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.

0.1 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 Decentralized Finance (DeFi) ecosystem exhibits severe fragmentation, particularly within lending protocols and Collateralized Debt Position (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:

- **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.
- **Cross-Chain Money Markets (e.g., Radiant Capital):** These protocols 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 **??** 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 (Siloed Liquidity)	Fragmented Pools	Unified Global State (Hub-and-Spoke)
Verification	Native On-chain Verification	Trusted Third-Party (e.g., LayerZero Oracle)	Sovereign On-chain Verification (Solana)
3rd Party Risk	None	High (Bridge dependency)	Minimized (Cryptographic proof required)
Capital Effic.	Low (Trapped on one chain)	High	High

Bång 1: Comparison of Lending Architectures

0.2 Functional Overview

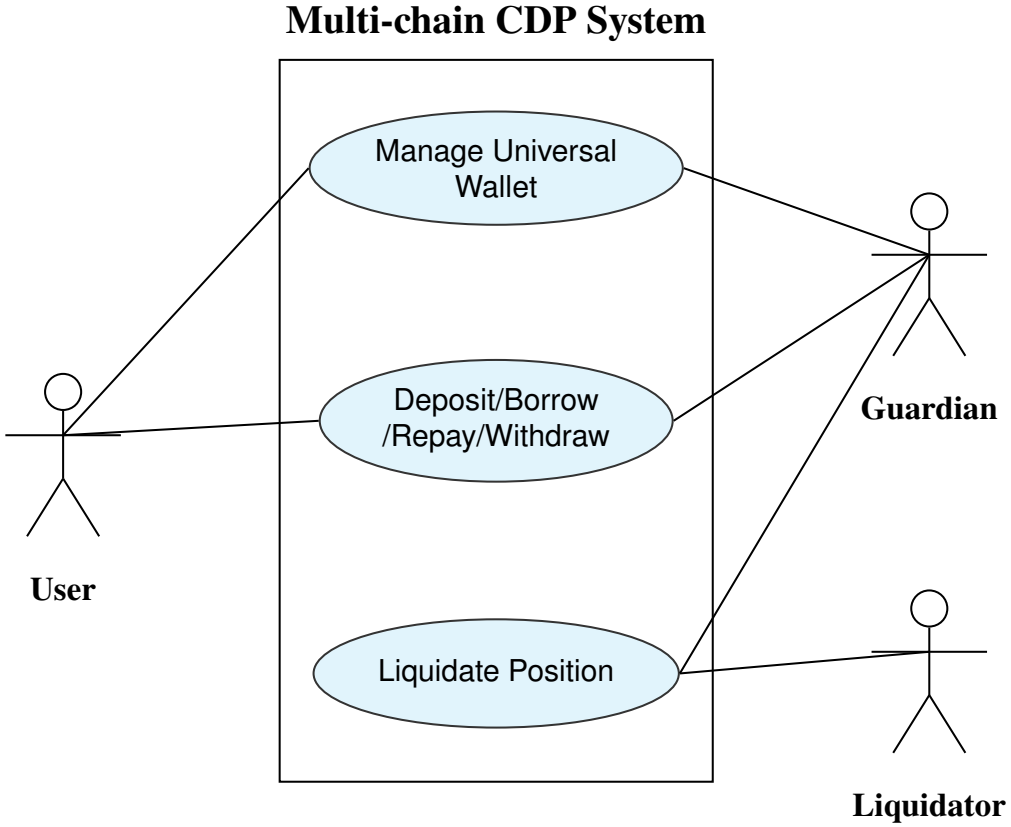
The system is designed to function as a decentralized lending protocol that orchestrates state across the Solana blockchain and various EVM-compatible chains.

0.2.1 General use case diagram

The system involves three primary actors:

1. **User:** The borrower who holds collateral on EVM chains. They interact with the system to deposit assets, mint stablecoins, and manage their global debt position.
2. **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.
3. **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 0.1 illustrates the general use cases.



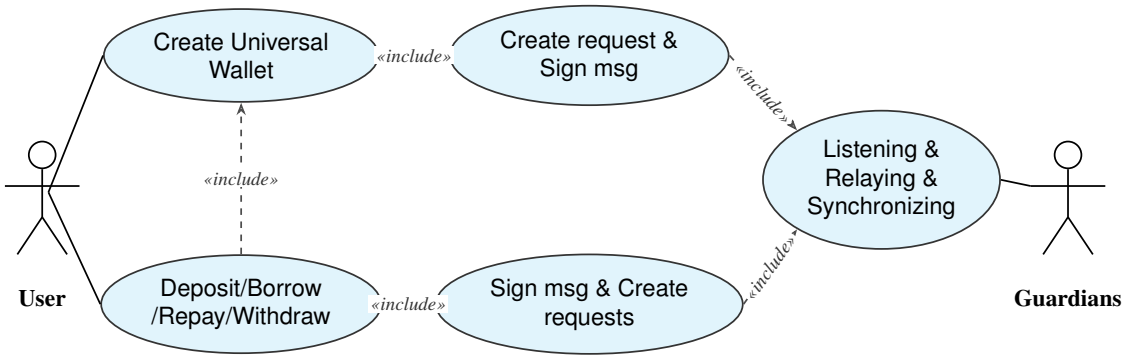
Hình 0.1: General Use Case Diagram

0.2.2 Detailed use case diagram

To provide a granular view of the system's operation, Figure **??** 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 delineates the following key logical dependencies:

- **Universal Wallet Dependency:** The “Deposit/Borrow/Repay/Withdraw” use case has an «`include`» relationship with the “Create Universal Wallet” use case. This enforces a strict precondition: a user must establish a valid identity (Universal Wallet PDA) on Solana before performing any asset-related operations.
- **Cryptographic Authorization:** Both the wallet creation and asset management use cases include the “Sign msg & Create requests” sub-use case. This illustrates that users do not write directly to the Solana blockchain; instead, they generate and cryptographically sign structured messages on the EVM interface.
- **Guardian Orchestration:** The message creation process further includes the “Listening & Relaying & Synchronizing” use case, executed by the Guardian actor. This signifies that 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 result back to the source chain.



Hình 0.2: Detailed Use Case Diagram

0.2.3 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.

As illustrated in Figure 0.3, the process follows a strict "Verify-then-Execute" lifecycle:

1. **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 con-

tract emits a event. Note that at this stage, no minting or unlocking occurs; the assets are simply locked or the request is queued.

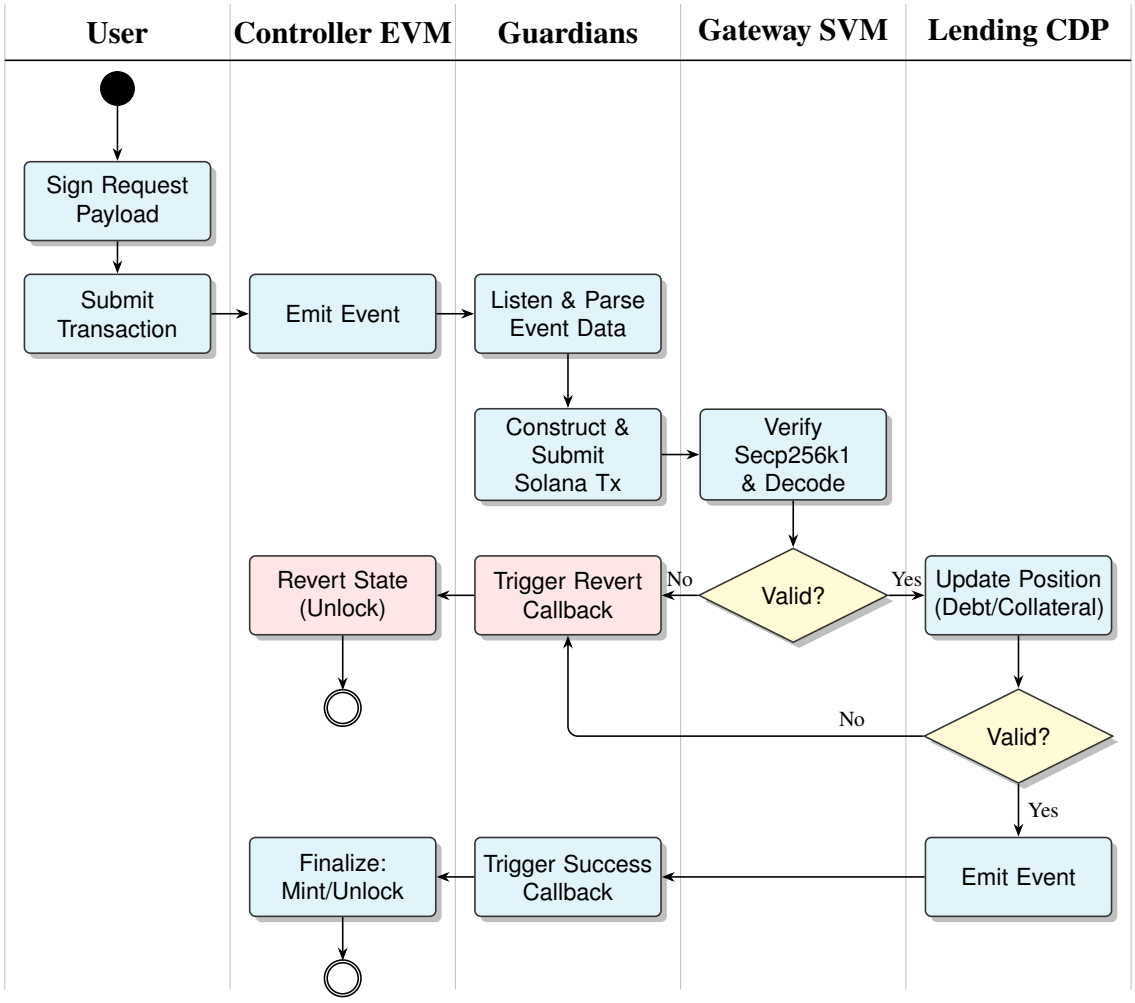
2. **Observation & Relay (Guardians):** The Guardian network detects the event on the EVM chain. Instead of verifying the logic off-chain, the Guardian parses the event data and encapsulates the raw signature into a transaction destined for Solana.
3. **Verification Layer (Solana Gateway):** The Gateway 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 **Valid**, the request is forwarded to the Lending CDP contract.
4. **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.
5. **Synchronization & Finalization:** Guardians listen for the outcome on Solana to trigger the corresponding callback on the EVM chain:
    - **Success Path:** If a event is captured, Guardians trigger the *Success Callback* on the EVM contract to finalize the action (e.g., Mint stablecoins).
    - **Revert Path:** 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.

0.3 Functional Description

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

0.3.1 Description of use case: Create Universal Wallet

This use case establishes the link between the user's EVM address and a Solana identity, creating the Universal Wallet storage on the Solana blockchain.



Hình 0.3: Business Process Activity Diagram

Use Case Name	Create Universal Wallet
Actors	User, Guardian, EVM Contract, Solana Main Contract
Pre-conditions	1. User has an EVM wallet (e.g., MetaMask) with funds for gas. 2. User possesses a Solana wallet to sign the verification message.
Main Flow	1. **Request on EVM:** The User initiates a transaction on the EVM Contract to request wallet creation. The contract emits an event containing the EVM address. 2. **Solana Signature:** Off-chain, the User signs a specific message containing their EVM address using their Solana private key. This signature is submitted to the Guardian API. 3. **Guardian Verification:** The Guardian captures the EVM event and verifies the off-chain Solana signature to ensure the User controls both addresses. 4. **Execution on Solana:** Upon successful verification, the Guardian submits a transaction to the Solana Main Contract to initialize the UniversalWallet PDA, mapping the EVM address to the new Solana state.
Post-conditions	An UniversalWallet is initialized on Solana. The User can now perform cross-chain actions.

Bảng 2: Functional Description: Create Universal Wallet

0.3.2 Description of use case: Deposit Collateral

This process allows users to lock assets on an EVM chain to increase their collateral balance on the Solana state.

Use Case Name	Deposit Collateral
Actors	User, Guardian, EVM Contract, Solana Main Contract
Pre-conditions	User has initialized a Universal Wallet and holds supported assets on the EVM chain.
Main Flow	1. **Lock Assets:** The User sign the payload and submit signatures by calls the `requestDeposit` function on the EVM Contract. The assets are transferred to the contract vault, and an event is emitted. 2. **Relay:** The Guardian detects the event then reads onchain data. After that, it forwards the payload and signature to the Solana Gateway. 3. **State Update:** The Gateway contract on Solana verify the signature and payload. After that, the Solana Main Contract identifies the user's Universal Wallet and increments the collateral balance for that specific chain ID. 4. **Synchronization:** The Guardian updates the EVM contract to confirm the deposit request has been synced, allowing the user to proceed with other actions.
Post-conditions	Collateral is locked on EVM. The collateral balance in the Universal Wallet on Solana is increased.

Bảng 3: Functional Description: Deposit Collateral

0.3.3 Description of use case: Mint Stablecoin

Users generate stablecoins against their collateral. This action requires strict verification of the Health Factor on Solana.

Use Case Name	Mint Stablecoin
Actors	User, Guardian, EVM Contract, Solana Main Contract
Pre-conditions	User has sufficient collateral (Health Factor remains above the minimum ratio after minting).
Main Flow	1. **Request:** The User signs a mint request payload then call `requestMint` function in the EVM Contract. The contract locks the user's mutex. 2. **Validation on Solana:** The Guardian listen the event then submits the signature and payload to Solana. The Main Contract verifies the signature and calculates the projected Health Factor. 3. **Debt Increase:** If the Health Factor is valid, the Solana contract increases the user's debt balance. 4. **Minting on EVM:** The Guardian catches the success event from Solana and executes a transaction on the EVM Contract to mint the stablecoins to the User's wallet and release the Mutex.
Alternative Flow	If the Health Factor is insufficient, the Solana transaction fails. The Guardian then triggers a `revert` on the EVM Contract to unlock the user's mutex.
Post-conditions	User receives stablecoins on EVM. Debt position is recorded on Solana.

Bảng 4: Functional Description: Mint Stablecoin

0.3.4 Description of use case: Repay Debt

Users return stablecoins to reduce their debt position and improve their Health Factor.

Use Case Name	Repay Debt
Actors	User, Guardian, EVM Contract, Solana Main Contract
Pre-conditions	User holds the protocol's stablecoin on the EVM chain.
Main Flow	1. **Lock Tokens:** The User sign the payload then calls `requestRepay` on the EVM Contract to submit signature. The specified amount of stablecoins is locked immediately, and an event is emitted. 2. **Relay:** The Guardian observes the event and submits the data and signature to Solana. 3. **State Update:** The Solana Gateway verify signature. After that, the Main Contract verifies the transaction and decreases the user's debt balance in the Universal Wallet. 4. **Completion:** The Guardian confirms the state update back to the EVM chain (burn stablecoin and updating the nonce).
Post-conditions	Stablecoins are burned on EVM. Debt balance on Solana is decreased.

Bảng 5: Functional Description: Repay Debt

0.3.5 Description of use case: Withdraw Collateral

This action allows users to retrieve their locked assets, provided their remaining collateral supports their outstanding debt.

Use Case Name	Withdraw Collateral
Actors	User, Guardian, EVM Contract, Solana Main Contract
Pre-conditions	User has sufficient free collateral (Health Factor remains safe after withdrawal).
Main Flow	1. **Request:** The User signs a withdraw request payload and submits it to the EVM Contract. The contract locks the user's mutex and emits an event if collateral token sufficient in vault EVM. 2. **Validation on Solana:** The Guardian forwards the request to Solana. The Main Contract checks if the remaining collateral is sufficient to cover the debt. 3. **State Update:** If valid, the Solana contract decreases the user's collateral balance. 4. **Unlock on EVM:** The Guardian triggers the EVM Contract to transfer the requested assets from the vault back to the User's wallet.
Alternative Flow	If the withdrawal would make the position insolvent, the Solana transaction is rejected. The Guardian reverts the request on EVM, keeping the assets locked.
Post-conditions	User receives assets on EVM. Collateral balance on Solana is reduced.

Bảng 6: Functional Description: Withdraw Collateral

0.4 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.

0.4.1 Security and Safety

Given the cross-chain nature of the protocol, security is the paramount requirement to prevent fund loss and bridge exploits.

- **Cryptographic Compatibility:** The Solana Gateway must natively support and verify **Secp256k1** signatures (EVM standard) to allow users to control their positions using existing Ethereum wallets without exposing private keys to a third party.
- **Cross-chain Replay Protection:** The system must implement a rigorous **Nonce Management** mechanism. Each transaction request must include a unique nonce associated with the specific Chain ID and User Address. The Solana contract must reject any request with a nonce lower than or equal to the current stored nonce to prevent replay attacks.
- **Guardian Decentralization (Trust Assumption):** The off-chain Guardian network must be designed to prevent a single point of failure. The system should require a threshold of signatures (e.g., Multisig or Threshold Signature Scheme) from Guardians before executing sensitive state changes (like unlocking collateral) to mitigate the risk of a single compromised Guardian node.
- **Atomic Revert Capabilities:** In the event of a failure during the cross-chain synchronization (e.g., Solana transaction fails due to slippage or insufficient health factor), the system must guarantee that the initial state on the EVM chain can be reverted (unlocking the user's Mutex) to prevent funds from being permanently frozen.

0.4.2 Performance and Efficiency

- **End-to-End Latency:** The total time for a generic user action (e.g., Minting) involves the sequence: $T_{EVM\_Confirm} + T_{Guardian\_Relay} + T_{Solana\_Finality} + T_{Guardian\_Callback}$. The system should aim for an end-to-end latency of under 30 seconds (excluding extreme network congestion on Ethereum Mainnet) to ensure a responsive user experience.
- **Compute Unit Optimization (Solana):** Since cryptographic verification (Secp256k1 recovery) is computationally expensive, the Solana smart contract must be

optimized to stay within the Block Compute Unit Limit, potentially utilizing Solana's native Secp256k1 program instructions to reduce costs.

- **Scalability:** The Guardian architecture must support concurrent monitoring of multiple EVM chains (e.g., Ethereum, BSC, Arbitrum, Optimism) without significant degradation in relaying speed.

0.4.3 Reliability and Availability

- **Eventual Consistency:** The system must ensure that the state between the EVM Spokes and the Solana Hub eventually converges. In case of network partitions (Guardian downtime), the system must be able to recover and process pending events once connectivity is restored.
- **Idempotency:** Guardian operations must be idempotent. Submitting the same event multiple times (due to network retries) must not result in double-counting of debt or collateral on the Solana state.
- **Uptime:** The Guardian nodes and the Solana RPC endpoints used for query/-submission must maintain high availability (99.9%) to prevent liquidation failures during times of high market volatility.

0.4.4 Usability and User Experience

- **Transparent Signing (EIP-712):** To protect users from phishing, all off-chain requests signed by the user must adhere to the **EIP-712** standard (Typed Structured Data Hashing and Signing). This ensures that users can read clearly structured data (Action, Amount, ChainID) in their wallet interface (e.g., MetaMask) before signing, rather than signing opaque hex strings.
- **Wallets Experience:** Users can interact with the entire protocol using multiple wallets. The complexity of the usage process is significantly reduced by the interactive interface that supports multiple wallets..

0.5 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 (Secp256k1), 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.