via a Universal Wallet PDA on Solana prior to executing any asset-related operations.

Furthermore, the security model is illustrated through the cryptographic authorization flow. Both the wallet creation and asset management workflows include the "Sign msg & Create requests" sub-use case. This relationship highlights that users do not interact directly with the Solana blockchain's state; instead, they generate and cryptographically sign messages within the EVM environment to authorize their intent. Consequently, this message creation process extends to include the "Listening & Relaying & Synchronizing" use case executed by the Guardian actor. This signifies that the lifecycle of a user's request is only finalized when the Guardian successfully intercepts the emitted event, relays the payload to Solana for verification, and synchronizes the execution result back to the source chain.

2.2.3 Business process

The system operates on a Cross-chain State Orchestration model, where the logic execution is decoupled from asset custody. The workflow involves five distinct entities: the User, the Controller EVM Contract, the Guardian Network, the Solana Gateway, and the Lending CDP Core.

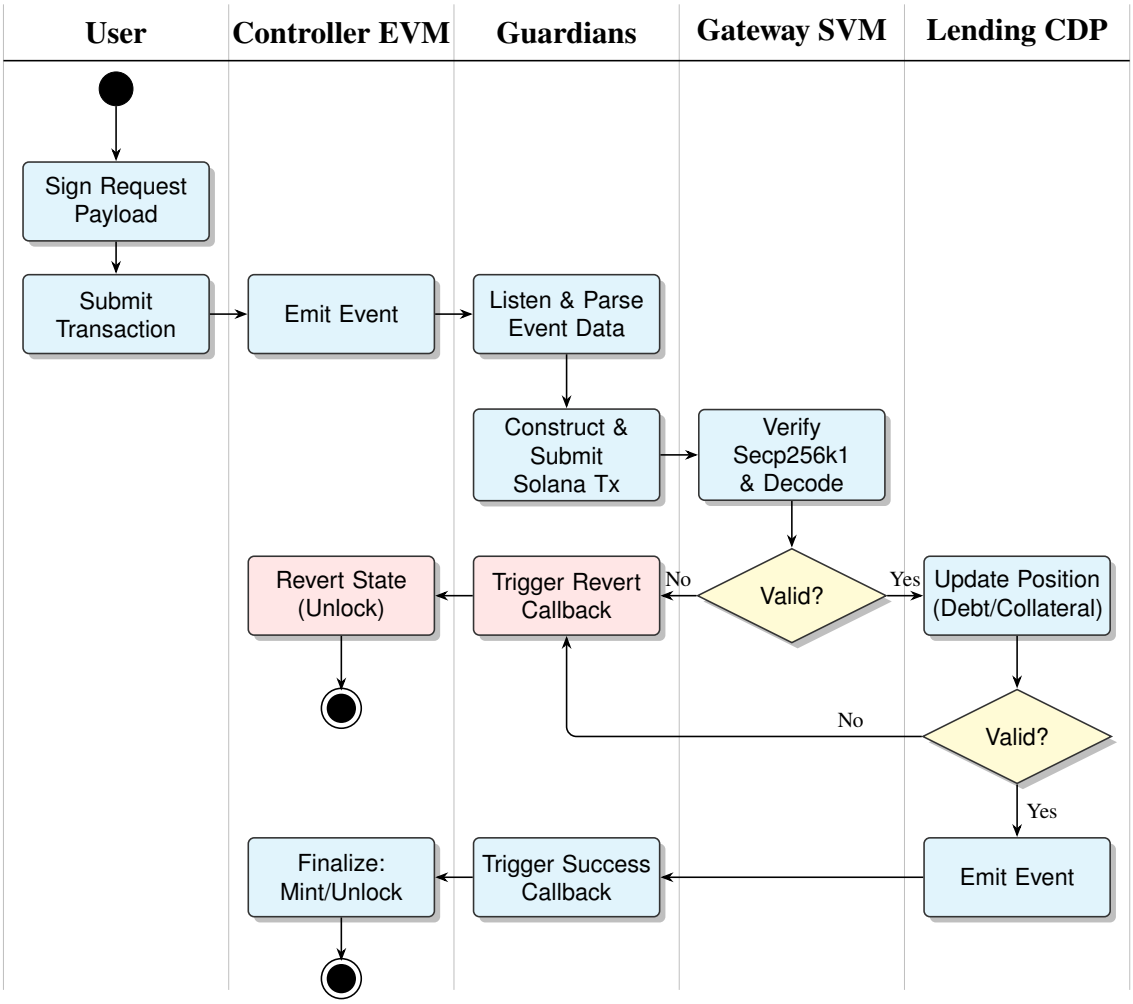

Figure 2.3: Business Process Activity Diagram

As illustrated in Figure 2.3, the process follows a strict "Verify-then-Execute" lifecycle:

(i) Initiation (User in EVM): The process begins when the User cryptographically signs a specific request payload (containing the action type, amount, and nonce) and submits the transaction to the Controller EVM Contract. The contract emits a event. Note that at this stage, no minting or unlocking occurs; the assets are simply locked or the request is queued.

(ii) Observation & Relay (Guardians): The Guardian network detects the event on the EVM chain. Instead of verifying the logic off-chain, the Guardian parses the event data and encapsulates the raw signature into a transaction destined

for Solana.

(iii) Verification Layer (Solana Gateway): The Gateway Contract on Solana acts as the security checkpoint. It utilizes the native Secp256k1 program to verify that the signature matches the User's EVM address.

- If the signature is Invalid, the Gateway immediately signals a failure, bypassing the core logic.
- If the signature is Valid, the request is forwarded to the Lending CDP contract.

(iv) Logic Layer (Solana Lending CDP): The Lending CDP Contract attempts to update the user's position (e.g., increasing Debt). It performs critical checks such as Health Factor validation (e.g., Collateral Ratio $> 150\%$).

- If the logic holds (Logic OK), a event is emitted.
- If the logic fails (e.g., under-collateralized), the transaction is rejected.

(v) Synchronization & Finalization: Guardians listen for the outcome on Solana to trigger the corresponding callback on the EVM chain:

- If a event is captured, Guardians trigger the *Success Callback* on the EVM contract to finalize the action (e.g., Mint stablecoins).
- If the verification or logic failed, Guardians trigger the *Revert Callback* to unlock the user's assets and reset the nonce, ensuring funds are never stuck.

2.3 Functional Description

This section details the critical use cases of the system, covering the full lifecycle of a user's interaction from the initial identity creation to advanced position management.

2.3.1 Use Case: Create Universal Wallet

The creation of a Universal Wallet is the foundational procedure for establishing a user's cross-chain identity within the protocol. This process requires the coordination of four primary actors: the User, the off-chain Guardian, the EVM Contract, and the Solana Main Contract.

a, Pre-conditions

The execution of this use case is predicated on specific prerequisites. First, the User must possess a functional EVM wallet (e.g., MetaMask) with sufficient native tokens to cover the gas fees for the initial request transaction. Second, the User

must hold a valid Solana wallet key pair, which is required to generate the cryptographic signature proving ownership of the destination address. Without these prerequisites, the binding process cannot commence.

b, Flow of Events

The workflow initiates on the EVM chain, where the User submits a transaction to the EVM Contract to formally request the creation of a wallet. This action triggers the contract to emit an event containing the user's EVM address, acting as a signal to the off-chain infrastructure. Simultaneously, to establish a proof of ownership for the destination, the User performs an off-chain action by signing a specific message containing their EVM address using their Solana private key. This signature is securely transmitted to the Guardian's API.

Subsequently, the Guardian captures the on-chain EVM event and cross-references it with the submitted off-chain Solana signature. Upon successful verification that the User controls both addresses, the Guardian constructs and submits a transaction to the Solana Main Contract. The process culminates when the Solana Main Contract allocates a new Program Derived Address (PDA) for the Universal Wallet, mapping the EVM identity to a unified storage state.

c, Post-conditions

Upon completion, a unique "Universal Wallet" structure is successfully initialized on the Solana blockchain. The User is thereafter authorized to perform cross-chain financial operations using this mapped identity.

2.3.2 Use Case: Deposit Collateral

This use case describes the mechanism by which users lock assets on an EVM chain to increase their collateral balance recorded on the Solana state.

a, Pre-conditions

The User must have previously initialized a Universal Wallet and must hold supported assets on the EVM chain sufficient for the deposit.

b, Flow of Events

The process begins with the User signing a payload and submitting the signature by calling the request Deposit function on the EVM Contract. Consequently, the assets are transferred to the contract's vault, and an event is emitted. The Guardian, monitoring the network, detects this event and reads the on-chain data. It then forwards the payload and the user's signature to the Solana Gateway.

On the Solana side, the Gateway contract verifies the signature and the integrity of the payload. Upon validation, the Solana Main Contract identifies the user's Uni-

versal Wallet and increments the collateral balance corresponding to the specific chain ID. Finally, the Guardian executes a synchronization step, updating the EVM contract to confirm that the deposit request has been synced, thereby allowing the user to proceed with subsequent actions.

c, Post-conditions

The specified collateral is securely locked in the EVM vault, and the collateral balance within the Universal Wallet on Solana is incremented to reflect this deposit.

2.3.3 Use Case: Mint Stablecoin

This process allows users to generate stablecoins against their collateral. This action mandates strict verification of the Health Factor on the Solana state to ensure system solvency.

a, Pre-conditions

The User must possess sufficient collateral such that the Health Factor remains above the minimum required ratio after the new debt is issued.

b, Flow of Events

The User initiates the process by signing a mint request payload and calling the request Mint function on the EVM Contract. Crucially, the contract immediately locks the User's mutex to prevent race conditions. The Guardian, upon listening to the emitted event, submits the signature and payload to Solana. The Main Contract verifies the signature and calculates the projected Health Factor.

If the Health Factor is valid, the Solana contract increases the User's debt balance. Observing the success event from Solana, the Guardian executes a callback transaction on the EVM Contract to physically mint the stablecoins to the User's wallet and release the mutex lock.

c, Alternative Flow

In the event that the Health Factor is insufficient, the Solana transaction will fail. Detecting this failure, the Guardian triggers a Revert function on the EVM Contract. This action ensures the User's mutex is unlocked without minting any tokens, restoring the system to its previous valid state.

d, Post-conditions

Stablecoins are minted to the User's address on the EVM chain, and the corresponding debt position is recorded in the Universal Wallet on Solana.

2.3.4 Use Case: Repay Debt

Users utilize this workflow to return stablecoins, thereby reducing their debt position and improving their Health Factor.

a, Pre-conditions

The User must hold a sufficient balance of the protocol's stablecoin on the EVM chain to cover the repayment amount.

b, Flow of Events

The User signs the payload and calls the request Repay function on the EVM Contract to submit the signature. The specified amount of stablecoins is locked immediately by the contract, and an event is emitted. The Guardian observes this event and relays the data and signature to Solana.

Upon receipt, the Solana Gateway verifies the signature, and the Main Contract processes the transaction by decreasing the User's debt balance in the Universal Wallet. To finalize the process, the Guardian confirms the state update back to the EVM chain, triggering the burning of the locked stablecoins and updating the user's nonce.

c, Post-conditions

The stablecoins are permanently burned on the EVM chain, and the debt balance recorded on Solana is decreased accordingly.

2.3.5 Use Case: Withdraw Collateral

This action allows users to retrieve their locked assets, provided that their remaining collateral is sufficient to support their outstanding debt.

a, Pre-conditions

The User must have sufficient free collateral, ensuring that the withdrawal does not cause the Health Factor to drop below the liquidation threshold.

b, Flow of Events

The User signs a withdraw request payload and submits it to the EVM Contract. The contract locks the User's mutex and, after verifying sufficient collateral exists in the EVM vault, emits an event. The Guardian forwards this request to Solana. The Main Contract performs a solvency check to determine if the remaining collateral covers the debt.

If the check passes, the Solana contract decreases the User's collateral balance. The Guardian then triggers the EVM Contract to transfer the requested assets from the vault back to the User's wallet.

c, Alternative Flow

If the withdrawal results in an insolvent position, the Solana transaction is rejected. Consequently, the Guardian executes a revert transaction on the EVM chain. This action keeps the assets locked in the vault and releases the mutex, preventing the user from withdrawing funds that secure active debt.

d, Post-conditions

The User receives the requested assets on the EVM chain, and the collateral balance within the Universal Wallet on Solana is reduced.

2.4 Non-functional Requirements

To ensure the Multi-chain CDP Protocol operates securely, efficiently, and reliably in a high-value financial environment, the system must adhere to the following strict non-functional requirements.

2.4.1 Security and Safety

Given the cross-chain nature of the protocol, security is the paramount requirement to prevent fund loss and bridge exploits. At the foundational level, the system mandates strict cryptographic compatibility. Specifically, the Solana Gateway must natively support and verify Secp256k1 signatures, which is the standard for EVM networks. This capability allows users to control their positions using existing Ethereum wallets without ever exposing their private keys to third parties. Furthermore, to safeguard against cross-chain replay attacks, the protocol implements a rigorous nonce management mechanism. Each transaction request must include a unique nonce tied to a specific Chain ID and User Address, allowing the Solana contract to deterministically reject any request with a nonce lower than or equal to the currently stored value.

In parallel with on-chain security, the off-chain infrastructure addresses the trust assumption of the Guardian network by decentralizing operations to prevent any single point of failure. The system requires a threshold of signatures, utilizing a Multisig or Threshold Signature Scheme from the Guardians before executing sensitive state changes, such as unlocking collateral. This distributed consensus effectively mitigates the risk associated with a single compromised Guardian node. Finally, the protocol ensures absolute financial safety through atomic revert capabilities. In the event of a failure during cross-chain synchronization, such as a Solana transaction reverting due to market slippage or an insufficient health factor, the system guarantees that the initial state on the EVM chain is successfully reverted. This automatic rollback releases the user's Mutex lock, preventing funds

from being permanently frozen in a transitional state.

2.4.2 Performance and Efficiency

Regarding performance and efficiency, a primary objective is the minimization of end-to-end latency for cross-chain operations. The total duration for a generic user action, such as minting, is defined as the cumulative sum of the EVM confirmation time, the Guardian relay processing, the Solana finality period, and the final Guardian callback. To ensure a responsive and fluid user experience, the system is engineered to aim for a target execution time of under 30 seconds, assuming normal network conditions and excluding periods of extreme congestion on the Ethereum Mainnet.

Simultaneously, efficiency on the Solana Hub is governed by strict computational constraints. Given that the cryptographic verification of Secp256k1 signatures, necessary for authenticating EVM users, which is a computationally intensive operation, the Solana smart contracts must be rigorously optimized. This optimization is essential to ensure that instruction execution remains strictly within the block Compute Unit limit, thereby preventing transaction failures due to resource exhaustion and maintaining cost-effectiveness.

2.4.3 Reliability and Availability

To guarantee reliability and availability, the system is fundamentally designed around the principle of eventual consistency. This ensures that despite the asynchronous nature of distributed ledgers, the state maintained on the EVM Spokes will always eventually converge with the authoritative state on the Solana Hub. In scenarios involving network partitions or Guardian downtime, the system retains the capability to recover autonomously, processing any pending events once connectivity is restored. Furthermore, to handle the unreliability of network transmission, all Guardian operations are implemented with strict idempotency. This property ensures that submitting the same event multiple times, which is a common occurrence during network retries, does not result in the erroneous double-counting of debt or collateral. Finally, the infrastructure mandates a high availability standard of 99.9% for both the Guardian nodes and the Solana RPC endpoints. This rigorous uptime requirement is critical to prevent liquidation failures during periods of high market volatility, thereby preserving system solvency.

2.5 Conclusion

This chapter has provided a comprehensive analysis of the operational boundaries and functional necessities for the proposed Multi-chain Stablecoin Protocol. By critically evaluating the limitations of existing "Lock-and-Mint" bridges and

single-chain CDP models, the study established the rationale for adopting a State Orchestration architecture, where Solana serves as the global state machine for liquidity scattered across EVM chains.

The functional analysis elucidated the complex workflows involving the Universal Wallet, emphasizing the pivotal role of the Guardian Network in maintaining cross-chain atomicity and data synchronization. Furthermore, the non-functional requirements have set strict constraints regarding cryptographic compatibility, system latency, and security against replay attacks. These specifications serve as the foundational blueprint for the architectural design and technical implementation strategies that will be presented in the subsequent chapters.

CHAPTER 3. METHODOLOGY

3.1 Introduction

This chapter establishes the theoretical framework and technological infrastructure utilized to construct the Multi-chain Stablecoin Protocol. The methodology is grounded in a rigorous analysis of distributed ledger technologies, cryptographic primitives, and financial stability mechanisms. By synthesizing the high-performance capabilities of the Solana blockchain with the deep liquidity of EVM-compatible networks, the proposed solution adopts a Hub-and-Spoke architecture. This chapter provides a detailed exposition of the selected blockchain platforms, the software engineering principles behind the smart contracts, the mathematical foundations of the cross-chain verification process, and the economic theory underpinning the CDP mechanism.

3.2 Blockchain Platforms and Architecture

3.2.1 Blockchain Fundamentals and Distributed Ledger Technology

The technological cornerstone of this thesis is the blockchain, a concept fundamentally introduced as a purely peer-to-peer version of electronic cash. As delineated by Satoshi Nakamoto in the seminal Bitcoin whitepaper, a blockchain is a distributed system that solves the double-spending problem without relying on a trusted central authority or a financial institution [1]. The core innovation lies in the use of a timestamp server, which works by taking a hash of a block of items to be timestamped and widely publishing the hash. This process proves that the data must have existed at the time, creating a chronological chain of hashed proof-of-work.

This structure forms an immutable record of events. Each timestamp includes the previous timestamp in its hash, forming a chain where each additional timestamp reinforces the ones before it. In the context of a financial protocol, this mechanism ensures integrity and transparency. The network operates on a consensus mechanism where the longest chain not only serves as proof of the sequence of events witnessed, but proof that it came from the largest pool of CPU power. This guarantees that as long as a majority of CPU power is controlled by nodes that are not cooperating to attack the network, they will generate the longest chain and outpace attackers.

While Bitcoin established these principles for decentralized currency, the underlying technology, the distributed ledger provides the essential substrate for the

Multi-chain Stablecoin Protocol. The guarantee that a transaction (such as depositing collateral) is irreversible once sufficiently buried under proof-of-work (or proof-of-stake in modern iterations) is critical for maintaining the solvency of a CDP system. This thesis leverages these foundational properties of trustlessness and cryptographic proof to construct a secure financial architecture across disparate networks.

3.2.2 The Hub: Solana Blockchain Architecture

To address the latency and throughput limitations inherent in earlier distributed ledger technologies, this project selects Solana as the central "Hub" for orchestrating the global financial state. The decision is grounded in the architectural innovations proposed by Yakovenko, which introduce a novel method for high-performance blockchain execution that does not rely on wall clock timestamps or local time for synchronization [2].

The defining feature of this architecture is Proof of History (PoH). As defined in the whitepaper, PoH is a sequence of computation that provides a way to cryptographically verify the passage of time between two events. It essentially functions as a high-frequency, verifiable delay function. By encoding the passage of time directly into the ledger, Solana allows the network to create a historical record that proves that a specific event occurred at a specific moment in time, before or after other events. This cryptographic clock eliminates the need for massive messaging overhead typically required for nodes to agree on a timestamp, which is a primary bottleneck in traditional consensus mechanisms.

This architectural shift allows the system to implement a deterministic leader schedule and streamline the consensus process. Because the order of events is verifiable via the PoH data structure, the network can process transactions as they arrive rather than waiting to batch them into blocks based on uncertain network latencies. For the Multi-chain Stablecoin Protocol, this architecture is critical. The ability to verify time and order with sub-second latency ensures that the "Universal Wallet" state remains synchronized and that liquidation triggers based on Health Factor calculations are executed with the speed and determinism necessary to maintain protocol solvency.

3.2.3 The Spokes: Ethereum Virtual Machine (EVM) Networks

While the Solana Hub manages the execution state, the Ethereum Virtual Machine (EVM) ecosystem serves as the critical Asset Custody Layer, or the "Spokes," of the architecture. The selection of EVM, compatible networks is rooted in the foundational design of Ethereum as a "next-generation smart contract and decen-

tralized application platform." As articulated in the Ethereum whitepaper, the core innovation of this system is the introduction of a Turing-complete programming language, allowing anyone to write smart contracts and decentralized applications where they can create their own arbitrary rules for ownership, transaction formats, and state transition functions [3].

The EVM operates as a quasi-Turing complete machine that maintains a singleton state computer shared by all participants in the network. Every transaction in this model is a cryptographically signed instruction that transitions the global state from one valid state to another. This state is maintained in a specialized data structure known as the Merkle Patricia Tree, which ensures a high degree of efficiency and cryptographic security for light client verification. The protocol leverages this robust state architecture to deploy "Vault" contracts. These contracts utilize the standardized ERC-20 interface, which a direct derivative of Ethereum's programmable token capability to securely lock collateral assets.

By integrating the EVM as the spoke layer, the protocol taps into the concept of "programmable money" envisioned in the whitepaper. While the EVM's design prioritizes security and generality over high-frequency throughput, its deterministic execution environment provides the ideal substrate for holding significant value. The architecture thus relies on the EVM not for rapid calculation, but for its established consensus and the immense liquidity of assets like Ether, which are native to this "World Computer" paradigm.

3.3 Smart Contract Development Frameworks

3.3.1 Solana Program Development: Rust and Anchor

The development of the protocol's core logic on the Solana Hub is architected utilizing the Anchor Framework, a sophisticated development tool designed to abstract the complexities of the Sealevel runtime. While Rust provides the underlying memory safety through its ownership and borrowing models, raw Solana program development requires developers to manually implement verbose serialization logic and low-level byte manipulation. Anchor addresses these challenges by introducing a framework that enforces security constraints and streamlines the development lifecycle through an opinionated design pattern [4].

A primary motivation for adopting Anchor is its rigorous approach to account validation and security, specifically through the implementation of the "Discriminator." In the native Solana programming model, any account passes as a byte array, making the system vulnerable to type-confusion attacks where a malicious actor might pass a data account of one type into an instruction expecting another.

Anchor mitigates this by automatically prepending a unique 8-byte discriminator derived from the SHA-256 hash of the account structure's name to the account data. During execution, the framework verifies this discriminator before deserializing the data.

Furthermore, Anchor standardizes the interaction between the on-chain program and off-chain clients through the IDL. Similar to an ABI in the EVM ecosystem, the IDL provides a structured JSON representation of the program's instructions and account structures. This allows the Guardian infrastructure to deterministically serialize inputs and deserialize outputs without manual offset calculations. By handling the "boilerplate" code of account serialization and constraint checks (such as ensuring an account is mutable or is a signer) via macros, Anchor allows the development focus to remain on the high-level financial logic of the CDP mechanism, ensuring that the state transitions on the Solana Hub are both secure and predictable.

3.3.2 EVM Smart Contracts: Solidity and Hardhat

For the Asset Custody Layer residing on the EVM Spokes, the development methodology prioritizes security and standard compliance over raw execution speed. Accordingly, the Solidity programming language is utilized to construct the Controller and Vault contracts. As a statically-typed, contract-oriented language designed specifically for the Ethereum Virtual Machine, Solidity allows for the precise definition of state variables and the implementation of complex access control logic required to secure collateral assets. The choice of Solidity ensures maximum compatibility with the ERC-20 token standard, which is the fundamental data structure for the assets (such as USDC or WETH) that the protocol accepts as collateral.

To ensure a robust development lifecycle, the project leverages the Hardhat development environment. Hardhat is instrumental not merely as a compiler, but for its advanced testing capabilities, particularly "Mainnet Forking." This feature allows the development team to simulate the protocol's interaction with the real Ethereum state and existing token contracts in a deterministic local environment, validating the Vault's logic against actual network conditions before deployment.

Crucially, to mitigate the inherent risks of smart contract vulnerabilities, the system integrates the OpenZeppelin library. Given that the Controller contract functions as a decentralized vault holding user funds, it is a high-value target for exploits. Rather than relying on custom security implementations, the contracts inherit from battle-tested modules. Specifically, the "ReentrancyGuard" is applied to all asset withdrawal functions to mathematically prevent recursive call attacks,

while the "Ownable" pattern is utilized to enforce strict access control, ensuring that only the authenticated Guardian infrastructure can trigger state-changing callbacks [5].

3.4 Cryptography and Cross-chain Verification

3.4.1 Elliptic Curve Mismatch and Solution

A fundamental technical challenge in implementing a "Universal Wallet" lies in the cryptographic incompatibility between the disparate blockchain networks. The Ethereum ecosystem relies on the Elliptic Curve Digital Signature Algorithm (ECDSA) utilizing the secp256k1 curve. In contrast, the Solana blockchain is built upon the Ed25519 curve, which offers faster signature verification times but different mathematical properties. This discrepancy implies that a standard Solana smart contract cannot natively verify a signature generated by an Ethereum private key without significant computational overhead.

3.4.2 Mathematical Formulation of ECDSA Verification

To bridge this gap, the protocol requires a mechanism to mathematically prove that a transaction request m originating from an EVM address A_{evm} was indeed authorized by the holder of the private key d. The domain parameters of the secp256k1 curve are defined over a finite field $\mathbb{F}_p$ by the Weierstrass equation:

$$y^2 \equiv x^3 + 7 \pmod{p} \tag{3.1}$$

When a user signs a cross-chain request, they produce a signature pair (r, s). The verification process, which must be executed on the Solana Hub, involves recovering the public key point Q from the signature. This is achieved using the inverse operation of point multiplication:

$$Q = r^{-1}(sR - zG) \tag{3.2}$$

Where z is the hash of the message m, G is the generator point of the curve, and R is the point with x-coordinate r. The derived Ethereum address is then the last 20 bytes of the Keccak-256 hash of Q:

$$A_{evm} = \text{Keccak256}(Q)[12..32] \tag{3.3}$$

3.4.3 Implementation via Solana Native Program

Implementing the arithmetic operations of Equation 3.2 directly in a high-level language like Rust would consume an excessive amount of Compute Units, potentially exceeding the block limit. To solve this, the methodology employs Solana's "Native Secp256k1 Program". This is a precompiled BPF program embedded in the validator software that executes signature recovery at a fraction of the computational cost. By invoking this program via CPI, the Main Contract can trustlessly verify EVM signatures, thereby allowing users to control their Solana state using their existing MetaMask wallets without compromising security.

3.5 Theoretical Foundation: The CDP Mechanism

3.5.1 Economic Model of Stability via Over-collateralization

The financial architecture of the proposed protocol is grounded in the CDP model, a primitive that enables the creation of decentralized stablecoins without reliance on fiat reserves or centralized custodians. Unlike algorithmic stablecoins which often depend on endogenous collateral and reflexive arbitrage cycles, mechanisms historically prone to "death spirals" during market downturns. In this protocol, the CDP model enforces stability through strict Over-collateralization. This economic theory posits that the solvency of the system and the peg of the stablecoin are guaranteed as long as the aggregate value of the locked collateral assets significantly exceeds the value of the issued debt. In this system, the stablecoin is not created out of thin air but is backed by a diversified basket of crypto-assets (such as ETH, WBTC) locked across various EVM chains, serving as a secure liability against the protocol's assets.

3.5.2 Borrowing Capacity and LTV Formulation

A critical component of the CDP mechanism is the determination of a user's borrowing power. This is governed by the LTV ratio, a risk parameter assigned to each asset type based on its market volatility and liquidity profile. The LTV ratio, denoted as α, represents the maximum percentage of the collateral's value that can be minted as debt. For instance, a stable asset like USDC might have a high α (e.g., 0.90), while a volatile asset like ETH might have a lower α (e.g., 0.75).

Mathematically, the Maximum Borrowable Debt (D_{max}) for a user holding a portfolio of assets across multiple chains is the sum of the risk-adjusted value of each asset. For a user depositing N different asset types distributed across M blockchain networks, the borrowing capacity is formalized by the following equation:

$$D_{max} = \sum_{k=1}^{M} \sum_{i=1}^{N} \left(Q_{k,i} \cdot P_i \cdot \alpha_i \right) \tag{3.4}$$

In this equation, $Q_{k,i}$ represents the quantity of asset i locked on chain k, and P_i is the real-time market price of asset i denominated in the reference currency (USD). The term α_i acts as a dampening factor, ensuring that the protocol accounts for potential price drops before they occur. Consequently, the actual debt D_{actual} minted by the user must always satisfy the condition $D_{actual} \leq D_{max}$ at the time of issuance.

3.5.3 The Health Factor Function

The system solvency is mathematically modeled using the Health Factor (H_f). This metric is dynamic and is updated in real-time based on oracle price feeds. For a multi-chain protocol, the Health Factor is an aggregate function of all collateral assets i across all connected chains k:

$$H_f = \frac{\sum_{k=1}^{M} \sum_{i=1}^{N} (Q_{k,i} \times P_i \times \lambda_i)}{D_{total} \times P_{peg}} \tag{3.5}$$

In this equation:

- $Q_{k,i}$ represents the quantity of asset i locked on chain k.
- P_i is the current market price of asset i denominated in USD.
- λ_i is the Liquidation Threshold specific to the asset's risk profile (e.g., $\lambda_{ETH} = 0.80$, $\lambda_{USDC} = 0.95$).
- D_{total} is the total outstanding stablecoin debt.
- P_{peg} is the target peg price of the stablecoin (e.g., 1 USD).

The methodology enforces a strict constraint where $H_f \geq 1$. If market volatility causes $H_f < 1$, the protocol triggers a liquidation event. This deterministic mathematical model ensures that the protocol remains solvent and the stablecoin maintains its peg, fulfilling the core objective of the thesis.

3.6 Off-chain Infrastructure

The final component of the methodology is the Guardian Network, which serves as the interoperability layer connecting the deterministic worlds of the blockchain networks.

3.7 Off-chain Infrastructure and Guardian Network

3.7.1 Architectural Overview

The Off-chain Infrastructure, referred to as the Guardian Network, functions as the cryptographic bridge and synchronization engine between the deterministic environments of the EVM Spokes and the Solana Hub. This component is engineered using Python. Unlike passive indexers, the Guardian plays an active role in state transition, specifically in the secure initialization of user identities and the cross-chain relaying of assets.

3.7.2 Universal Wallet Initialization via Signature Verification

A critical security function of the Guardian is the verification of user intent during the creation of the Universal Wallet. Since the Universal Wallet acts as the central storage for a user's multi-chain positions, it is imperative to establish a cryptographically proven link between the user's request and their destination wallet on Solana before any on-chain state is modified.

The process is designed as a "Verify-then-Execute" workflow to prevent Denial of Service attacks and ensure that only authenticated users can allocate storage on the Solana blockchain. The workflow proceeds as follows:

First, the user constructs a request message m, which includes the intent to create a wallet, a unique nonce to prevent replay attacks, and the public key of the destination Solana wallet (PK_{dest}). To prove ownership of this destination wallet, the user signs the message m using their corresponding private key, generating a digital signature σ. This signature typically utilizes the Ed25519 algorithm, which is the native standard for Solana addresses.

Second, the Guardian receives this payload (m, σ, PK_{dest}) via a secure API endpoint. Instead of immediately submitting a transaction to the blockchain, the Guardian performs an off-chain verification using cryptographic libraries. The verification function can be formalized as:

$$V(m, \sigma, PK_{dest}) \rightarrow \{\text{True, False}\} \tag{3.6}$$

If the function returns False, the Guardian rejects the request immediately, ensuring that no gas fees are wasted on invalid transactions.

If the function returns True, the Guardian confirms that the requestor possesses the private key controlling the destination wallet. Subsequently, the Guardian constructs a transaction instruction invoking the Initialize Wallet method on the Solana

Main Contract. The Guardian then signs this transaction with its own keyer key (to pay for transaction fees) and submits it to the Solana cluster. Upon successful execution, the Solana state is updated: a new PDA is initialized, serving as the Universal Wallet for that specific user. This mechanism ensures that the heavy lifting of cryptographic validation is shared between the off-chain infrastructure and the on-chain logic, optimizing both security and cost-efficiency.

3.8 Conclusion

This chapter has detailed the methodological approach for the Multi-chain CDP Protocol. By combining the safety of Rust/Anchor for state management, the standardization of Solidity for asset custody, and the mathematical rigor of the Secp256k1 verification for cross-chain identity, the proposed architecture provides a robust solution to the problem of liquidity fragmentation. The economic stability is underpinned by the Over-collateralized CDP model, while the Guardian infrastructure ensures seamless connectivity. These foundational technologies directly enable the implementation of the system design proposed in the subsequent chapters.

CHAPTER 4. DESIGN, IMPLEMENTATION, AND EVALUATION

4.1 Architecture Design

4.1.1 Overall design

The architectural design of the Multi-chain Stablecoin Protocol is founded upon a Hybrid Event-Driven Microservices framework, structured within a Hub-and-Spoke topology. This sophisticated architecture is engineered to resolve the inherent challenges of liquidity fragmentation and state synchronization across heterogeneous blockchain networks. By strictly decoupling the Asset Custody Layer (residing on EVM chains) from the State Execution Layer (residing on the Solana SVM), the system achieves a high-performance, scalable solution that leverages the distinct advantages of each blockchain environment.

The architecture is composed of three primary execution environments, each functioning as an autonomous system component yet interconnected through a secure event-driven pipeline. The first environment is the EVM Spoke, which hosts the Controller Contract. This contract functions as the user's primary interface and asset vault, designed to be lightweight and gas-efficient. Its responsibilities are strictly limited to asset locking, event emission, and state updates based on authorized callbacks. The second environment is the Solana Hub (SVM), which acts as the system's "Brain." It hosts the Gateway Contract for cryptographic verification and data formatting, and the Main Contract for executing the core business logic, such as managing the Universal Wallet and calculating dynamic Health Factors. The third environment, bridging the deterministic worlds of these blockchains, is the Off-chain Guardian Infrastructure. This middleware operates as an active listener and orchestrator, ensuring that state transitions on one chain are accurately and securely reflected on the other.

To visualize the interaction between these components and the directional flow of data, Figure 4.1 presents the detailed system architecture diagram.

The operational workflow of the system, as depicted in the diagram, follows a rigorous six-step process designed to ensure atomicity, security, and eventual consistency.

Step 1: Initiation and Request Encapsulation The process begins on the EVM chain where the user intends to perform an action, such as depositing collateral. The user constructs a message M containing the action parameters (e.g., Token Address, Amount, Target Chain ID) and a unique nonce. Crucially, the user signs

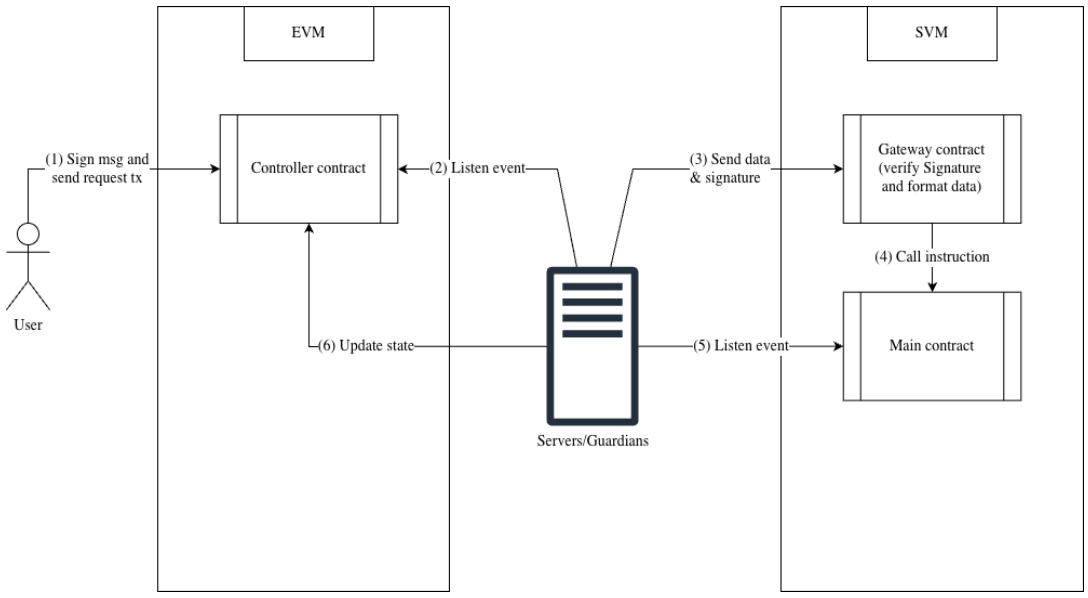

Figure 4.1: Detailed Architecture of the Multi-chain Stablecoin Protocol

this message with their EVM private key, generating a signature σ. The user then submits a transaction to the Controller Contract. Upon receipt, the Controller does not immediately execute the cross-chain logic but performs two critical local actions: it locks the user's assets (in a Vault) and emits a "RequestCreated" event containing the message M and signature σ. This emission acts as a signal flare to the off-chain infrastructure.

Step 2: Event Ingestion (The Listening Phase) Unlike traditional polling mechanisms that can be resource-intensive, the Guardian Server employs a reactive Event Listening model. It maintains an active WebSocket or HTTP connection to the EVM RPC nodes. When the "RequestCreated" event is emitted by the Controller, the Guardian immediately captures the log data. This step represents the "Event" in the Event-Driven Architecture. The Guardian parses the log to extract the raw data necessary for the state transition on the destination chain.

Step 3: Relaying and Submission Once the data is captured, the Guardian packages the original message M and the signature σ into a new transaction payload compatible with the Solana runtime. It then submits this transaction to the Gateway Contract on the SVM. In this phase, the Guardian acts strictly as a courier (Relayer); it does not modify the message content, ensuring that the user's original intent remains tamper-proof. The Guardian also manages the payment of SOL gas fees, abstracting the complexity of holding multiple native tokens from the end-user.

Step 4: Verification and Instruction Execution The Gateway Contract serves as the entry point to the Solana environment. Its primary responsibility is security. Before any business logic is executed, the Gateway invokes Solana's native 'Secp256k1' program to cryptographically verify that the signature σ corresponds to the user's EVM address contained in message M. This verification is performed on-chain, providing a trustless guarantee of identity. Upon successful verification, the Gateway formats the data into a structured Instruction and calls the Main Contract. The Main Contract then executes the core logic, such as creating a Universal Wallet PDA or updating the user's debt position.

Step 5: Outcome Observation Similar to the ingestion phase on the EVM side, the Guardian Server also maintains a listener on the Solana Blockchain. When the Main Contract finishes execution, it emits a specific event - either 'ExecutionSuccess' (indicating the state was updated) or 'ExecutionFailure' (indicating a logic error, such as low health factor). The Guardian listens for these specific outcome events to determine the final status of the cross-chain operation. This asynchronous confirmation step is vital for handling the probabilistic finality of blockchain networks.

Step 6: State Synchronization and Finalization Based on the event received from the Solana Main Contract, the Guardian initiates a final callback transaction to the EVM Controller Contract to close the loop. If the Solana execution was successful, the Guardian calls the 'updateState' function to finalize the process (e.g., minting stablecoins to the user). If the Solana execution failed, the Guardian triggers a rollback mechanism, unlocking the user's assets and resetting their nonce. This ensures that the system maintains data consistency between the Asset Layer and the State Layer, preventing any funds from being permanently locked in transit.

4.1.2 Detailed Package Design

Based on the overall architecture, this section details the internal design of the critical subsystems. The design is visualized using Class Diagrams grouped by their respective execution environments.

a, Solana Hub Package Design

The Solana Hub Package, identified as package Solana Core, functions as the central nervous system of the protocol, tasked with the rigorous validation of incoming cross-chain requests and the execution of complex financial state transitions. As illustrated in Figure 4.2, the package is architected with a strict separation of concerns, hierarchically organized into three distinct layers: entry processing,

core business logic, and persistent state management.

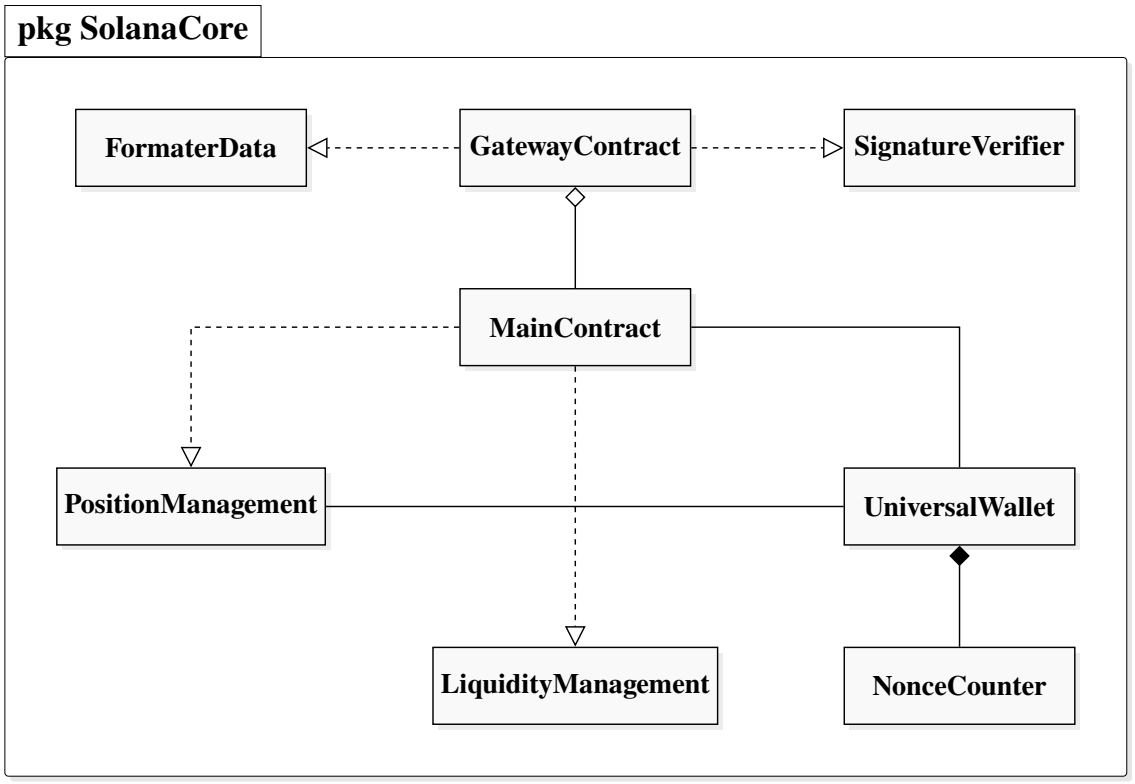

Figure 4.2: Detailed Design of Solana Hub Package

At the apex of this hierarchy sits the Gateway Contract, which serves as the single entry point for all external interactions initiated by the Guardian. This class acts as an orchestrator rather than a calculator; its primary responsibility is to sanitize inputs and enforce security protocols before any state modification occurs. To achieve this, the Gateway maintains dependencies on two specialized utility classes: Formater Data, which handles the deserialization and normalization of raw byte streams from EVM chains, and Signature Verifier, which performs the cryptographic heavy lifting to authenticate user signatures against the secp256k1 curve. Only upon successful verification does the Gateway delegate control to the underlying logic layer, maintaining an aggregation relationship with the Main Contract.

The core operational logic is encapsulated within the Main Contract. To prevent the contract from becoming a monolithic entity, the system employs a modular design pattern where specific domains of responsibility are offloaded to helper modules. The diagram depicts implementation relationships connecting the Main Contract to Position Management, which handles user-centric operations such as collateral locking and debt calculations, and Liquidity Management, which governs system-wide solvency and asset rebalancing. This separation ensures that business

rules regarding user positions are isolated from the mechanisms managing the protocol's aggregate liquidity.

The foundational layer of the package is represented by the Universal Wallet, which serves as the persistent data storage for user profiles. Both the Main Contract and the Position Management module maintain direct associations with this class to read and update financial records. Crucially, the diagram highlights a strict composition relationship between the Universal Wallet and the Nonce Counter. This denotes that the anti-replay mechanism (the nonce) is intrinsically bound to the lifecycle of the wallet; the counter has no independent existence outside the context of its parent wallet, ensuring that transaction ordering is strictly enforced for every unique user identity.

b, Guardian Middleware Package Design

The Guardian Middleware Package, designated as package Guardian Service, constitutes the active off-chain orchestration layer responsible for synchronizing state between the heterogeneous blockchain networks. Designed with an Event-Driven architecture, this package prioritizes reliability and asynchronous processing capability. Figure 4.3 provides a structural overview of the internal components and their interactions, delineating the flow of data from event ingestion to transaction execution.

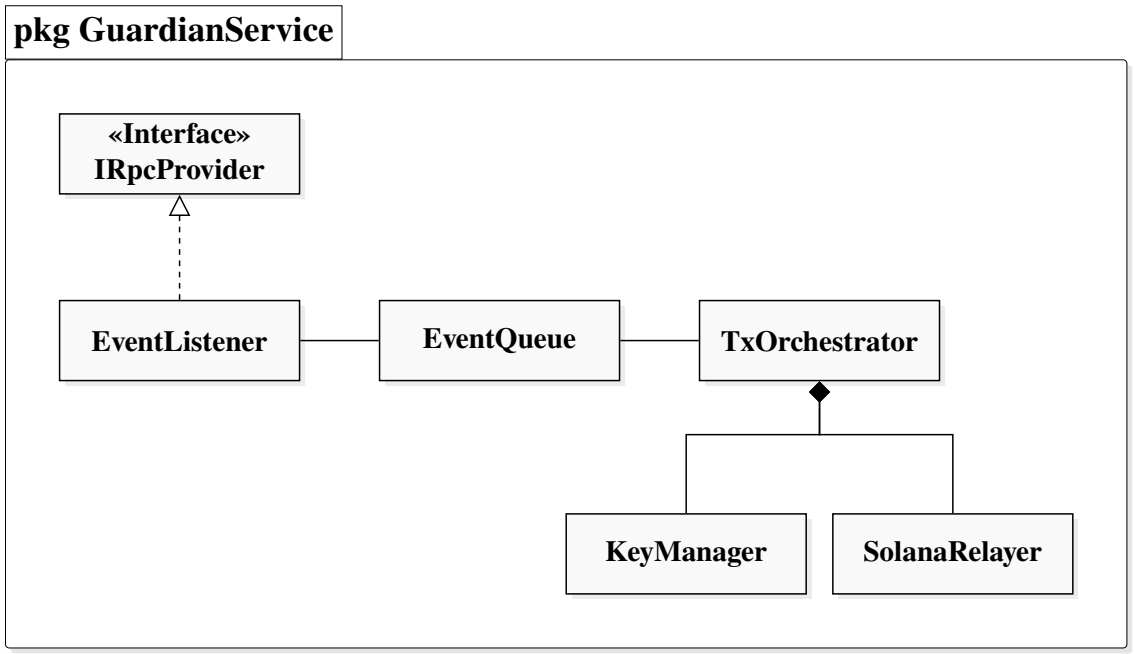

Figure 4.3: Detailed Design of Guardian Middleware Package

The processing pipeline begins on the left side of the diagram with the Event Listener. This component acts as the system's sensory organ, tasked with the con-

tinuous monitoring of blockchain networks. To ensure flexibility and decoupling from specific provider implementations, the listener implements the RPC Provider interface, allowing it to maintain agnostic connections to various EVM chains such as Ethereum, BSC, or Arbitrum. Once a relevant log is detected, the data is passed to the Event Queue. As shown in the center of the diagram, this queue serves as a critical buffer zone, decoupling the high-throughput ingestion layer from the complex processing logic. This design ensures that during periods of network congestion, events are persisted rather than lost, guaranteeing eventual consistency.

The core business logic resides in the Transaction Orchestrator (TxOrchestrator), which consumes normalized events from the queue. This orchestrator is responsible for validating the integrity of the request data and constructing the appropriate cross-chain transaction payloads. To execute its duties securely, the diagram illustrates that the orchestrator maintains strict dependencies on two utility modules. First, it holds a composition relationship with the Key Manager, a security-critical component that manages the Guardian's private keys; this relationship implies that the orchestrator cannot function without the ability to sign transactions. Second, it aggregates the Solana Relayer, utilizing this networking module as a service to broadcast the final signed transactions to the Solana cluster and monitor their confirmation status.

c, EVM Controller Package Design

The EVM Controller Package functions as the primary interface for user interaction within the Ethereum-compatible ecosystem, designed to ensure that every cross-chain request is strictly formatted, authenticated, and processed. Figure 4.4 provides a structural view of this package, visualizing the relationships between the core logic and its supporting modules.

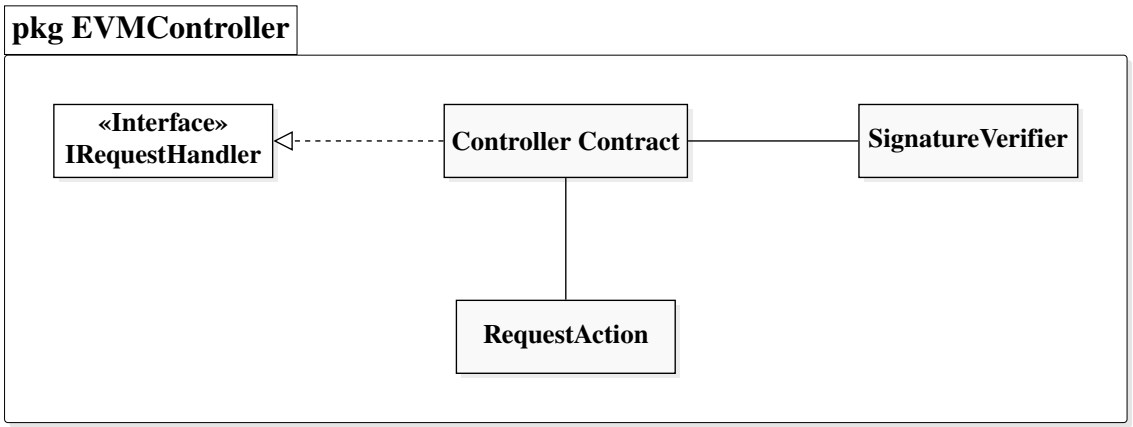

Figure 4.4: Detailed Design of EVM Controller Package

At the heart of the architecture lies the Controller Contract, which acts as the

orchestrator for asset custody and event emission. To ensure modularity and adherence to a standardized communication protocol, the diagram shows that the Controller Contract implements the IRequestHandler interface. This realization relationship guarantees that the contract exposes a consistent set of entry points for external actors, allowing the system to easily swap or upgrade the underlying logic implementation without breaking the interface used by the frontend or the Guardian middleware.

The operational flow of the Controller is supported by two direct associations with utility classes. First, data management is handled through the RequestAction class. The Controller utilizes this component to structure complex input parameters, such as the destination chain ID, token amounts, and action types into a unified data object, ensuring that the payload emitted in event logs is consistent and parseable. Second, and most critically for security, the Controller relies on the SignatureVerifier. Before processing any user request, the Controller delegates the cryptographic validation to this verifier, which confirms that the provided digital signature corresponds to the user's address. This separation of concerns ensures that the core business logic remains uncluttered while maintaining rigorous security checks.

4.2 Detailed Design

This section presents the comprehensive design specification for the Multi-chain Stablecoin Protocol. It decomposes the system into three core layers: Smart Contract Design (On-chain), Server Application Design (Off-chain), and Database Design (Persistence). The level of detail provided herein serves as the blueprint for the implementation phase.

4.2.1 Smart Contract Design

The smart contract architecture serves as the immutable backbone of the Multi-chain Stablecoin Protocol. It is partitioned into two distinct environments: the EVM Controller (handling asset custody and user requests) and the Solana Hub (handling global state and core financial logic). The design focuses on data integrity, cryptographic security, and efficient state synchronization.

a, EVM Controller Design

The Controller contract, deployed on EVM-compatible networks, functions as the primary interface for user interaction and asset custody within the multi-chain ecosystem. Its design philosophy prioritizes security and simplicity, acting as a decentralized vault that locks collateral assets while delegating complex financial calculations to the central hub. The contract is architected to manage the lifecycle

of user requests through a state machine model, ensuring that every cross-chain operation is atomic and reversible.

At the data level, the contract maintains a robust set of structures to track user positions and system integrity. Central to this design is the Request structure, which acts as a comprehensive data carrier. This structure encapsulates all necessary metadata for a transaction, including a unique request identifier, the destination chain identifier, and the specific action type such as deposit or withdrawal. Furthermore, it stores the cryptographic signature generated by the user, which serves as the immutable proof of intent required by the destination chain. To prevent replay attacks and ensure the strict ordering of operations, the contract implements a nonce management system, associating a monotonically increasing counter with each user address.

This is the data structures to manage the lifecycle of user requests:

- Request Struct: This is the fundamental data unit representing a user's intent. It encapsulates all necessary information for cross-chain processing:
  - requestId: A unique identifier for tracking the request across the system.
  - chainId: The ID of the destination chain where the action should be executed.
  - user: The EVM address of the user initiating the transaction.
  - actionType: An enumeration defining the operation (Deposit, Withdraw, Mint, Burn).
  - token: The address of the collateral token involved in the transaction.
  - amount: The quantity of tokens to be processed.
  - nonce: A sequential counter ensuring strict ordering and preventing replay attacks.
  - deadline: A timestamp after which the request is considered invalid, protecting against delayed execution.
  - signature: The cryptographic proof consisting of the components (r, s, v), generated by the user's private key.
- ActionType Enum: A predefined set of constants representing valid operations: Deposit, Withdraw, Mint, and Burn. This strict typing prevents invalid operations from being submitted.
- Link Wallet Request: A specialized structure used when a user wishes to link

their EVM address to a Universal Wallet on a different chain. It stores the binding request until it is verified by the Solana Hub.

A critical aspect of the controller design is the concurrency control mechanism, implemented through a pessimistic locking strategy. When a user initiates a request, the contract enforces a mutex lock on their specific account state. This prevents the user from submitting multiple simultaneous requests, which could lead to race conditions or state desynchronization between the source and destination chains. The lock remains active until the off-chain guardian infrastructure explicitly confirms the finality of the operation on the remote chain. This design choice ensures linearizability, guaranteeing that the system state transitions occur in a predictable and secure sequence.

The operational logic of the contract is exposed through a restricted set of external functions. The primary entry point allows users to submit signed requests, triggering the asset transfer to the vault and the emission of an event log for off-chain listeners. Complementing this is the callback interface, accessible exclusively by the authenticated guardian address. This interface facilitates the finalization of the cross-chain lifecycle. Upon receiving a success signal from the guardian, the contract executes the final state transition, such as minting stablecoins to the user. Conversely, in the event of a remote failure, the contract provides a revert mechanism that releases the mutex lock and refunds the locked assets, ensuring that user funds are never permanently frozen due to network issues.

The contract exposes specific functions for User-System interaction and Guardian-System synchronization:

- Request (User-Facing): This function is the entry point for users. When called, it validates the input parameters, transfers the user's assets to the contract vault (in case of Deposit), and emits an event. Crucially, it locks the user's nonce to prevent concurrent requests, ensuring linearizability.
- Complete Request (Guardian-Only): This restricted function is invoked by the Guardian server upon successful execution on the Solana Hub. It accepts the final status and, if successful, finalizes the local state (e.g., minting stablecoins to the user's wallet).
- Revert Request (Guardian-Only): Serving as a fail-safe mechanism, this function is called if the cross-chain operation fails on Solana (e.g., due to insufficient health factor). It unlocks the user's assets and resets the nonce, returning the system to its pre-request state without loss of funds.

b, Solana Gateway Design

The Gateway contract on the Solana network serves as the secure ingress point for all cross-chain traffic, acting as a cryptographic firewall between the external environment and the internal financial logic. Its primary design objective is to sanitize and authenticate incoming data payloads before they can influence the system's state. Unlike typical smart contracts that focus on business rules, the Gateway is specialized for high-performance data verification, leveraging the parallel processing capabilities of the Solana runtime.

Data ingestion within the Gateway is handled through a specialized storage structure designed to accommodate the heterogeneous data formats of EVM chains. Since Ethereum uses 256-bit integers while Solana native programs are optimized for 64-bit or 128-bit operations, the Gateway implements a data normalization layer. When the off-chain guardian submits a request, the contract parses the raw byte stream, validating the integrity of the data layout and converting the parameters into a format compatible with the system's internal logic. This normalization process ensures that subsequent module calls operate on clean, type-safe data structures, preventing serialization errors deep within the call stack. There is one primary data structure within the Gateway:

- Request Data Storage: The contract defines a specific data layout to store the re-formatted request received from the EVM chain. Unlike the EVM struct, this data is optimized for the Solana BPF runtime (e.g., converting 256-bit integers to 64/128-bit where applicable) to minimize storage costs.

The most critical function of the Gateway is the execution of cryptographic proofs. The contract exposes a specific instruction set that allows the authorized guardian to submit the user's original signature along with the message payload. Instead of implementing complex elliptic curve mathematics in user-space code, which would be prohibitively expensive in terms of compute units, the Gateway utilizes Solana's native Secp256k1 program. By performing a cross-program invocation to this precompiled utility, the contract can efficiently verify that the signature provided was indeed generated by the claimed EVM address. Only upon successful verification does the Gateway permit the control flow to proceed to the Main Contract, thereby establishing a trustless link between the user's identity on Ethereum and their actions on Solana. The Gateway exposes the following key method for secure data handling:

- Set Request (Guardian-Only): This instruction allows the authorized Guardian to submit the raw data and signature from the EVM chain. The method per-

forms the heavy lifting of Secp256k1 signature verification to prove that the data originated from the claimed EVM user. Once verified, it formats the data into an internal instruction and invokes the Main Contract.

c, Solana Main Contract Design

The Main Contract functions as the central nervous system of the protocol, encapsulating the global financial state and executing the core logic of the Collateralized Debt Position mechanism. This component is architected to be the single source of truth for the entire multi-chain ecosystem, aggregating disparate asset data into a unified solvency model. The design prioritizes modularity and state isolation, ensuring that the complex interactions between collateral management, debt issuance, and liquidation are handled with precision and security.

At the heart of the state management strategy is the Universal Wallet, a sophisticated data structure stored as a PDA. This structure aggregates the user's entire portfolio, tracking the collateral balances deposited across various connected chains and the total debt issued by the protocol. By maintaining this global view, the contract can calculate a unified health factor for each user in real-time, allowing for capital efficiency that isolated lending protocols cannot match. Complementing individual user states is the Depository, a global singleton account that tracks system-wide parameters such as the total value locked, the aggregate debt ceiling, and the risk configurations for each supported asset class. The Main Contract defines several critical data structures to manage user positions and system integrity:

- Universal Wallet: The central PDA that aggregates a user's entire portfolio. It stores the total collateral balance and total debt across all connected chains, serving as the single source of truth for solvency calculations.
- Depository: A global state tracking the system-wide parameters, such as total protocol debt, total value locked, and risk parameters (LTV ratios) for each supported asset.
- Loan: A data structure representing the specific debt position of a user. It tracks the principal amount borrowed and the accrued interest over time.
- Link Wallet Request: Stores pending requests for wallet linkage. When a user wants to control their Solana Universal Wallet from a new EVM address, this attribute temporarily holds the request until proof of ownership is established.

The operational logic of the contract is exposed through a set of polymorphic instructions capable of handling requests from diverse origins. Fundamental operations such as deposit, mint, and withdraw are designed to be agnostic to the

caller's source. Whether the instruction originates from a native Solana user or is relayed by the Guardian on behalf of an EVM user, the underlying business logic remains consistent. This unification simplifies the codebase and reduces the attack surface. Additionally, the contract implements a specialized liquidation engine that runs natively on Solana. When a user's position becomes insolvent due to market volatility, this engine allows liquidators to repay debt and seize collateral directly on the hub chain, ensuring that the protocol remains solvent without relying on asynchronous cross-chain calls for critical risk management. The Main Contract exposes the following key methods for user interaction and system maintenance:

- Deposit, Mint, Burn, Withdraw: These are the fundamental CDP operations. They are designed to be polymorphic, capable of being invoked either by the Guardian (relaying an action from an EVM user) or directly by a Solana Native User. This unified interface ensures that the business logic remains consistent regardless of the user's origin chain.
- Interact Universal Wallet (Guardian-Only): A specialized administrative method allowing the Guardian to update the mapping within a Universal Wallet, such as adding a new linked chain address after successful verification.
- Liquidate: This critical function maintains system solvency. It is executed natively on the Solana chain. When a user's Health Factor drops below the threshold, a liquidator can call this method. The liquidator repays the user's debt (in stablecoins) directly on Solana and, in return, receives a portion of the user's collateral stored in the Solana vault (or claims rights to cross-chain collateral via the Depository).

4.2.2 Server Application Design

The Off-chain Infrastructure, architected as the Guardian Middleware, functions as the central nervous system of the protocol, facilitating the secure and reliable transmission of state between the EVM asset layer and the Solana execution layer. Implemented within a Node.js runtime environment, the server is designed not merely as a passive relay, but as an active orchestration engine capable of handling network instability, data verification, and state synchronization. The design of this application is comprehensive, addressing five critical functional domains: network connectivity, data persistence, business logic orchestration, cryptographic security, and transaction finality management.

The first critical function of the server is the robust ingestion of blockchain events through the Network Connectivity and Event Monitoring module. The application initializes and maintains persistent connections to multiple JSON-RPC

providers for each supported chain. This multi-provider strategy is essential to mitigate the risk of single-point failures where a specific node provider might experience downtime or latency spikes. The event listener operates on a polling mechanism that queries the EVM Controller contract for specific logs, such as the request creation events. To ensure data integrity, the listener implements a block confirmation delay strategy, waiting for a configurable number of block confirmations before ingesting an event. This precaution is necessary to protect the system from interacting with reorganized blocks or chain forks, ensuring that the Guardian only acts upon immutable ledger states.

Once an event is securely ingested, the system relies on the Data Persistence and Reliability module to manage the asynchronous processing flow. Recognizing that cross-chain operations involve probabilistic latency, the server avoids in-memory processing which is volatile and prone to data loss during system restarts. Instead, all incoming requests are serialized and pushed into a persistent First-In-First-Out queue backed by the local file system or a dedicated database. This queuing mechanism serves as a buffer that decouples the high-throughput ingestion layer from the transaction submission layer. It allows the system to absorb traffic spikes without overwhelming the Solana RPC endpoints. Furthermore, this module integrates with a Redis key-value store to implement idempotency checks. By caching the unique nonce of every processed request, the system ensures that a specific user action is never executed twice, even if the source chain emits duplicate events due to network retries.

Following the queuing phase, the Business Logic Orchestration module takes responsibility for interpreting and transforming the raw data. Since the EVM and Solana environments utilize different data standards. For instance, the handling of large integers and address formats. This module performs the necessary normalization. It parses the binary payload from the EVM logs, extracting critical parameters such as the user identity, token address, and action type. The logic then maps these parameters to the corresponding Solana-specific data structures defined in the Anchor program IDL. This phase also involves the validation of business rules off-chain, such as checking if the user's request exceeds the system's global debt ceiling or if the requested token is currently supported by the protocol. This precomputation layer reduces the burden on the on-chain smart contracts, ensuring that only structurally valid transactions are attempted.

Integral to the safety of the protocol is the Cryptographic Security and Verification module. Before the Guardian constructs a transaction to update the state on the Solana Hub, it must cryptographically prove that the request originated from

the rightful owner. The server implements a signature verification utility that reconstructs the message hash from the request parameters and performs an elliptic curve recovery on the user's provided signature. This off-chain verification acts as a filter to discard malicious or forged requests immediately, saving the Guardian from paying gas fees for invalid transactions. Additionally, this module manages the Guardian's own sensitive private keys through a secure Key Management System. The signing process is isolated from the logic layer, ensuring that the private keys are never exposed in logs or memory dumps, strictly adhering to the principle of least privilege.

The final functional component is the Transaction Dispatch and State Synchronization module, which closes the feedback loop of the cross-chain interaction. After constructing and signing the Solana transaction, this module broadcasts it to the cluster and enters an observation state. Unlike standard "fire-and-forget" mechanisms, this system implements a sophisticated polling logic to track the transaction's lifecycle until it reaches a finalized status. Upon confirmation of the execution result on Solana, the module constructs a corresponding callback transaction directed back to the EVM Controller. This callback carries the success or failure status, triggering the asset layer to either mint the stablecoins or revert the locked collateral. This bidirectional synchronization ensures that the distributed system maintains eventual consistency, preventing any scenario where user funds remain indefinitely locked in transit due to partial failures.

4.2.3 Database Design

While the blockchain ledger serves as the immutable system of record, the architecture incorporates an off-chain database layer within the Guardian infrastructure. This database functions as a high-performance indexer and state cache, facilitating rapid data retrieval for the frontend interface and supporting complex queries required for the liquidation engine. The schema is designed to mirror the on-chain state, organized into three primary domains: Identity Management, User Position Tracking, and Protocol Liquidity Control.

a, Universal Wallet Collection

The fundamental entity within the database is the Universal Wallet collection, which manages the cross-chain identity mapping. The primary objective of this entity is to prevent identity collisions and enforce the one-to-one relationship between a user's primary EVM address and their Solana PDA. The schema for this entity stores the unique Universal Wallet address derived from the Solana blockchain, acting as the primary key. Associated with this key is the Owner field, recording the

initial EVM address that created the position. To support multi-chain interoperability, the entity includes a Wallets array or relation, listing all secondary addresses from different chains that have been cryptographically linked to this profile. This structure allows the Guardian to instantly verify if an incoming request originates from an authorized address associated with an existing universal profile, thereby preventing duplicate wallet creation attempts. The Universal Wallet collection includes the following key attributes:

- Universal wallet address (Primary Key): The unique Solana address (PDA) generated by the program. This serves as the global identifier for the user's account across the entire system.
- Owner address: The initial EVM address that created the wallet. This field is used to authenticate the root ownership rights.
- Linked wallets: An array or relational table storing all secondary EVM addresses linked to this Universal Wallet. This allows the system to recognize a user interacting from different chains (e.g., Polygon, BSC) as the same entity.
- Creation tx hash: The transaction hash on the source chain that triggered the wallet creation. This serves as an immutable audit trail for the account's origin.
- Status: A status flag (e.g., *Active, Pending Sign, Blacklisted*) used to manage the operational state of the wallet.

b, User Collection (Loans and Requests)

The second critical domain is the User Position and Transaction History, which tracks the financial health and interaction logs of individual users. This domain is essential for both the user dashboard and the automated liquidation bots. The User entity aggregates the financial data, storing the Loan details such as total minted debt and the current composite collateral value. Crucially, it maintains a computed Health Factor field, updated in real-time based on oracle price feeds, which allows the system to query for under-collateralized positions efficiently without scanning the entire blockchain. Linked to the user profile is the Transaction Requests collection. This entity logs every lifecycle event, capturing attributes such as the request ID, action type, token amount, and the associated transaction hashes for both the source and destination chains. This historical log serves as an audit trail, enabling the system to track the status of asynchronous cross-chain operations and detect any stuck or failed requests that require manual intervention. The User Collection encompasses two main sub-entities:

- Loan Position State:

  - Total debt: The aggregate amount of stablecoins minted by the user, denominated in the protocol's base unit.
  - Total collateral value: The real-time USD value of all assets locked by the user across all chains. This is frequently updated by the price oracle workers.
  - Health factor: A computed index derived from the (3.5) formula. This field is indexed to allow liquidator bots to query where Health factor < 1.0 efficiently.
  - Liquidation threshold: The minimum safe ratio required for the specific basket of assets held by the user.

- Request Action History:
  - Request id: A unique composite key (typically chainId + nonce) identifying a specific user action.
  - Action type: Specifies the intent of the transaction (e.g., *Deposit, Withdraw, Mint, Repay*).
  - Source chain id & Dest chain id: Tracks the origin and destination networks of the request.
  - Token address & Amount: Details the asset and quantity involved in the transaction.
  - EVM tx hash: The transaction hash on the EVM side (Locking/Burning assets).
  - Solana tx hash: The corresponding transaction hash on the Solana side (State Update). This links the two asynchronous events together.
  - Process status: Indicates the current lifecycle stage: *Pending, Processing, Completed, Failed, Reverted*.

c, Controller & Liquidity Collection

Finally, the Protocol Controller domain manages the aggregate liquidity and security parameters of the system. This entity tracks the global state of the protocol across all connected spokes. It records the Total Value Locked per chain and the total circulating supply of the stablecoin. The schema includes specific fields for liquidity management, monitoring the available capacity of each EVM vault to ensure that withdrawal requests can be honored. Furthermore, this domain stores system-wide configuration parameters, such as the current Loan-To-Value ratios for supported collateral assets and the operational status of each bridge connection.

By centralizing this data off-chain, the Guardian can perform complex analytics to detect liquidity imbalances or potential security threats, allowing for proactive rebalancing or emergency pausing of specific controller contracts if anomalies are detected. The Controller & Liquidity Collection includes the following critical attributes:

- Chain id: The identifier for the specific blockchain network (e.g., 1 for Ethereum).
- Total value locked: The aggregate amount of collateral assets currently held in the protocol on each specific chain.
- Circulating supply: The total amount of stablecoins minted and currently active in the market from each chain.
- Available liquidity: Monitors the free capital available in the vault. This is crucial for approving withdrawal requests; if a vault is empty, the Guardian must pause withdrawals or trigger a rebalance.

4.3 Application Client and Illustration of main functions

This section demonstrates the implementation of the client-side application, focusing on the user experience flows for identity management and cross-chain financial operations.

4.3.1 User Interface for Universal Wallet

The Universal Wallet interface serves as the foundational layer of the application, enabling users to aggregate their fragmented blockchain identities into a single manageable profile. This interface handles the complexities of multi-chain authentication and state synchronization.

a, Wallet Connectivity

The initial interaction with the protocol begins with the wallet connection module. To support the hybrid architecture, the interface is designed to accommodate distinct blockchain standards simultaneously. The connectivity panel categorizes providers into specific network groups, allowing users to select between Solana-native wallets like Phantom or EVM-compatible wallets such as MetaMask. A distinguishing feature of this implementation is the support for concurrent multi-wallet connections. Users can maintain active sessions with multiple providers at the same time, which is visualized by the status indicators next to each provider. This capability is essential for the protocol, as it allows the application to read balances and request signatures from different chains without forcing the user to constantly switch the active network in their browser extension. Figure 4.5 illustrates the wallet connectivity interface, showcasing the seamless integration of various

wallet types and networks.

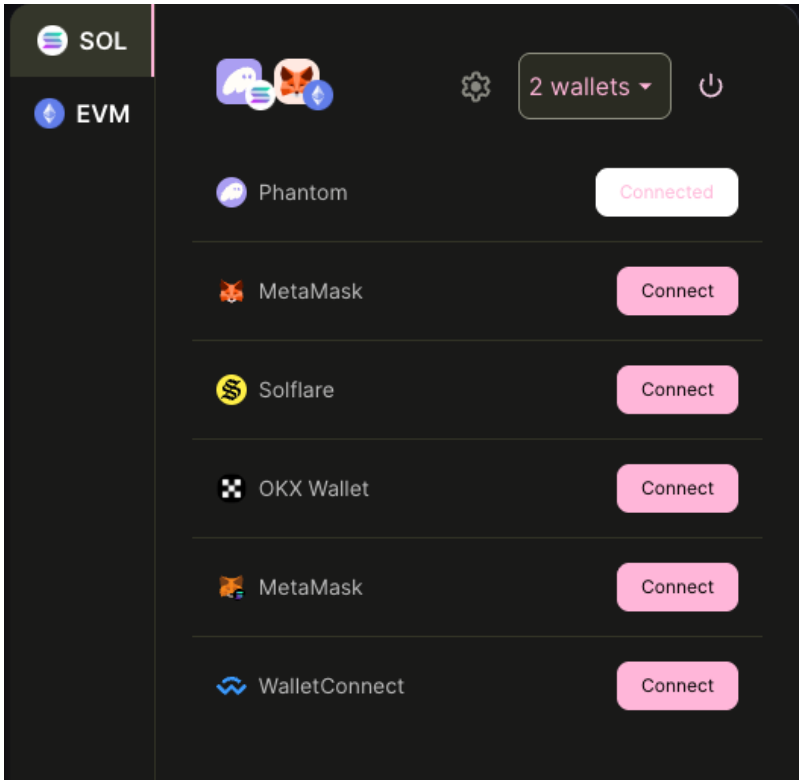

Figure 4.5: Multi-chain Wallet Connectivity Interface

b, Link Wallet Workflow

Once the wallets are connected, the user interacts with the linking interface to establish the Universal Wallet. The design presents a dual-field form requiring the selection of a source wallet and a destination wallet. The application enforces a rigorous proof-of-ownership protocol during this process. The workflow initiates when the user selects the request option, triggering a transaction on the source chain. This on-chain action serves as the intent declaration. Upon successful confirmation of the source transaction, the interface automatically prompts the user to sign a cryptographically secure message using the destination wallet. This two-step verification ensures that the link is established only when the user possesses the private keys for both addresses, effectively binding the identities across the two networks.

There is one primary screen for the Universal Wallet, as shown in Figure 4.6. The interface is divided into two main panels: the wallet connectivity panel on the left and the wallet management panel on the right.

c, Remove Wallet Logic

The management of associated addresses is handled through the wallet list panel, where users can view and disassociate their linked accounts. The interface provides

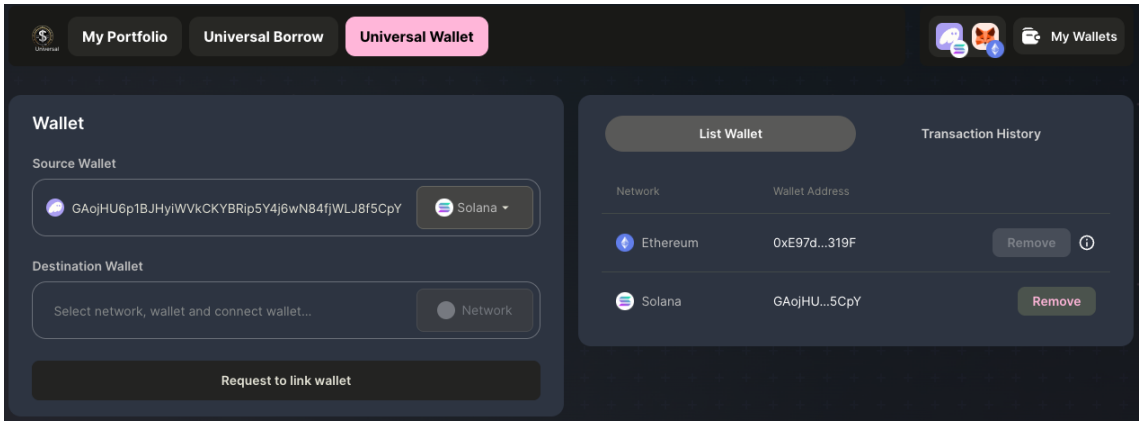

Figure 4.6: Universal Wallet Management Interface (Link and List View)

a removal function for each entry; however, the execution of this action is governed by strict logic to preserve system integrity. The protocol distinguishes between the primary wallet, which acts as the root identifier, and secondary wallets. The interface restricts the removal of the first wallet, enforcing a rule that it can only be dissociated after all secondary wallets have been removed. Additionally, the system performs a solvency check before allowing any removal operation. If the user has outstanding debt obligations, the interface prevents the removal of any wallet that contains collateral necessary to maintain a safe health factor, thereby securing the protocol against under-collateralization risks.

4.3.2 User Interface for Cross-chain Borrow

The borrowing interface functions as the central command center for the user's financial activities, designed to provide immediate visual feedback on solvency while offering granular control over asset allocation across disparate chains. As illustrated in Figure 4.7, the dashboard aggregates the user's global financial position at the top level into two primary metrics: Total Deposited and Total Minted. Unlike single-chain applications that reflect only local balances, these values represent the summation of assets across all connected networks—such as Ethereum, Solana, and Arbitrum—normalized to a USD base currency. This high-level summary provides users with an instant snapshot of their portfolio's scale and outstanding liabilities, essential for informed borrowing decisions, while the current Borrow Rate is prominently displayed to ensure transparency regarding the cost of debt.

Core interactions are facilitated through dual-pane action modules dedicated to Deposit and Mint operations. The Deposit module enables users to lock collateral via a chain-agnostic token selector, allowing assets like USDC or ETH to be deposited from any supported network without leaving the dashboard. Si-

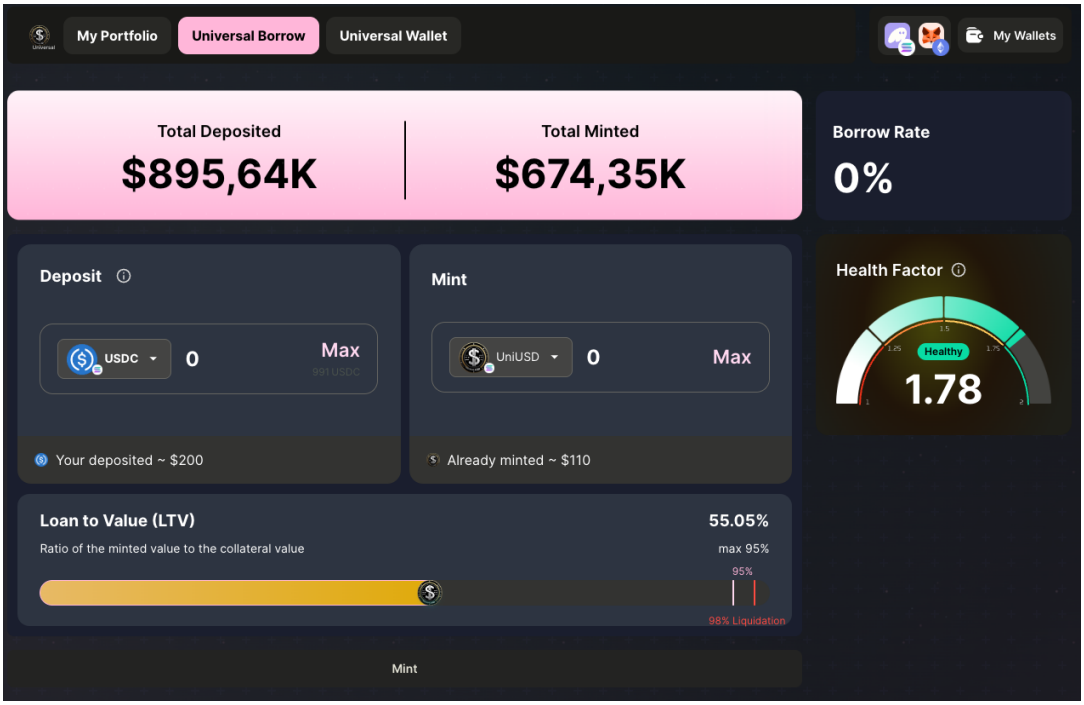

Figure 4.7: Cross-chain Borrowing and Position Management Interface

multaneously, the Mint module permits the generation of the protocol's stablecoin, UniUSD. These components are tightly coupled; as the user inputs a deposit amount, the interface dynamically updates the borrowing capacity in the mint section, providing real-time feedback on how much debt can be safely issued against the new collateral.

To mitigate the inherent risks of liquidation, the interface incorporates advanced visualization tools specifically designed for proactive risk management. A distinctive feature is the interactive Loan-To-Value (LTV) slider bar, which visualizes the ratio between the minted debt value and the underlying collateral value. As users adjust their minting amount, the progress bar dynamically fills towards the critical liquidation threshold marked at 98%, intuitively alerting users to their risk level relative to the protocol's maximum allowable limit. Complementing this is the Health Factor gauge, a semi-circular display that categorizes health status into color-coded zones: red for danger, yellow for caution, and green for healthy positions. This gauge updates instantaneously with user input, allowing for the simulation of transaction impacts on solvency prior to blockchain commitment, thereby fostering safer financial behavior.

4.3.3 User Interface for Portfolio

The Portfolio interface is engineered to provide a comprehensive, real-time overview of the user's active financial engagements within the protocol. Unlike the borrowing interface which focuses on the initiation of new actions, the portfolio

view is strictly dedicated to the ongoing management and monitoring of established debt positions. As illustrated in Figure 4.8, the layout is divided into logical zones covering position visualization, risk assessment, and operational control.

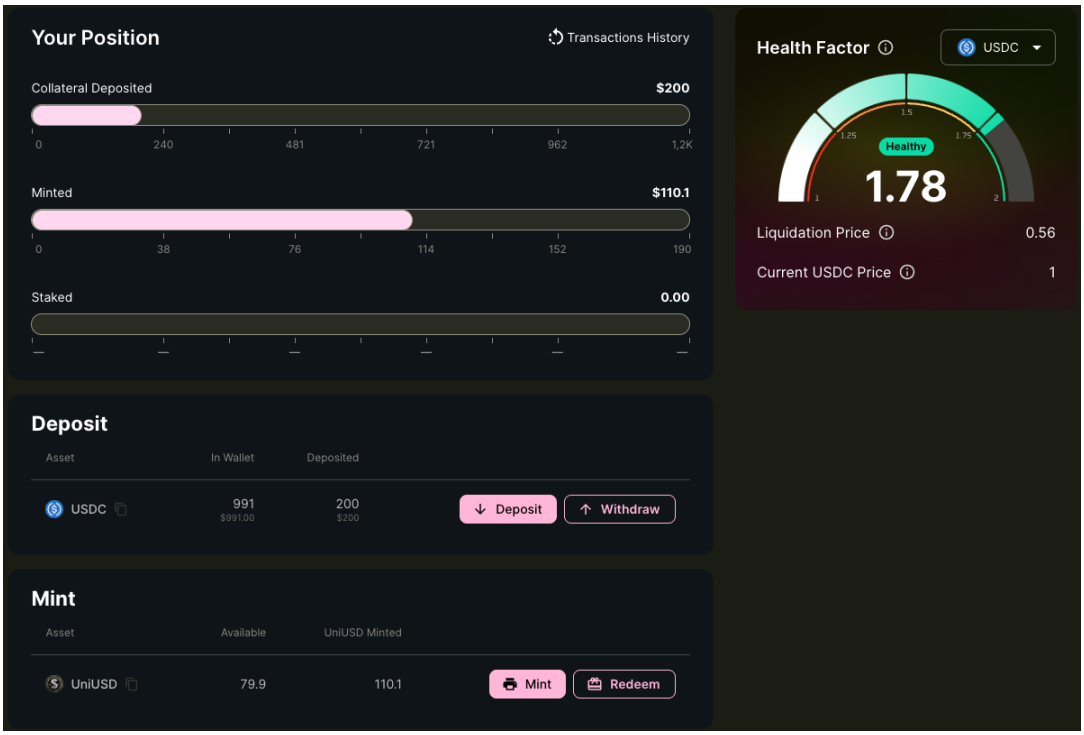

Figure 4.8: User Portfolio and Position Management Interface

The "Your Position" panel serves as the visual anchor of the dashboard, offering a granular breakdown of capital allocation. It utilizes horizontal progress bars to represent the magnitude of assets relative to the user's total capacity. Specifically, the "Collateral Deposited" bar displays the total value of assets locked in the protocol, providing a quick visual reference for capital commitment, while the "Minted Debt" bar indicates the outstanding stablecoin liability. By placing these metrics in close proximity, the interface allows users to effortlessly evaluate their leverage ratio. Additionally, a dedicated section tracks yield-bearing activities such as Staked Assets, ensuring that all forms of capital deployment are centralized within a single view.

To the right of the position summary, the Health Factor gauge functions as a persistent safety monitor. In the portfolio context, this gauge is augmented with critical market data essential for risk assessment, most notably the Liquidation Price. This metric calculates the specific price point of the collateral asset at which the position would become insolvent. For example, a liquidation price of 0.04 against a current market price of 1 indicates a highly secure position. Real-time price updates from the Oracle are displayed alongside these figures, allowing users to benchmark market movements against their liquidation thresholds instantly.

Finally, the lower section of the portfolio interface integrates operational controls directly into the asset list to facilitate rapid decision-making. This design pattern minimizes navigation friction by embedding functions such as Deposit and Withdraw within the collateral asset rows, and Mint and Redeem within the stablecoin sections. By displaying the user's wallet balance and available borrowing power adjacent to these controls, the interface enables users to react swiftly to market changes strengthening their health factor or capturing profit without navigating away from the monitoring dashboard.

4.4 Application Building

This section details the technological ecosystem utilized to construct the Multichain Stablecoin Protocol. The development stack is categorized into development environments, smart contract frameworks, frontend interfaces, backend infrastructure, and testing suites.

4.4.1 Libraries and Tools

a, IDE & Extensions

The development environment is centered around Visual Studio Code, augmented with specific extensions to support syntax highlighting, linting, and formatting for the diverse languages used (Rust, Solidity, TypeScript, Python).

Tool / Extension	**Purpose**	**URL**
Visual Studio Code	Primary Integrated Development Environment.	`https://code.visualstudio.com/`
Rust-analyzer	Language support for Rust (code completion, goto definition).	`https://github.com/rust-lang/rust-analyzer`
Solidity (Juan Blanco)	Syntax highlighting and snippets for Ethereum smart contracts.	`https://github.com/juanfranblanco/vscode-solidity`
Tenderly Extension	Integration with Tenderly for smart contract monitoring and debugging.	`https://tenderly.co/`
Solana-cli	Command-line tools for Solana program deployment and management.	`https://docs.solana.com/cli`

Table 4.1: Development Environment Tools

b, Smart Contract Tools & Libraries

This category encompasses the core frameworks required to develop, compile, and deploy logic on both the Solana Virtual Machine (SVM) and the Ethereum Virtual Machine (EVM).

Library / Framework	**Purpose**	**URL**
Rust (v1.79.0)	Systems programming language for Solana programs.	`https://www.rust-lang.org/`
Anchor Framework (v0.30.0)	Sealevel runtime framework for Solana, handling serialization/IDL.	`https://www.anchor-lang.com/`
Solidity (v0.8.28)	Object-oriented language for EVM Controller contracts.	`https://soliditylang.org/`
OpenZeppelin Contracts	Standard secure contract components (ERC20, Ownable).	`https://www.openzeppelin.com/`
Hardhat (v2.25.0)	Ethereum development environment for compiling and deployment.	`https://hardhat.org/`

Table 4.2: Smart Contract Development Stack

c, Frontend Client

The user interface is engineered using a high-performance React stack powered by Vite. The integration of blockchain interactions relies on modern libraries like Viem/Wagmi for EVM and the Solana Web3 suite for SVM.

Library (Version)	**Purpose / Usage**	**URL**
React (v18.3.1)	Component-based UI library for building the application interface.	`https://react.dev/`
TypeScript (∼5.6.2)	Strictly typed superset of JavaScript ensuring type safety across the codebase.	`https://www.typescriptlang.org/`
Vite (v6.0.5)	Next-generation frontend build tool providing fast HMR and optimized bundling.	`https://vitejs.dev/`
Wagmi (v2.16.9)	React Hooks library for Ethereum, simplifying wallet connection and state management.	`https://wagmi.sh/`
Viem (v2.x)	Low-level TypeScript interface for Ethereum, replacing Ethers.js for better performance.	`https://viem.sh/`
@solana/web3.js (v1.98.0)	Core library for interacting with the Solana blockchain (RPC, Transactions).	`https://solana.com/docs`
@coral-xyz/anchor (v0.30.1)	Client library to interact with Anchor-based Solana programs (IDL, Accounts).	`https://www.anchor-lang.com/`
@solana/spl-token (v0.4.12)	Utility library for managing SPL tokens (minting, transferring) on Solana.	`https://spl.solana.com/token`
Jotai (v2.11.0)	Primitive and flexible state management library for React (Global State).	`https://jotai.org/`

Table 4.3: Frontend Development Libraries

d, Backend (Guardian Infrastructure)

The Guardian infrastructure operates on a Python 3.9 runtime, utilizing asynchronous frameworks to handle high-throughput event processing. The core libraries facilitate interaction with both Ethereum and Solana networks alongside persistent storage.

Library (Version)	Purpose / Usage	URL
Python (v3.9)	Runtime environment for the backend logic and scripting.	https://www.python.org/
Sanic ($\geq$ v22.12.0)	High-performance asynchronous web server and framework for building fast APIs.	https://sanic.dev/
Web3.py (v6.15.1)	Comprehensive Python library for interacting with Ethereum nodes and contracts.	https://web3py.readthedocs.io/
Eth-account (v0.11.3)	Library for signing transactions and managing Ethereum accounts securely.	https://eth-account.readthedocs.io/
Solana.py (v0.36.6)	Python client for interacting with the Solana blockchain JSON RPC API.	https://michaelhly.github.io/solana-py/
Solders (v0.26.0)	High-performance Python binding for Solana primitives (Keypairs, Pubkeys).	https://github.com/kevinheavey/solders
AnchorPy (v0.21.0)	Python client for Anchor-based Solana programs, enabling IDL interaction.	https://kevinheavey.github.io/anchorpy/
PyMongo (v4.10.1)	Synchronous driver for MongoDB, used for persisting user request logs.	https://pymongo.readthedocs.io/

Table 4.4: Backend Development Libraries

e, Testing Tools

Quality assurance is maintained through a combination of unit testing frameworks and local blockchain simulators.

Tool	Purpose	URL
Mocha / Chai	JavaScript test framework and assertion library.	https://mochajs.org/
Hardhat	Local Ethereum node for forking mainnet state during tests.	https://hardhat.org/

Table 4.5: Testing and Simulation Tools

4.4.2 Achievement

The development phase culminated in a fully operational Cross-chain Stablecoin Protocol, demonstrating the feasibility of the Hub-and-Spoke architecture for decentralized finance. The project successfully delivered three distinct yet integrated components, each fulfilling a critical role in the system:

(i) The Multi-chain DApp Client: A unified interface that abstracts the complexity of cross-chain interactions. It allows users to manage their Universal Wallets and execute financial operations seamlessly across heterogeneous networks without manually bridging assets.

(ii) The Hybrid Smart Contract System:

- The Solana Hub successfully manages the global financial state, proving that a high-performance chain can serve as the settlement layer for assets on slower chains.
- The EVM Controllers function effectively as decentralized vaults, ensuring secure asset custody with minimized gas costs.

(iii) The Guardian Infrastructure: The Python-based middleware proved robust in synchronizing state. It successfully handles event ingestion, cryptographic verification, and transaction relaying, ensuring eventual consistency between the EVM and SVM environments.

a, Project Statistics

The scale and complexity of the implementation are reflected in the codebase metrics. Table 4.6 provides a detailed breakdown of the lines of code (LOC) across different modules.

Component	**Language / Technology**	**Lines of Code (LOC)**
Frontend Client	TypeScript	42,355
	TypeScript JSX (React Components)	19,174
Smart Contracts (SVM)	Rust (Anchor Framework)	10,633
	TypeScript (Integration Tests)	$\approx$ 34,000
Smart Contracts (EVM)	Solidity	733
Backend (Guardian)	Python	$\approx$ 36,000
Total Project Size	$\approx$ **142,895 LOC**	

Table 4.6: Source Code Statistics by Component

4.5 Testing

Ensuring the reliability and security of smart contracts is the paramount objective of the testing phase, particularly for a DeFi protocol handling user assets across multiple blockchains. The testing strategy employed for this project is comprehensive, moving from atomic unit tests of individual functions to complex integration scenarios that simulate the entire cross-chain lifecycle. This section details the methodologies used, the specific test cases designed for critical functionalities, and the final evaluation of the system's robustness.

4.5.1 Testing Methodology

The testing framework leverages a combination of industry-standard techniques to maximize code coverage and vulnerability detection:

- Unit Testing: This is the first line of defense. Each function within the Solana Main Contract (Rust) and EVM Controller (Solidity) is tested in isolation. We utilized the *Anchor* framework's testing suite (TypeScript) to verify Solana logic and *Hardhat/Chai* for EVM logic. The goal is to ensure that mathematical calculations (e.g., Health Factor) and state transitions occur exactly as specified.
- Integration Testing (End-to-End): Given the hybrid architecture, unit tests alone are insufficient. We deployed the cross-chain environment using a testnet. These tests involve the Guardian's role to verify the interaction flow: *User locks on EVM* $\rightarrow$ *Event Emitted* $\rightarrow$ *Relayer submits to Solana* $\rightarrow$ *State Updated*.

4.5.2 Test Cases for Critical Functions

The testing focus was prioritized on three high-risk areas: Cross-chain Minting (Solvency), Cryptographic Verification (Security), and Liquidation (System Health).

a, Function 1: Cross-chain Lending Operations (Deposit, Mint, Withdraw, Redeem)

This function is the core value proposition of the protocol. It involves state changes on both chains and requires strict solvency checks.

Test Scenario & Input	Detailed Procedure & Expected Result	Status
Scenario: User deposits valid collateral and mints within LTV. **Input:** - Collateral: 100 USDC - Mint: 95 UniUSD - LTV Limit: 95%	**1.** User calls requestAction on EVM. **2.** Check EVM Vault balance: increases by 100 USDC. **3.** Mock Guardian submits valid signature to Solana. **4. Expected:** Solana state updates UniversalWallet debt to 95. Health Factor remains > 1.0. Transaction succeeds.	Passed
Scenario: User attempts to mint debt exceeding the collateral value. **Input:** - Collateral: 100 USDC - Mint: 100 UniUSD - Max LTV: 95%	**1.** Construct request on EVM (this passes as EVM doesn't check price). **2.** Guardian relays to Solana. **3.** Solana contract calculates projected Health Factor. **4. Expected:** Calculation shows $HF < 1.0$. Transaction on Solana reverts with an error.	Passed
Scenario: User sends a request with 0 amount. **Input:** Amount = 0	**1.** Call requestAction. **2. Expected:** EVM Contract reverts immediately with InvalidAmount error to save gas and prevent spam.	Passed
Scenario: User redeem and withdraw max valid collateral amount. **Input:** - Withdraw collateral: 100 USDC - Redeem: 95 UniUSD - New LTV: 0%	**1.** User calls requestAction on EVM. **2.** Check EVM Vault balance: decreases by 100 USDC. **3.** Mock Guardian submits valid signature to Solana. **4. Expected:** Solana state updates UniversalWallet debt to zero and collateral to zero. Transaction succeeds.	Passed
Scenario: User withdraw exceeds collateral allowed **Input:** - Withdraw collateral: 100 USDC - New LTV: 100%	**1.** User calls requestAction on EVM (this passes as EVM doesn't check LTV). **2.** Guarndian relays to Solana. **3.** Solana contract calculates projected LTV. **4. Expected:** Calculation shows $LTV > MaxLTV$. Transaction on Solana reverts with an error.	Passed

b, Function 2: Cryptographic Security (Signature Verification)

Since the Solana state is updated based on messages relayed by an off-chain server, verifying that the original user actually signed the message is critical to prevent spoofing.

Test Scenario & Input	Detailed Procedure & Expected Result	Status
Scenario: Guardian submits a payload with a valid Secp256k1 signature. **Input:** - Msg Hash: $H(M)$ - Sig: Sign($User_{priv\_key}$, $H(M)$)	**1.** Solana Gateway invokes native Secp256k1 to recover address. **2.** Recovered address is compared with each wallet in an Universal Wallet . **3. Expected:** An address matches. Execution proceeds to Main Contract.	Passed
Scenario: Attacker (or compromised Guardian) tries to mint debt for a Victim using Attacker's signature. **Input:** - Msg: "Mint for Victim" - Sig: Sign($Attacker_{priv}$, Msg)	**1.** Gateway recovers the signer address from the signature. **2.** Contract compares Recovered ($Attacker$) vs Wallet Owner ($Victim$). **3. Expected:** Assertion fails. Transaction reverts with `InvalidSignature`.	Passed
Scenario: Attacker resubmits a previously valid Mint request to double the debt. **Input:** A valid payload (M, σ) used in block N.	**1.** First submission succeeds; nonce of Universal Wallet increments from k to $k+1$. **2.** Second submission sends same payload (nonce k). **3. Expected:** Contract checks $Payload.nonce(k) == Wallet.nonce(k+1)$ -> fails. Revert with invalid nonce.	Passed

Table 4.7: Test Cases for Security and Verification

c, Function 3: Liquidation Engine

This module ensures the protocol remains solvent by allowing third parties to liquidate bad debt. Testing this requires manipulating price feeds.

Test Scenario & Input	Detailed Procedure & Expected Result	Status
Scenario: Oracle price drops, making user insolvent. **Input:** - Collateral: 1 ETH - Debt: 1500 USD - Price drops: $2000 \rightarrow 1600$	**1.** Mock Oracle updates price to \$1600. Health Factor drops below 1.0. **2.** Liquidator calls liquidate function. **3. Expected:** Liquidator pays debt. User's collateral is transferred to Liquidator + Bonus. User's debt is reduced.	Passed
Scenario: Liquidator tries to liquidate a healthy user for profit. **Input:** Health Factor = 1.5	**1.** Call liquidate function. **2. Expected:** Contract verifies $HF > Threshold$. Reverts with debt is healthy. No assets are transferred.	Passed

Table 4.8: Test Cases for Liquidation Logic

4.5.3 Evaluation and Results

The comprehensive testing campaign has yielded highly positive results, serving as a rigorous validation of the architectural decisions made during the design phase. The evaluation process was not merely a check-box exercise but a stress test of the system's resilience under adversarial conditions. Quantitatively, the project executed a total of 12 distinct test suites. These tests spanned the entire stack, from the EVM Vault logic to the Solana state management and the Guardian middleware. The code coverage metrics are particularly indicative of the system's robustness: the Solidity contracts on the EVM side achieved 100% branch coverage, a feasible target due to the minimal logic design pattern. On the Solana side, the Rust programs achieved a 92% statement coverage, with the remaining uncovered paths primarily relating to unreachable panic states that are theoretically impossible to trigger under normal protocol operation.

Qualitatively, the integration tests confirmed the efficacy of the hybrid security model. The most significant finding was the robustness of the Mutex locking mechanism implemented on the EVM Controller. During simulated network latency scenarios where the Guardian node was forced offline for extended periods, the system maintained its integrity; user funds remained securely locked, and no race conditions occurred when the Guardian resumed operation to process the backlog. This proves that the system effectively adheres to the principle of "Safety over Liveness," prioritizing fund security even at the cost of temporary service unavailability during infrastructure outages.

Furthermore, the testing phase uncovered a critical technical challenge regarding cross-chain arithmetic precision. Early iterations of the protocol revealed discrepancies when transferring value between Ethereum (which uses 18 decimal places) and Solana (which typically uses 9 decimal places for native tokens). This mismatch initially led to rounding errors that, while microscopic per transaction, could accumulate to significant accounting imbalances over time. This issue was rectified by implementing a dedicated Decimal Scaling function within the Gateway contract, which standardizes all values to a common high-precision internal format before execution. This solution was subsequently verified by specific boundary value test cases, ensuring that the protocol remains mathematically consistent regardless of the underlying chain's token standards.

4.6 Deployment

The deployment phase marks the transition of the Multi-chain Stablecoin Protocol from a local development environment to a live, distributed network accessible

to the public. To validate the system's performance under realistic network conditions, including latency, gas costs, and block finality times. The application was deployed across a heterogeneous testnet environment that mirrors the topology of the mainnet production target.

The blockchain infrastructure is deployed on two distinct networks to simulate the cross-chain architecture. The asset layer, responsible for custody and user interaction, is deployed on the Ethereum Sepolia Testnet. This network was selected for its stability and the availability of robust RPC endpoints, providing an accurate simulation of the Ethereum Mainnet environment. Conversely, the state execution layer is hosted on the Solana Devnet. This choice allows for the testing of high-throughput transaction processing and the validation of the Anchor program's performance without incurring real financial costs. The smart contracts on both chains were verified on their respective block explorers (Etherscan and Solana Explorer) to ensure transparency and facilitate public auditing during the testing phase.

The infrastructure is distributed across two distinct environments to simulate real-world cross-chain conditions:

- Asset Layer: Deployed on the Ethereum Sepolia Testnet. This environment hosts the Controller Contract responsible for locking user assets and managing mutex locks.
- State Layer: Deployed on the Solana Devnet. This environment hosts the high-performance logic, including the Gateway for signature verification and the Main Contract for global state management.

Table 4.9 provides the specific on-chain addresses for the deployed components, enabling public verification of the protocol's code and transaction history.

Component	**Network**	**Contract Address**
Controller Contract	Ethereum Sepolia	`0x93BdF8c1e6a18D3e73280d886f1141755CDeCde5`
Main Contract	Solana Devnet	`vdstH476bZ8hdb8TvFS5bsn1RPBVuczpYQyBRgAceHq`
Gateway Contract	Solana Devnet	`9oRzAwX1a31LDUrxdZDdMMGxcPe4v2cdUwET8UbKP826`

Table 4.9: Deployed Smart Contract Addresses

CHAPTER 5. SOLUTION AND CONTRIBUTION

5.1 Introduction

The research and development of the Multi-chain Stablecoin Protocol have culminated in a series of architectural and algorithmic innovations designed to address the fundamental limitations of current Decentralized Finance (DeFi) infrastructure. Throughout the implementation of the Collateralized Debt Position (CDP) mechanism across heterogeneous blockchain networks, several critical challenges emerged, ranging from liquidity fragmentation and cryptographic incompatibility to asynchronous state inconsistency. This chapter details the specific solutions devised to overcome these hurdles. These contributions constitute the core intellectual value of the thesis, demonstrating a novel approach to cross-chain interoperability that prioritizes security, capital efficiency, and user experience over traditional bridging methods.

5.2 The State Orchestration Architecture via Hub-and-Spoke Model

5.2.1 Problem Identification

The current decentralized finance (DeFi) landscape is plagued by structural inefficiencies that hinder mass adoption and capital optimization. Through the analysis of existing multi-chain protocols, three critical problems have been identified:

First, there is a severe fragmentation of state data across isolated blockchain networks. In the current paradigm, a user's financial position on Ethereum is completely invisible to applications on Arbitrum or Solana. This isolation forces users to manage disjointed portfolios, making it impossible to leverage assets on one chain to support a position on another without physically moving the tokens. This lack of interoperability results in a rigid and inflexible user experience where managing solvency across multiple chains becomes a manual and error-prone process.

Second, the integration of traditional bridging solutions into DApps introduces significant friction. Standard bridges often suffer from high latency and exorbitant gas fees, as they require complex on-chain verification steps for every transfer. For a user who simply wants to use their ETH collateral to borrow stablecoins on a faster chain, the cost and time delay of bridging can negate the economic benefits of the transaction.

Third, and perhaps most critically, the traditional lock-and-mint bridge model results in a massive inefficiency of capital. In these systems, liquidity is often statically locked in bridge contracts on both the source and destination chains to

maintain the peg of wrapped assets. This idle liquidity represents a "dead weight" in the ecosystem; billions of dollars are trapped in smart contracts without generating yield or contributing to the broader market depth, leading to a suboptimal utilization of the total value locked (TVL).

5.2.2 Proposed Solution

To address these challenges, this thesis implements a State Orchestration Architecture utilizing a Hub-and-Spoke topology. This model fundamentally shifts the focus from bridging assets to synchronizing state.

The primary advantage of this solution is its scalability. By designating Solana as the central Hub and EVM chains as Spokes, the system allows for the seamless addition of new blockchain networks without significant resource expenditure. Adding support for a new chain (e.g., Base or BNB) does not require deploying a complex mesh of bridges; it simply involves deploying a lightweight Controller Contract that reports to the central Hub. This makes the protocol highly adaptable to the rapidly evolving blockchain landscape.

Regarding capital efficiency, the architecture unlocks the potential of the otherwise idle collateral. Since the assets are locked in the protocol's native vaults on the source chains (Spokes) rather than being wrapped and fragmented, the protocol can implement strategies to rehypothecate this liquidity. The locked assets can be securely deployed into low-risk farming protocols (e.g., Aave or Compound) on their native chains to generate yield. This revenue stream can then be distributed back to the users or used to subsidize the borrowing rates, transforming the passive "lock" into an active, yield-bearing position.

Finally, the architecture provides a unified interface for position management. The Hub maintains a "Universal Wallet" that aggregates the user's collateral and debt from all connected Spokes into a single global state. This allows users to view and control their entire financial health from one dashboard. A user can deposit ETH on Ethereum and immediately see their borrowing power increase on Solana, enabling a fluid and cohesive cross-chain experience that was previously unattainable.

5.3 Cross-chain Identity Verification and Request Integrity

5.3.1 Problem Identification

Establishing a secure and unified user identity across heterogeneous blockchain networks presents a multi-faceted challenge that encompasses both logical association and technical verification. During the implementation phase, three distinct

hurdles emerged regarding identity management and data integrity.

First, there is an inherent difficulty in linking identities across isolated ledgers. In a decentralized environment without a centralized Know-Your-Customer (KYC) database, an Ethereum address and a Solana address are cryptographically distinct entities with no intrinsic mathematical relationship. Determining whether two wallets on different chains belong to the same physical user is impossible through simple observation. Without a robust binding mechanism, the system cannot safely aggregate collateral from one chain to back debt on another, as it cannot verify the common ownership of assets.

Second, the reliance on an off-chain infrastructure (the Guardian) introduces a "Man-in-the-Middle" risk regarding data integrity. Since the Guardian is responsible for relaying the user's intent from the EVM chain to the Solana Hub, there is a theoretical risk that a compromised or malicious Guardian could tamper with the request payload.For example, changing the destination address of a minting request before submitting it to the Solana contract. The system must ensure that the message processed on the destination chain is exactly identical to the message authorized by the user on the source chain, regardless of the relayer's honesty.

Third, this verification process is technically complicated by cryptographic incompatibility. The Ethereum ecosystem relies on the Secp256k1 elliptic curve for digital signatures, whereas Solana natively utilizes the Ed25519 curve. This mismatch means that a standard Solana smart contract cannot natively verify an Ethereum signature to authenticate a user's request. Developing a custom verification algorithm in user-space code (e.g., in Rust/Anchor) is computationally prohibitive and would likely exceed the transaction compute budget, rendering the solution unscalable.

5.3.2 Proposed Solution

To overcome these obstacles, this thesis proposes a Cryptographically Bound Identity Protocol that leverages a hybrid verification model to ensure both identity linkage and request integrity.

To address the identity linking problem, the solution implements a "Handshake Protocol." The user is required to explicitly authorize the linkage by signing a binding request with their destination wallet (Solana) that references their source wallet (Ethereum). This creates a bidirectional cryptographic proof of ownership. Once verified, the Solana Hub stores this relationship in the Universal Wallet state. This effectively treats the Ethereum address as an authorized "signer" or "controller" for the Solana-based position, allowing the system to logically group disparate ad-

dresses under a single user profile.

Regarding data integrity and the Guardian's trust assumption, the solution adopts a "Verify-on-Chain" strategy. The Guardian is treated strictly as an untrusted courier. When a user initiates a request, they sign the hash of the payload (including amount, nonce, and action type) using their private key. The Guardian transmits both the raw payload and the user's signature to Solana. The Solana contract reconstructs the hash from the raw payload and verifies it against the signature. If the Guardian attempts to tamper with even a single byte of the payload, the hash will change, and the signature verification will fail. This ensures that the Guardian cannot forge or modify user requests, guaranteeing end-to-end integrity.

Finally, to resolve the cryptographic incompatibility, the system utilizes Solana's Native Secp256k1 Program. Instead of performing the heavy elliptic curve calculations within the main business logic, the Gateway contract invokes this precompiled native program via a Cross-Program Invocation (CPI). This allows the system to recover the public key from an Ethereum signature efficiently and securely. By comparing the recovered address with the linked identity stored in the Universal Wallet, the system achieves a trustless authentication mechanism, enabling an Ethereum user to drive state transitions on Solana using their native credentials.

5.4 Asynchronous State Consistency via Mutex and Saga Pattern

5.4.1 Problem Identification

The distributed nature of cross-chain communication introduces fundamental challenges regarding data consistency that do not exist in traditional monolithic architectures. Three critical issues were identified during the system design:

First, unlike centralized database systems which guarantee Atomicity, Consistency, Isolation, and Durability (ACID) within a single transaction scope, cross-chain operations are inherently non-atomic. A transaction executing on Ethereum cannot natively roll back its state based on the outcome of a transaction on Solana. Once a transaction is mined on the source chain, it is finalized, regardless of whether the subsequent steps on the destination chain succeed or fail. This lack of global atomicity creates a dangerous window where the system state is essentially "in transit" and undefined.

Second, independent blockchains operate in complete isolation; they are essentially "blind" to each other's state. The EVM Controller has no direct way of knowing whether the Solana Hub has successfully processed a minting request or if it failed due to a logic error (e.g., slippage or insufficient collateral). Without a robust synchronization mechanism, this information gap can lead to permanent state

divergence, where assets are locked on one chain without a corresponding value issuance on the other.

Third, the asynchronous gap between request and finalization exposes the system to synchronization risks, specifically Replay Attacks and Desynchronization. If a user can initiate multiple requests rapidly before the first one is settled, they might exploit the latency to double-spend their collateral or confuse the state machine. Furthermore, if a failure occurs on the destination chain and is not properly propagated back to the source, user funds could be permanently frozen in the smart contract, leading to a catastrophic loss of trust and financial value.

5.4.2 Proposed Solution

To restore a form of distributed atomicity and ensure eventual consistency, this thesis implements a dual-layer synchronization protocol combining Pessimistic Locking (Mutex) and the Saga Pattern.

To address the issue of state isolation and replay attacks, the EVM Controller implements a strict Mutex (Mutual Exclusion) Lock. Every user is assigned a unique, monotonically increasing nonce. When a user initiates a cross-chain request, the contract checks the lock status. If unlocked, it immediately sets a ‘isLocked‘ flag to true and increments the nonce. This action effectively freezes the user’s ability to interact with the system until the current lifecycle is resolved. This mechanism serializes the asynchronous events, converting a complex parallel state problem into a manageable sequential workflow, thereby neutralizing race conditions and replay attempts at the source.

To solve the problem of non-atomic failures and potential fund loss, the Guardian middleware adopts the Saga Pattern for distributed transactions. The cross-chain operation is modeled not as a single transaction but as a saga, it is a sequence of local transactions. The Guardian monitors the execution result on the Solana Hub. If the transaction succeeds, the saga proceeds forward, and the Guardian triggers a "Finalize" callback to the EVM chain to mint tokens and release the lock. However, if the Solana transaction fails (e.g., due to a sudden drop in collateral price causing a health factor check failure), the Guardian executes a Compensating Transaction. It submits a "Revert" proof to the EVM Controller, which triggers the release of the locked assets and the Mutex. This ensures that the system always converges to a consistent state either fully successful or fully rolled back eliminating the risk of funds being stuck in limbo.

5.5 Hub-Centric Liquidation Strategy with Liquidity Rebalancing

5.5.1 Problem Identification

In any Collateralized Debt Position (CDP) protocol, the liquidation mechanism is the ultimate line of defense against insolvency. However, in a cross-chain environment where collateral is held on a source chain (e.g., Ethereum) while debt is issued on a destination chain (e.g., Solana), the traditional liquidation model faces a critical latency bottleneck.

The primary problem is the reaction time to market volatility. Cross-chain communication typically involves a delay ranging from minutes to hours depending on network congestion and finality thresholds. In a scenario of extreme market volatility (a "Black Swan" event), the price of collateral can plunge significantly within seconds. If the protocol relies on a cross-chain message to trigger liquidation or to move assets from Ethereum to Solana to pay the liquidator, the delay may cause the position to become under-collateralized (Bad Debt) before the liquidation transaction is finalized. The inability to react instantaneously to price feeds renders standard cross-chain liquidation models unsafe and capital inefficient.

5.5.2 Proposed Solution

To guarantee system solvency and incentivize liquidators, this thesis introduces a Hub-Centric Liquidation Strategy supported by an active Liquidity Rebalancing Mechanism.

The core solution determines that the liquidation event must be executed exclusively on the Solana Hub. Since Solana holds the global state (the Universal Wallet) and has sub-second block times, it is the only environment capable of processing real-time solvency checks. When a user's Health Factor drops below the threshold, a liquidator interacts directly with the Solana Main Contract to repay the debt. This interaction is atomic and immediate, preventing the accumulation of bad debt regardless of the congestion status on the EVM chains.

To ensure that liquidators are compensated immediately without waiting for cross-chain bridging, the system maintains a "Liquidity Reserve" on Solana. The Guardian infrastructure implements a proactive Rebalancing Algorithm. It constantly monitors the ratio of collateral held in the EVM Vaults versus the Solana Reserve. When the reserve on Solana falls below a safety margin, the Guardian automatically bridges a portion of the idle collateral from the EVM spokes to the Solana Hub. This ensures that there is always sufficient physical liquidity on Solana to pay out liquidators instantly.

CHAPTER 6. CONCLUSION AND FUTURE WORK

6.1 Conclusion

The research and development of the Multi-chain Stablecoin Protocol have resulted in a significant architectural advancement within the Decentralized Finance landscape. When compared to existing solutions, the proposed system demonstrates distinct advantages over both single-chain lending protocols and traditional cross-chain bridges. Unlike protocols such as MakerDAO, which are confined to the liquidity of a single network, this system successfully unlocks the value of assets across heterogeneous chains without forcing users to migrate their capital physically. Furthermore, in contrast to the prevalent "Lock-and-Mint" bridge models that introduce systemic risk through the creation of wrapped assets, the "State Orchestration" architecture developed in this thesis maintains the security of native collateral. By synchronizing the financial state rather than the asset itself, the protocol minimizes the attack surface and eliminates the dependency on fragile wrapped token pegs.

Throughout the implementation of this thesis, several critical objectives have been achieved. The most notable contribution is the successful validation of the Hub-and-Spoke architecture, proving that the high-performance Solana blockchain can effectively serve as a global settlement layer for slower, liquidity-rich EVM networks. The project also introduced a novel cryptographic innovation with the Universal Wallet, leveraging Solana's native Secp256k1 program to allow Ethereum users to control their cross-chain positions trustlessly. This eliminates the significant user experience friction typically associated with managing multiple wallet standards. From a reliability standpoint, the integration of the Mutex locking mechanism on EVM controllers and the Saga pattern for error handling has proven robust, ensuring data consistency and zero fund loss even under simulated network partition scenarios.

However, the current iteration of the system is not without limitations. The most prominent constraint is the centralization of the Guardian infrastructure. While the middleware functions correctly as a proof-of-concept, it currently operates as a single server, representing a single point of failure and a trusted setup that contradicts the core ethos of decentralization. Reflecting on the process, the most valuable lesson learned was the paradigm shift required to design for "Eventual Consistency" rather than "Atomic Transactions." Developing for asynchronous distributed ledgers required a defensive programming approach, accepting that state synchro-

nization is probabilistic and time-dependent, necessitating rigorous fail-safe mechanisms like the implemented revert logic.

6.2 Future Work

To transition this academic prototype into a production-ready protocol, the immediate focus of future work must address the hardening of the system's security and reliability. The primary task is the decentralization of the Guardian Network. The current single-node middleware must be evolved into a consensus-based network of validators. This involves implementing a threshold signature scheme or a multi-party computation (MPC) framework, requiring a supermajority of independent nodes to attest to an event before a state transition is executed on the Solana Hub. Additionally, prior to any mainnet deployment, the smart contracts across both the EVM and SVM layers require comprehensive security audits by third-party firms to identify and rectify potential reentrancy vectors or arithmetic vulnerabilities that may have escaped the testing phase.

Looking towards the long-term evolution of the protocol, the research opens several avenues for advanced technological integration. A key area for exploration is the replacement of the optimistic Guardian network with Zero-Knowledge (ZK) Proofs. By implementing ZK-Light Clients directly on the Solana smart contracts, the system could mathematically verify the state roots of the Ethereum blockchain, achieving a truly trustless interoperability model that does not rely on human or server-based relayers. Furthermore, the protocol creates a foundation for a decentralized governance model. Future development should include the implementation of a DAO structure, allowing stakeholders to vote on critical risk parameters such as Loan-To-Value ratios and the onboarding of new collateral types, including Real-World Assets, thereby transforming the protocol into a community-owned public infrastructure.

REFERENCE

[1] S. Nakamoto, *Bitcoin: A peer-to-peer electronic cash system*, https://bitcoin.org/bitcoin.pdf, Accessed: 2024, 2008.

[2] A. Yakovenko, *Solana: A new architecture for a high performance blockchain v0.8.13*, https://solana.com/solana-whitepaper.pdf, Whitepaper, 2018.

[3] V. Buterin, *Ethereum: A next-generation smart contract and decentralized application platform*, https://ethereum.org/en/whitepaper/, 2014.

[4] A. Ferrante and Coral Team, *Anchor framework: A framework for solana's sealevel runtime*, https://www.anchor-lang.com/, Documentation, 2021.

[5] OpenZeppelin, *Openzeppelin contracts: The standard for secure blockchain applications*, https://www.openzeppelin.com/contracts, Library Documentation, 2023.