UNIVERISTY OF PIRAEUS - DEPARTMENT OF INFORMATICS
ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ – ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

MSc «Cybersecurity and Data Science»
ΠΜΣ «Κυβερνοασφάλεια και Επιστήμη Δεδομένων»

MSc Thesis

Μεταπτυχιακή Διατριβή

Thesis Title: Τίτλος Διατριβής:	**Design and Development of a MEV-Resistant Decentralized Multi-Collateral CDP Platform** Σχεδίαση και ανάπτυξη αποκεντρωμένης πλατφόρμας θέσεων χρέους πολλαπλών ενέχυρων ανθεκτικής σε Επιθέσεις MEV
Student's name-surname: Ονοματεπώνυμο φοιτητή:	**SPYRIDON PAPAGIANNOPOULOS** ΣΠΥΡΙΔΩΝ ΠΑΠΑΓΙΑΝΝΟΠΟΥΛΟΣ
Father's name: Πατρώνυμο:	**IOANNIS** ΙΩΑΝΝΗΣ
Student's ID No: Αριθμός Μητρώου:	ΜΠΚΕΔ24032
Supervisor: Επιβλέπων:	**Panayiotis Kotzanikolaou , Professor** Παναγιώτης Κοτζανικολάου, Καθηγητής

March 2026/ Μάρτιος 2026

3-Member Examination Committee

Τριμελής Εξεταστική Επιτροπή

Panagiotis Kotzanikolaou	Vangelis Malamas	Thomas Dasaklis
Professor	Teaching Staff	Teaching Staff

Περίληψη

Η ραγδαία εξέλιξη του οικοσυστήματος της Αποκεντρωμένης Χρηματοδότησης (Decentralized Finance – DeFi) έχει αναδείξει τα συστήματα Εξασφαλισμένων Θέσεων Χρέους (Collateralized Debt Positions – CDPs) ως θεμελιώδη πυλώνα για την παροχή ρευστότητας και την έκδοση αποκεντρωμένων σταθερών νομισμάτων (stable-coins). Ωστόσο, η εγγενής διαφάνεια των δημόσιων blockchain και η δομή του mempool καθιστούν τις υφιστάμενες υλοποιήσεις ευάλωτες σε επιθέσεις τύπου Maximal Extractable Value (MEV). Συγκεκριμένα, πρακτικές όπως το frontrunning και το liquidation sniping εκμεταλλεύονται την προτεραιότητα εκτέλεσης συναλλαγών, υπονομεύοντας τη δικαιοσύνη των μηχανισμών ρευστοποίησης και προκαλώντας συστημική αστάθεια και απώλεια κεφαλαίων για τους χρήστες.

Αντικείμενο της παρούσας μεταπτυχιακής διατριβής αποτελεί η σχεδίαση και ανάπτυξη μιας πρότυπης πλατφόρμας CDP, η οποία υποστηρίζει πολλαπλές μορφές ενεχύρων (multi-collateral) και ενσωματώνει εγγενή αντίσταση σε επιθέσεις MEV. Η προτεινόμενη λύση καινοτομεί εισάγοντας έναν μηχανισμό ρευστοποίησης βασισμένο στο κρυπτογραφικό σχήμα commit–reveal. Μέσω αυτής της μεθόδου, οι προσφορές των εκκαθαριστών (liquidators) υποβάλλονται αρχικά σε κρυπτογραφημένη μορφή (commit phase) και αποκαλύπτονται μόνο μετά την οριστικοποίηση του block και την πάροδο ενός ασφαλούς χρονικού παραθύρου. Η τεχνική αυτή αποκρύπτει την πληροφορία τιμής από το mempool, καθιστώντας τις ευκαιριακές παρεμβάσεις από bots και επικυρωτές (validators) οικονομικά ασύμφορες. Η αρχιτεκτονική του συστήματος υλοποιήθηκε σε Solidity, αξιοποιώντας το πλαίσιο ανάπτυξης Hardhat, και βασίζεται σε ένα σύνολο έξυπνων συμβολαίων (VaultManager, StabilityPool, LiquidationEngine, PriceFeed) που διασφαλίζουν την ακεραιότητα της δανειοληπτικής διαδικασίας. Το πρωτότυπο αναπτύχθηκε σε δοκιμαστικό δίκτυο συμβατό με το Ethereum Virtual Machine (EVM), όπου και υποβλήθηκε σε εκτεταμένα σενάρια προσομοίωσης επιθέσεων. Η πειραματική αξιολόγηση κατέδειξε ότι ο μηχανισμός commit–reveal μειώνει δραστικά την επιφάνεια επίθεσης για MEV, διασφαλίζοντας μια πιο δίκαιη διαδικασία ανακάλυψης τιμής κατά τη ρευστοποίηση, χωρίς να επηρεάζει αρνητικά τη λειτουργική αποδοτικότητα του συστήματος.

Abstract

The rapid expansion of Decentralized Finance (DeFi) has established Collateralized Debt Positions (CDPs) as a cornerstone for on-chain liquidity and stablecoin issuance. However, the inherent transparency of public mempools exposes existing CDP protocols to Maximal Extractable Value (MEV) vectors, specifically frontrunning and liquidation sniping. These adversarial strategies exploit transaction ordering privileges, undermining the fairness of liquidation mechanisms and introducing systemic instability.

This dissertation presents the design and implementation of a multi-collateral, MEV-resistant CDP platform, featuring a novel liquidation engine based on a commit–reveal cryptographic scheme. Unlike traditional auctions, the proposed mechanism decouples bid submission from bid revelation: liquidators submit encrypted bids (commit phase) which are revealed only after a secure block interval. This architecture effectively eliminates the pre-execution visibility of prices, rendering opportunistic arbitrage by searchers and validators economically infeasible.

The system is architected using modular smart contracts (VaultManager, StabilityPool, LiquidationEngine, PriceFeed) to ensure separation of concerns and security. A fully functional prototype was developed using Solidity and Hardhat and deployed on an Ethereum-compatible testnet. The simulation-based evaluation focused on attack surface reduction and gas efficiency, verifying that the commit–reveal mechanism significantly mitigates frontrunning risks without imposing prohibitive operational overhead. This thesis demonstrates that application-layer commit–reveal auctions can significantly reduce MEV extractability in CDP liquidations, without reliance on off-chain infrastructure.

List of Abbreviations

Abbreviation	Description
ABI	Application Binary Interface
ADR	Architecture Decision Record
AMM	Automated Market Maker
API	Application Programming Interface
batchId	Identifier for a liquidation batch
BPS	Basis Points
CDP	Collateralized Debt Position
CEI	Checks–Effects–Interactions (Solidity pattern)
CI	Continuous Integration
CLI	Command Line Interface
commit	Commitment phase of a commit–reveal auction
CR	Collateralization Ratio
DAI	Dai (MakerDAO stablecoin)
DAO	Decentralized Autonomous Organization
DApp	Decentralized Application
DEX	Decentralized Exchange
DoS	Denial of Service
E2E	End-to-End (testing)
ECDSA	Elliptic Curve Digital Signature Algorithm
EIP	Ethereum Improvement Proposal
EOA	Externally Owned Account
ERC	Ethereum Request for Comments
ERC-20	Ethereum Request for Comments 20 (fungible token standard)
ERC-4626	Ethereum Request for Comments 4626 (tokenized vault standard)
ESLint	JavaScript / TypeScript linting tool
ETH	Ether
EVM	Ethereum Virtual Machine
FCFS	First-Come, First-Served
FSM	Finite State Machine
FSS	Fair Sequencing Services
IDE	Integrated Development Environment
JSON	JavaScript Object Notation

JSON-RPC	JSON Remote Procedure Call
L1	Layer 1
L2	Layer 2
LTV	Loan-to-Value
MCR	Minimum Collateralization Ratio
MEV	Maximal Extractable Value
msg.sender	Transaction sender in Solidity
NFT	Non-Fungible Token
nonReentrant	Solidity reentrancy guard modifier
OZ	OpenZeppelin (smart contract library)
PBS	Proposer–Builder Separation
PoC	Proof of Concept
PoS	Proof of Stake
reentrancy	Recursive contract call vulnerability
reveal	Revelation phase of a commit–reveal auction
REPL	Read–Eval–Print Loop
RPC	Remote Procedure Call
SDK	Software Development Kit
SHA	Secure Hash Algorithm
SLA	Service Level Agreement
stETH	Staked Ether (Lido)
TDD	Test-Driven Development
TS	TypeScript
TWAP	Time-Weighted Average Price
tx.origin	Original external account initiating a transaction
TVL	Total Value Locked
UML	Unified Modeling Language
VM	Virtual Machine
VRF	Verifiable Random Function
WAD	Fixed-point unit with 18-decimal precision (10^{18})
WBTC	Wrapped Bitcoin
YAML	YAML Ain’t Markup Language

Table of Contents

1. Introduction

1.1 Background and Context

The emergence of blockchain technology has transformed traditional financial paradigms by enabling the creation of open, transparent and programmable economic systems. Among the most impactful innovations in this space is Decentralized Finance (DeFi), a rapidly expanding ecosystem that leverages smart contracts to provide financial services without centralized intermediaries. DeFi platforms facilitate lending, borrowing, trading and asset management through automated protocols deployed on public blockchains, primarily Ethereum. These systems are governed by deterministic code rather than institutional authorities, aligning with the principles of decentralization and trust minimization.
Beyond financial applications, blockchain has been extensively studied as a trust and integrity layer for security-critical distributed systems, enabling tamper-evident logging, accountability, and forensic traceability in adversarial environments [1].

Within the DeFi ecosystem, Collateralized Debt Position (CDP) mechanisms have become a cornerstone of decentralized lending and stablecoin issuance. A CDP allows users to lock collateral assets (such as ETH, WBTC, or other ERC-20 tokens) into a smart contract in exchange for minting a corresponding amount of stablecoins, typically pegged to a fiat currency like the U.S. dollar. The collateral must exceed the borrowed value, ensuring over-collateralization and system solvency. Prominent examples include MakerDAO's DAI, which pioneered this model and other derivative systems such as Liquity and Synthetix that expanded upon similar architectures.

Despite their success, CDP-based systems face intrinsic challenges related to liquidation mechanisms, which ensure system stability when users' collateral ratios fall below predefined thresholds. When undercollateralization occurs, the protocol triggers a liquidation event to sell the collateral and recover the outstanding debt. These liquidation events, however, operate within the public and permissionless environment of blockchain networks, where transaction ordering and visibility can be exploited by external participants. This vulnerability forms the foundation of a growing class of adversarial behaviors known as Miner Extractable Value (MEV), now referred to as Maximal Extractable Value to reflect the shift from mining to validator-based consensus and the inclusion of other actors like searchers and builders.

MEV arises when validators or sophisticated network participants reorder, insert, or censor transactions within a block to extract financial profit. In the context of CDP liquidations, MEV often manifests as frontrunning, backrunning, or sniping attacks, where bots compete to capture liquidation opportunities by exploiting the deterministic nature of mempool data and transaction ordering. These attacks can lead to unfair market dynamics, inefficient liquidations and reduced protocol stability, ultimately undermining user trust and systemic robustness within DeFi.

1.2 Motivation, Research Problem and Objectives

The motivation for this dissertation stems from the growing realization that Maximal Extractable Value (MEV) constitutes a systemic threat to the fairness, efficiency and long-term sustainability of decentralized finance (DeFi) protocols. As DeFi adoption increases and liquidation events become more frequent and higher value, liquidation mechanisms which were originally designed as an internal stability function, have evolved into profit targets for external actors. MEV searchers and validators can leverage mempool transparency, sophisticated ordering strategies and private relay infrastructure to capture liquidation rewards, often at the expense of protocol users and market integrity.

Conventional CDP liquidation designs typically rely on open auction mechanisms or first-come-first-served execution paths that broadcast transaction intent and bid information in cleartext prior to execution. While such models offer transparency, they inadvertently leak actionable information to the public mempool, enabling adversaries to front-run legitimate liquidators by submitting competing transactions with higher fees or preferential inclusion routes. In practice, this dynamic can result in liquidation markets dominated by automated bots, exclusion of ordinary participants and recurring priority gas auctions that amplify network congestion and cost. The consequence is a liquidation process that is economically inefficient and structurally unfair, undermining user trust and weakening the protocol's stabilizing function.

The transition to Proof-of-Stake (PoS) further compounds these issues by shifting significant ordering power to validators and the broader block-building supply chain. Although initiatives such as Flashbots and Proposer–Builder Separation (PBS) reduce public mempool exposure through private orderflow, these approaches primarily re-route or redistribute MEV rather than eliminating the underlying extraction vector. From a protocol design perspective, a critical gap remains: liquidation mechanisms that resist transaction-ordering exploitation by construction, without requiring off-chain infrastructure, privileged relays, or consensus-layer changes.

This dissertation addresses the above gap by investigating whether liquidation fairness can be improved through application-layer mechanism design. Specifically, it explores a commit–reveal auction approach that conceals bid information during an initial commitment phase and reveals it only after a safe interval. By introducing temporal separation between bid submission and bid disclosure, the design aims to disrupt the deterministic visibility that MEV strategies rely upon, shifting the liquidation process from reactive mitigation (e.g., private relays) toward preventive, protocol-native MEV resistance.

Accordingly, the central objective of this dissertation is to design, implement and evaluate a prototype multi-collateral CDP system that incorporates an MEV-resistant liquidation mechanism grounded in commit–reveal auctions. This objective is operationalized through the following specific research objectives:

1.3 Research Questions

To fulfill these objectives, the study is guided by the following research questions:

RQ1. How do MEV attacks affect the fairness and efficiency of liquidation mechanisms in existing CDP-based DeFi protocols and what architectural and cryptographic design principles can be employed to reduce the visibility and exploitability of liquidation transactions?

RQ2. Can a commit–reveal auction mechanism, combined with guarded oracles and batch settlement, effectively mitigate frontrunning and sniping attacks in a decentralized multi-collateral liquidation environment?

RQ3. What are the security trade-offs, performance implications, practical limitations and broader implications of the proposed MEV-resistant design for future research and real-world DeFi implementations?

These questions collectively shape the direction of the research, ensuring both theoretical understanding of the MEV problem and rigorous practical validation of the proposed solution.

Each research question is addressed through a dedicated part of the dissertation. The impact of MEV on liquidation fairness and the relevant design principles (RQ1) are analyzed in Chapters 2 and 3. The effectiveness of the commit–reveal auction mechanism against frontrunning and sniping attacks (RQ2) is developed and evaluated in Chapters 4,

5, and 6. The security trade-offs, performance implications, limitations, and implications for future research and real-world deployments (RQ3) are examined through experimental results and synthesized in Chapter 7.

1.4 Scope of the research and Contributions

The scope of this dissertation centers on the architectural design, prototypical implementation and security evaluation of an MEV-resistant liquidation mechanism within a Collateralized Debt Position (CDP) ecosystem. The research is situated at the convergence of blockchain architecture, applied cryptography and cryptoeconomic security, with a specific focus on Ethereum Virtual Machine (EVM) compatible environments. The primary undertaking involves the comprehensive analysis of transaction-ordering vulnerabilities in current DeFi protocols, specifically frontrunning and liquidation sniping, and the subsequent engineering of a "Commit–Reveal" liquidation engine designed to mitigate these vectors at the application layer.

From a methodological standpoint, the work encompasses the end-to-end development of a modular smart contract suite, comprising the VaultManager, StabilityPool and LiquidationEngine, implemented in Solidity using the Hardhat framework. The scope extends to the functional validation and security auditing of these components under simulated adversarial conditions. This includes the formulation of a threat model specifically targeting mempool-based attacks, followed by an empirical evaluation of the proposed mechanism’s ability to cryptographically conceal bid information and preserve protocol fairness during the liquidation process.

However, several limitations characterize the boundaries of this study. First, the prototype is evaluated within a controlled testing environment, utilizing local Hardhat networks and public testnets. Consequently, the experimental setup cannot fully replicate the hostile and highly competitive nature of the Ethereum mainnet's "Dark Forest," where generalized MEV bots operate with nanosecond-level latency optimization and complex priority gas auction strategies. While the simulation validates the logical correctness and security properties of the mechanism, it does not account for the extreme network congestion or adversarial bidding wars often observed in high-value production environments.

Furthermore, the architectural decision to implement a Commit–Reveal scheme introduces specific trade-offs regarding system responsiveness and capital efficiency. By mandating a temporal delay between the commitment of a bid and its revelation, the liquidation process transitions from an atomic, synchronous operation to an asynchronous one. This introduces volatility risk, as the market value of the underlying collateral may deviate significantly during the reveal window, potentially impacting liquidator profitability and system solvency. Additionally, the requirement to lock capital during the commitment phase creates a degree of capital inefficiency compared to the instant settlement models found in traditional lending protocols like Aave [2] or Compound [3] .

Finally, this research is strictly confined to on-chain, application-layer mitigations. It does not integrate external infrastructure solutions such as Proposer-Builder Separation (PBS) relays (e.g., Flashbots), nor does it model complex governance tokenomics or advanced oracle aggregation strategies beyond standard Chainlink integrations. Economic parameters, such as stablecoin peg stability mechanisms and incentive calibration, are simplified to maintain focus on the security and architectural dimensions of MEV resistance. Despite these constraints, the work provides a rigorous proof-of-concept for endogenous MEV resistance, demonstrating the feasibility of securing liquidation markets through protocol-level logic without reliance on centralized off-chain actors.

1.5 Structure of the Dissertation and Expected Contributions

This dissertation contributes to ongoing research in secure and fair decentralized finance systems by providing both theoretical and practical advancements in the design of MEV-resistant liquidation mechanisms. The expected contributions of this work are summarized as follows:

1. Architectural Design Framework: A detailed system architecture for a multi-collateral, MEV-resistant CDP platform composed of modular smart contracts, including the VaultManager, StabilityPool, LiquidationEngine and oracle components, which collectively enable stablecoin issuance and liquidation under cryptographic fairness constraints.

2. Prototype Implementation: A fully functional prototype implemented in Solidity using the Hardhat development framework and deployed on an Ethereum-compatible test network, demonstrating the feasibility of integrating commit–reveal mechanisms within practical DeFi liquidation logic.

3. Security and MEV Analysis: A comprehensive analysis of the impact of commit–reveal auctions on MEV exploitability, focusing on attack surface reduction, frontrunning mitigation and fairness under simulated adversarial conditions.

4. Performance Evaluation: Empirical evaluation of gas efficiency, transaction latency and system behavior, with comparative discussion against conventional liquidation mechanisms.

5. Research Insights and Future Directions: Conceptual insights into how MEV-resistant liquidation designs can inform broader classes of DeFi primitives, including lending protocols, decentralized exchanges and oracle systems.

The remainder of this dissertation is structured to provide a logical progression from the theoretical underpinnings of MEV to the practical engineering of an MEV-resistant solution. The document is organized into eight chapters, as follows:

Chapter 1 – Introduction: Establishes the research context at the intersection of decentralized finance (DeFi) and Maximal Extractable Value (MEV). It presents the motivation, problem statement, research objectives, scope, contributions, and guiding research questions.

Chapter 2 – Background and Related Work: Provides an overview of DeFi lending systems and Collateralized Debt Position (CDP) protocols, explains liquidation mechanisms in existing systems, introduces the MEV problem (with emphasis on liquidation-specific vectors such as frontrunning and sniping), reviews prior mitigation techniques, and synthesizes the literature to identify the key research gap addressed in this work.

Chapter 3 – System Architecture and Design: Formalizes the high-level architecture of the proposed multi-collateral MEV-resistant CDP platform. It details the modular components (VaultManager, StabilityPool, LiquidationEngine, GuardedOracle/PriceFeed), their interactions, sequence diagrams, and the core MEV-resistance design principles, including the commit–reveal auction mechanism and guarded oracle safeguards.

Chapter 4 – Implementation Details: Describes the technical realization of the architecture in Solidity using the Hardhat development framework. It covers contract-level logic, key functions, the commit–reveal scheme implementation, testing setup, and considerations for gas efficiency and deployment.

Chapter 5 – Security Analysis and MEV Resistance Evaluation: Defines the threat model and adversarial assumptions, enumerates the attack surface, analyzes vulnerability classes with corresponding mitigations, and evaluates resilience against MEV-specific attacks (frontrunning, sniping, copy-bidding, flash-loan manipulation) as well as conventional smart-contract risks.

Chapter 6 – Simulation and Testing: Presents the experimental setup (Hardhat-based deterministic simulation, mock infrastructure, test categories), performance metrics (gas costs, latency, determinism), evaluation of MEV resistance, and a comparative analysis against traditional liquidation mechanisms (FCFS and auction-based).

Chapter 7 – Conclusions and Future Work: Summarizes the key findings and contributions, discusses design trade-offs (latency vs. fairness, gas overhead, parameter sensitivity), acknowledges prototype limitations, and outlines directions for future research, formal verification, economic modeling, Layer-2 adaptations, and production deployment considerations.

2. Background and Related Work

2.1 DeFi Lending, Stablecoins and Collateralized Debt Position (CDP) Systems

Decentralized Finance (DeFi) introduces an alternative financial architecture in which lending, borrowing, and asset exchange are executed through programmable smart contracts rather than centralized intermediaries. Operating on public blockchains such as Ethereum, DeFi protocols offer open access, composability, and transparent execution, allowing users to interact directly with financial primitives without custodial trust. Among the most established DeFi primitives are over-collateralized lending markets. Protocols such as Aave and Compound enable users to deposit crypto assets as collateral in order to borrow other assets, subject to minimum collateralization ratios. These systems rely on continuous price monitoring and automatic liquidation to preserve solvency under market volatility. When collateral value falls below protocol-defined safety thresholds, positions become eligible for liquidation, and collateral is sold to repay outstanding debt. Stablecoins play a central role in these lending systems by providing a unit of account and medium of exchange insulated from cryptoasset volatility. While fiat-backed stablecoins depend on off-chain reserves, crypto-collateralized stablecoins rely entirely on on-chain mechanisms, over-collateralization, and liquidation processes. As a result, the robustness and fairness of liquidation mechanisms directly impact both protocol solvency and user outcomes.

Within this broader DeFi lending landscape, Collateralized Debt Position (CDP) systems emerged as a foundational model for on-chain credit creation and crypto-collateralized stablecoin issuance. CDP systems constitute the core mechanism underpinning decentralized stablecoin protocols in the DeFi ecosystem. As illustrated in Figure 1, a CDP system is composed of interacting modules that manage collateralized borrowing, stablecoin issuance, and liquidation. Users deposit collateral assets (e.g., ETH, WBTC) into a Vault managed by the CDP Manager, which locks the assets and allows the minting of stablecoins such as DAI. The amount of debt issued is constrained by a minimum Collateralization Ratio (CR), ensuring that the value of locked collateral exceeds outstanding liabilities. The system continuously monitors position health through a Risk Engine that relies on external Price Oracles to determine real-time collateral values. When market conditions cause the CR to fall below a predefined threshold, the Liquidation

Engine is triggered to restore solvency. In this process, third-party liquidators repay the outstanding debt in exchange for discounted collateral, typically via auction-based or pool-based mechanisms. Additional components such as the Stability Fee (interest) module and Stability Pool contribute to regulating supply and absorbing liquidations. The interaction between lending markets, stablecoin issuance, and liquidation enforcement culminates in CDP systems, which form the backbone of decentralized monetary systems [5]. These systems combine price oracles, risk parameters, and liquidation logic into an automated credit infrastructure whose security depends not only on smart contract correctness but also on economic and game-theoretic properties.

The most influential and widely studied implementation of this model is MakerDAO, which serves as the canonical reference architecture for CDP-based decentralized credit. MakerDAO pioneered CDP-based stablecoins on Ethereum and demonstrates how vault management, oracle-based pricing, and automated liquidation collectively ensure system robustness. Users deposit approved collateral assets into Vaults and generate DAI by drawing debt against this collateral. The protocol enforces minimum collateralization ratios that vary by asset class and reflect volatility and liquidity risk. When a Vault becomes under-collateralized due to price movements, MakerDAO triggers liquidation to ensure that the outstanding debt can be repaid. Historically, this process has relied on on-chain auctions in which third-party liquidators bid for collateral until sufficient funds are raised to cover the debt and associated penalties. While auction-based liquidations theoretically enable price discovery, their on-chain and transparent nature exposes them to strategic behavior. Empirical analyses of MakerDAO liquidations show that auctions often suffer from delayed participation, congestion, and discounted collateral sales, particularly during periods of market stress. Although subsequent protocol upgrades improved auction efficiency, the fundamental reliance on visible, time-sensitive bidding persists. This transparency allows participants to observe bids in real time, creating opportunities for frontrunning, sniping, and priority gas auctions. The MakerDAO model has inspired multiple successor systems, including Liquity, Reflexer (RAI), and Synthetix, each adapting the CDP concept to different stability or governance assumptions. However, most of these systems continue to depend on liquidation mechanisms that expose bid information or reward speed over fairness, inheriting similar vulnerabilities at the execution layer.

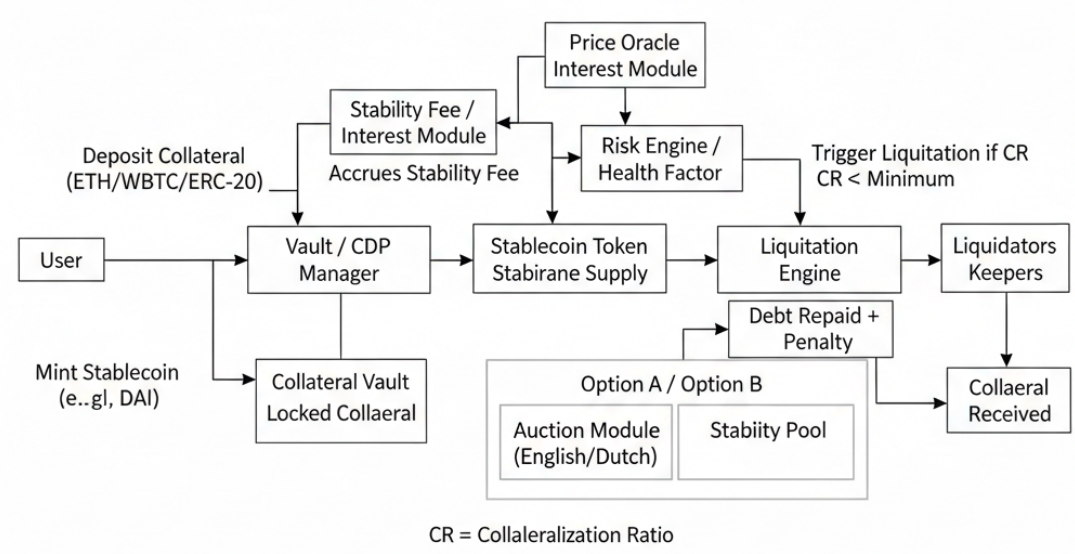

Figure 1 High-level architecture of a CDP system with vaults, price oracles and liquidation mechanisms

While early iterations such as Single-Collateral DAI utilized English auctions, modern protocols including MakerDAO's Multi-Collateral DAI and systems such as Liquity have evolved toward Dutch auctions or Stability Pool-

based liquidations to improve capital efficiency. Despite these variations, the core architectural challenge remains the same: designing a liquidation engine that preserves fairness and efficiency within an adversarial, highly transparent, and asynchronous distributed ledger environment. Similar principles of modular trust separation and fine-grained control have been explored in multi-blockchain architectures, where security-critical responsibilities are decomposed across interoperating ledgers to reduce systemic risk and improve auditability [4].

2.2 Liquidation Mechanisms in DeFi Protocols

Liquidation mechanisms define how a decentralized finance (DeFi) protocol resolves undercollateralized positions in order to preserve solvency. Beyond their economic role, these mechanisms critically influence the protocol's exposure to Maximal Extractable Value (MEV), as they determine how liquidation opportunities are discovered, priced and executed under conditions of transaction ordering competition. In contemporary DeFi systems, three liquidation models dominate the design space.

- **English auctions** represent one of the earliest approaches to decentralized liquidations. In this model, the protocol initiates an on-chain auction in which bidders compete by submitting increasingly higher bids for the distressed collateral. While English auctions provide strong theoretical price discovery guarantees, they are poorly suited to blockchain environments. Auction durations introduce latency, while bid visibility enables last block bid sniping, where adversaries with superior ordering or latency advantages submit winning bids at the final moment. As a result, the auction outcome often reflects transaction inclusion power rather than competitive valuation. [5]

- **Dutch auctions**, most notably used in MakerDAO's liquidation framework, invert the bidding process. The collateral is initially offered at a high price that deterministically decays over time until a participant accepts the trade. Although this removes explicit bidding wars, it introduces a different vulnerability. Because the optimal execution moment is publicly computable, liquidators engage in a race to submit the accepting transaction as soon as the price crosses a profitability threshold. In practice, this produces priority gas auctions, where competing actors aggressively increase fees to secure first inclusion, transferring value to validators rather than improving liquidation efficiency.

- **Fixed-spread or keeper-based systems**, employed by protocols such as Aave and Compound, further simplify the process by granting an immediate liquidation right to the first valid transaction. Liquidators receive a predefined bonus, typically between 5% and 10% of the collateral value. While operationally efficient, this design creates a strictly binary outcome: the first included transaction captures the entire reward. Consequently, liquidation opportunities are highly attractive to automated searchers and competition manifests almost exclusively through transaction ordering and fee escalation rather than through economic pricing.

Despite their structural differences, all three models share a fundamental vulnerability rooted in public mempool transparency. In Ethereum's transaction model, liquidation intent is broadcast to the mempool before execution. This visibility allows adversarial actors to observe pending liquidations, simulate outcomes and strategically submit competing transactions with higher fees or preferential routing. Over time, this dynamic transforms liquidation events into adversarial contests dominated by sophisticated MEV searchers, marginalizing ordinary participants and degrading the fairness of the system.

The consequence is a liquidation process that is increasingly disconnected from its original stabilizing purpose. Instead of efficiently restoring protocol solvency, liquidation mechanisms become arenas for MEV extraction, leading to recurrent gas wars, increased network congestion and value leakage to transaction ordering intermediaries. These

systemic limitations motivate the exploration of alternative liquidation designs that reduce information leakage at the protocol level, rather than relying on reactive mitigation strategies external to the smart contract architecture.

2.3 The MEV Problem in DeFi Liquidations

Maximal Extractable Value (MEV), formerly referred to as Miner Extractable Value, quantifies the total value that can be extracted by block producers and their associated supply chain through the strategic reordering, insertion, or censorship of transactions prior to block inclusion. In the Proof-of-Stake era, MEV has evolved into a specialized ecosystem involving searchers, who identify profitable opportunities, and builders, who assemble transaction bundles that are submitted to validators. This transformation has professionalized MEV extraction and intensified competition around transaction ordering [18].

At its core, MEV arises from the combination of two properties of blockchain systems: (i) the public visibility of pending transactions in the mempool, and (ii) the discretion of block producers in ordering transactions within a block. Because transactions are broadcast before execution and their effects can be simulated off-chain, adversarial actors are able to anticipate profitable state transitions and strategically position their own transactions to capture value. This gives rise to a class of ordering-dependent attacks, the most prominent of which are frontrunning, backrunning, and sandwich attacks [7], [8].

In a frontrunning attack, an adversary observes a pending transaction in the public mempool that is expected to generate profit (e.g., a liquidation or large trade). The adversary then submits a competing transaction with a higher gas fee, ensuring it is included before the original transaction. As a result, the attacker captures the available profit, while the original transaction either fails or executes under less favorable conditions. Conversely, backrunning involves inserting a transaction immediately after a target transaction to exploit its price impact, typically through arbitrage. Sandwich attacks combine both strategies, placing transactions before and after a victim transaction to extract value from induced price movement.

Figure 2 illustrates this process: a user submits a transaction (T_1) to the mempool; a searcher detects it and submits a higher-fee transaction (T_2); the block producer then orders T_2 before T_1, enabling the attacker to capture the profit opportunity.

In the context of Collateralized Debt Position (CDP) protocols, MEV manifests most severely during liquidation events. When a vault becomes undercollateralized, the protocol exposes a deterministic and time-sensitive profit opportunity: the liquidation bonus. Because this opportunity is publicly observable and economically attractive, multiple actors compete to execute the liquidation transaction first. In practice, this competition is dominated by automated bots with low-latency infrastructure and advanced transaction routing capabilities.

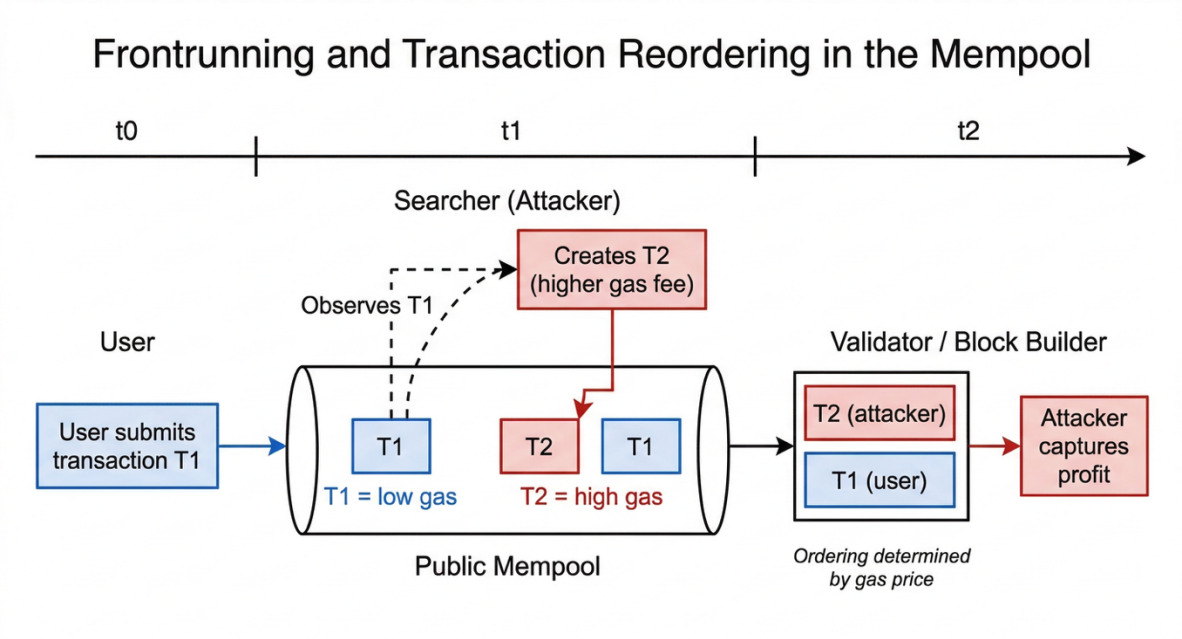

Figure 2 Frontrunning attack exploiting public mempool visibility and gas-based transaction ordering.

Frontrunning constitutes the primary attack vector in this setting. A liquidator submits a transaction to liquidate an undercollateralized position, which becomes visible in the mempool. An adversarial searcher detects this transaction, simulates its outcome, and submits a competing transaction with a higher gas price. Since Ethereum validators prioritize transactions based on fees, the attacker's transaction is included first, capturing the liquidation reward and invalidating or displacing the original transaction. This process results in repeated priority gas auctions, where competing actors continuously increase fees to secure execution priority.

A closely related strategy is liquidation sniping, in which bots monitor oracle updates and vault health factors to predict when a position is about to become liquidatable. Instead of reacting to existing transactions, attackers proactively prepare and submit liquidation transactions at the exact moment the threshold is crossed. This leads to multiple redundant transactions competing within the same block, increasing gas consumption without improving price discovery or system efficiency.

The cumulative effects of MEV-driven liquidation behavior are detrimental to decentralized finance ecosystems. Market fairness deteriorates, as participation becomes restricted to actors with specialized infrastructure. System efficiency declines due to gas wars and network congestion during periods of high liquidation activity. Protocol security is indirectly weakened, as persistent value extraction can fragment liquidity and destabilize pegged assets. Moreover, the reliance on MEV-intensive execution flows reinforces validator and infrastructure centralization, concentrating power among entities with superior ordering capabilities.

Although initiatives such as Flashbots and MEV-Boost aim to make MEV extraction more transparent or redistribute its proceeds, they do not address the root cause: the public visibility of transaction intent prior to execution. Consequently, mitigating MEV in liquidation mechanisms requires architectural solutions that reduce or conceal actionable information at the protocol level, rather than relying on off-chain coordination or preferential transaction routing.

2.4 Existing MEV Mitigation Techniques

Efforts to mitigate MEV have emerged from both the blockchain infrastructure layer and protocol-level mechanism design. The most prominent techniques can be grouped into four categories: private transaction relays (Flashbots), Proposer–Builder Separation (PBS), commit–reveal cryptographic mechanisms and fair sequencing protocols.

- **Flashbots, Off-Chain Bundle Submission.** Flashbots [9] is an industry-led initiative launched in 2021 to address MEV exploitation by providing a private relay network where traders can submit transaction bundles directly to miners or validators. These bundles specify explicit transaction orderings, preventing public mempool visibility. Flashbots' approach improves transparency around MEV extraction and reduces gas bidding wars, but it does not eliminate MEV. Instead, it redistributes MEV opportunities to approved searchers and validators, introducing centralization risks and off-chain trust assumptions. [10]

- **Proposer–Builder Separation (PBS).** PBS, introduced as part of Ethereum's roadmap following the transition to Proof-of-Stake, aims to separate block proposal from block construction. Specialized entities called "builders" compete to construct blocks with the most valuable transactions, while "proposers" (validators) select blocks based on bids. This design mitigates MEV centralization by distributing roles and increasing transparency. However, PBS primarily addresses validator-level MEV, not application-layer exploitation in DeFi protocols like liquidation frontrunning.

- **Commit–Reveal Schemes.** Commit–reveal mechanisms, widely studied in cryptographic literature and previously used in blockchain contexts such as random number generation and sealed-bid auctions, introduce temporal secrecy. Participants first commit to a hashed value (commit phase) and reveal it later with the original input and nonce (reveal phase). By hiding information until after block inclusion, commit–reveal schemes reduce the visibility window for frontrunners. Projects like Gnosis Auction [11] and early research on fair sequencing protocols have explored this model for fairer on-chain bidding. Its limitation, however, lies in added latency and complexity in user experience.

- **Fair Sequencing Services (FSS).** Fair sequencing protocols aim to enforce equitable transaction ordering independent of gas price bidding. Examples include Arbitrum's FSS proposal and academic designs like Aequitas [12] which use cryptographic commit–reveal or timestamping to ensure that transaction order reflects submission time rather than fee incentives. While promising, these systems typically require consensus-level integration, making them less applicable to individual DeFi protocols deployed on existing blockchains.

Existing MEV mitigation approaches span multiple layers of the blockchain stack, including off-chain relay systems such as Flashbots, consensus-layer approaches such as Proposer–Builder Separation (PBS), and protocol-level techniques such as commit–reveal schemes, fair sequencing services, and guarded oracle mechanisms. A recent systematization of knowledge by Yang et al. [13] surveys more than thirty MEV countermeasures across these categories and concludes that no single approach provides a comprehensive solution in practice. While commit–reveal mechanisms are identified as a promising protocol-level countermeasure, their application to liquidation-specific settings remains limited. Table 1 summarizes the most relevant strands of the literature and positions the proposed architecture relative to existing approaches.

2.5 Empirical and Security Research on DeFi Liquidations

A growing body of empirical research has examined liquidation behavior in major DeFi lending protocols, with particular focus on efficiency, borrower outcomes, and systemic risk [14] present a large-scale empirical analysis of liquidations across MakerDAO, Aave, Compound, and dYdX, demonstrating that existing liquidation mechanisms, both fixed-spread and auction-based, frequently result in excessive collateral sales and significant borrower losses, especially during periods of market volatility and network congestion. Zhou et al. [15], in a systematization-of-knowledge study of DeFi attacks, further identify liquidations as a recurring locus of economic inefficiency and adversarial exploitation, intersecting oracle behavior, transaction ordering, and incentive design. Complementing this empirical perspective, security-oriented research highlights how liquidation mechanisms interact with adversarial blockchain environments. Qin et al. [16] analyze flash-loan-powered attacks and demonstrate how attackers can manipulate oracle prices or market conditions within a single atomic transaction, force liquidations, and capture discounted collateral before prices revert. These findings show that liquidation systems are not isolated financial primitives but composable attack surfaces that combine oracle design, transaction atomicity, and ordering-dependent execution.

Systematization-of-knowledge (SoK) studies reinforce this view by cataloging liquidation mechanisms as persistent sources of vulnerability. Zhou et al. [15] document how oracle manipulation, MEV-driven transaction ordering, governance misconfiguration, and reentrancy risks frequently converge around liquidation logic. However, these analyses generally treat the liquidation execution mechanism itself as fixed, focusing on how adversaries exploit existing designs rather than questioning whether the liquidation process could be redesigned to be inherently more robust under adversarial conditions. [16] Across this literature, mitigation efforts predominantly focus on strengthening liquidation triggers, through guarded oracles, staleness checks, deviation bounds, or circuit breakers, while implicitly assuming that competition among liquidators will naturally yield fair outcomes once liquidation begins. Empirical evidence, however, suggests that this assumption does not hold in adversarial execution environments, leaving the liquidation execution phase exposed to MEV extraction and strategic manipulation. Beyond borrower-level inefficiencies, recent empirical work has highlighted the systemic implications of DeFi liquidation dynamics. Lehar and Parlour [17] analyze liquidation activity in major lending protocols such as Aave and Compound and show that liquidations frequently involve flash loans and immediate collateral sales that resemble large block trades on decentralized exchanges [17]. Their findings demonstrate that forced liquidations generate both temporary and persistent price impacts, amplifying market volatility and increasing the likelihood of cascading liquidations. Importantly, the study emphasizes that rapid, competition-driven liquidation mechanisms can transform localized insolvency events into protocol-wide instability. While the authors document the fragility induced by current liquidation designs, they do not propose alternative execution mechanisms to mitigate these effects, leaving open the question of how liquidation processes can be redesigned to reduce systemic risk under adversarial conditions.

2.6 Synthesis of Related Work and Identified Research Gap

Taken together, prior research on DeFi liquidations and MEV can be grouped into four broad strands, each addressing a different aspect of protocol safety but leaving important gaps at the liquidation execution layer.

First, empirical studies of lending protocols document inefficiencies in existing liquidation designs, including delayed participation, discounted collateral sales, and borrower over-penalization. Analyses by [7] and related work show that liquidation outcomes depend not only on price movements but also on network congestion, participation incentives, and transaction ordering. However, these studies largely treat transaction ordering as an external constraint rather than a design variable that can be addressed within the protocol itself [14]. Second, MEV research demonstrates that transparent mempools and discretionary transaction ordering create systematic incentives for frontrunning, priority gas auctions, and liquidation sniping. In [18] the authors formalize MEV and empirically show how ordering-dependent execution enables extractable value in decentralized systems, including liquidation-related

opportunities. While this literature convincingly characterizes the problem and quantifies its impact, it remains largely descriptive and infrastructure-oriented, offering limited guidance on how application-layer protocols can neutralize MEV by design [18]. Third, infrastructure-level mitigations such as private transaction relays and proposer–builder separation aim to reduce public frontrunning and validator-level MEV. Studies of PBS and private order-flow show that these approaches can improve efficiency and transparency under certain conditions, but their effectiveness depends on partial adoption, trusted intermediaries, and validator incentives that remain external to individual DeFi protocols. As a result, they provide no protocol-level fairness guarantees for users participating in liquidation events [19]. Fourth, research on oracle manipulation and flash-loan attacks motivates the use of guarded price feeds, time-weighted averages, and circuit breakers to protect liquidation eligibility. Frameworks such as SecPLF demonstrate that strengthening oracle update logic can significantly reduce the feasibility of price-based liquidation attacks. However, these approaches focus primarily on the liquidation trigger and do not address fairness or MEV extraction during the liquidation execution phase itself [20]. What remains largely unexplored is a protocol-native liquidation mechanism that explicitly assumes adversarial transaction ordering and minimizes MEV at the execution layer. Existing designs either reward speed through first-come execution or rely on visible auctions that invite strategic bidding and ordering manipulation. None integrate guarded price inputs with a liquidation process that conceals bid information, enforces deterministic settlement, and equalizes access among participants by construction. This gap motivates the approach developed in this dissertation: a liquidation architecture that combines hardened oracle inputs at the trigger layer with a commit–reveal, uniform-clearing batch auction at the execution layer. By hiding bids during the commitment phase, resolving liquidations in batches, and enforcing a single clearing price, the mechanism reduces liquidation-phase MEV without reliance on privileged transaction routing, off-chain relays, or consensus-level changes [21].

Table 1 Comparative Analysis of MEV Mitigation Techniques

Approach	**Layer**	**Core Principle**	**Representative Work**	**Advantages**	**Limitations**
Flashbots	Off-chain	Private bundle submission	[9], [10]	Reduces public mempool gas wars	Centralization, off-chain trust, does not eliminate MEV
PBS	Consensus	Proposer-builder separation	[18]	Reduces validator concentration of MEV	Does not directly solve application-layer liquidation MEV
Commit-Reveal	Protocol	Temporal bid secrecy	[11]	Conceals bid information, improves fairness	Latency, added interaction complexity

Fair Sequencing Services	Protocol / Consensus	Order fairness independent of gas bidding	[12]	Reduces ordering manipulation	Requires infrastructure or consensus support
Oracle Guards (TWAP, bounds, circuit breakers)	Protocol	Harden liquidation trigger conditions	[20]	Reduces oracle manipulation and flash-loan trigger risk	Does not ensure fair liquidation execution
Batch Auctions	Protocol	Uniform clearing price	[11]	Reduces ordering-based advantages and sniping incentives	Rarely applied in liquidation systems
Proposed Architecture	Protocol	Commit-reveal batch liquidation with guarded oracle inputs	This dissertation / [23]	Reduces bid visibility, mitigates liquidation-phase MEV, combines trigger-layer and execution-layer protections	Added latency, more complex auction flow, prototype-level evaluation

As shown in Table 1, most existing approaches address either transaction ordering at the infrastructure layer or price manipulation at the trigger layer, but not both simultaneously within the liquidation mechanism itself. The architecture proposed in this dissertation combines guarded oracle inputs with a commit–reveal batch auction, thereby addressing both liquidation eligibility and liquidation execution within a unified protocol-level design. Its main strength lies in reducing information leakage and ordering-based extraction without relying on private relays or consensus-layer modifications. Its main trade-offs are additional latency, increased interaction complexity, and the practical limitations associated with prototype-level evaluation.

Rather than treating these approaches as competing alternatives, the proposed design integrates their complementary strengths: oracle guards stabilize liquidation eligibility, while commit–reveal batch auctions enforce fairness during liquidation execution. This synthesis positions the protocol to address MEV not as an external nuisance, but as a mechanism-design problem that can be mitigated directly within the liquidation logic. Accordingly, the remainder of this dissertation focuses on designing and evaluating a liquidation mechanism that is MEV-aware by construction, rather than MEV-tolerant by competition.

2.7 Research Hypothesis: MEV-Resistant CDP Liquidations

The central hypothesis of this dissertation is that commit–reveal auction mechanisms can enhance MEV resistance and fairness in DeFi liquidations.

In a traditional auction, all bids are publicly visible as they are submitted, enabling frontrunners to react instantly. In contrast, a commit–reveal model introduces a two-phase bidding process:

1. Commit Phase: Participants submit cryptographic commitments to their bids (e.g., hashed commitments combining the bid amount and a secret nonce) without revealing actual bid details.

2. Reveal Phase: After a predefined delay or block interval, participants reveal their original bids and corresponding secrets, which the smart contract verifies against their commitments.

This separation between commitment and revelation disrupts the ability of MEV bots to frontrun bids, as the actual values remain concealed during the critical period when block proposers determine transaction order. By preventing immediate visibility, the commit–reveal model reduces the predictability of liquidation outcomes and neutralizes opportunistic transaction sequencing.

The research hypothesizes that implementing such a mechanism in a CDP liquidation system will:

- Mitigate frontrunning and sniping behavior by obscuring actionable bid information.
- Promote fairness and inclusivity by giving all participants equal opportunity to compete.
- Preserve protocol efficiency while introducing minimal latency.
- Serve as a proof-of-concept for broader MEV-resistant DeFi mechanisms beyond liquidations.

This design philosophy shifts the focus from off-chain MEV redistribution (as seen in Flashbots) to on-chain MEV prevention, embedding fairness directly into the protocol architecture.

3.System Architecture and Design

3.1 Design Goals and Principles

The architecture of the prototype CDP platform sets out to satisfy a set of rigorous design objectives. First, **modularity**: each core functional domain, vault management, stability pooling, liquidation auctions, price-oracles, governance, is implemented as a distinct module (smart-contract or contract suite) with clearly defined interfaces, enabling maintainability, extension and replacement of components without monolithic coupling. Second, **security**: the system adopts standard and advanced smart contract hardening (e.g., reentrancy guards, careful state transitions, validated oracle inputs) to reduce attack surface. Third, **transparency**: the protocol is designed to operate on an EVM-compatible chain, with events and state transitions available on chain, facilitating auditability and trust. Fourth, **MEV resistance**: the liquidation subsystem is engineered to reduce the exploitability of transac-

tion-ordering, frontrunning and sniping by adopting a commit–reveal auction mechanism and additional anti-MEV mitigations. Fifth, **extensibility for multi-collateral support**: although the prototype is built for demonstration, the architecture permits additional collateral token types to be plugged into the system via parameterization (collateral LTVs, risk-weights, permitted token registry) without redesigning core flows. Inherent in these goals are **trade-offs** between usability, performance and fairness. For usability, the system must provide a relatively intuitive user flow for vault creation, borrowing and repayment; yet implementing MEV-resistant auctions tends to introduce added complexity (commit phases, reveal phases) that may degrade user experience and increase latency. On the performance side, instant liquidations (e.g., simple keepers racing) are faster and more gas-efficient, but they sacrifice fairness by favoring bots with superior infrastructure. By contrast, fairness (i.e., equitable access for ordinary users, reduction of bot dominance) is improved via cryptographic hiding and batching but at the cost of latency and slightly higher gas overhead. The architecture therefore prioritizes **fairness and resistance to MEV** over minimal latency, recognizing that within a high-risk DeFi liquidation context, the integrity of the mechanism is at least as important as raw speed. The extensibility goal further adds complexity, supporting multiple collateral types means dynamic parameterization, registry mechanisms and more sophisticated risk-management flows, which can reduce simplicity. The design balances these aspects by decoupling modules, minimizing assumptions and choosing commit–reveal auction logic that trades a small latency cost for improved fairness.

3.1.1 Design Assumptions and Adversarial Capabilities

The architectural design of the proposed CDP protocol is grounded in an explicit adversarial model and a set of design assumptions consistent with contemporary Ethereum-based DeFi systems. These assumptions define the threat boundaries within which the protocol aims to provide security and MEV resistance.

Design Assumptions. The protocol assumes execution on an Ethereum-compatible, permissionless blockchain with a public mempool and probabilistic transaction ordering. Smart contracts are assumed to execute correctly according to the EVM specification and Solidity ^0.8.x safety guarantees (e.g., checked arithmetic) are relied upon. Governance and administrative roles (e.g., oracle administration, risk parameter configuration) are assumed to be controlled by honest operators or secured via multisignature wallets and, in production settings, time-locked execution. External price feeds are assumed to be eventually consistent and economically honest within the bounds enforced by oracle guards (staleness, deviation thresholds and TWAP).

Adversarial Capabilities. Adversaries are assumed to have full visibility of the public mempool and the ability to submit, reorder, or bundle transactions using elevated gas fees or private relay infrastructure. Attackers may deploy automated MEV searchers, leverage flash loans for short-term liquidity and attempt frontrunning, backrunning, liquidation sniping, or copy-bidding strategies. Validators or block builders may exhibit inclusion bias or limited censorship within individual blocks. However, the model does not assume long-term global censorship, control of governance roles, or compromise of cryptographic primitives.

Within this threat model, the system architecture prioritizes MEV resistance and liquidation fairness by minimizing actionable information leakage, decoupling bid submission from bid revelation and enforcing deterministic settlement rules. These assumptions inform the design decisions presented in this chapter and are evaluated empirically in the security analysis of Chapter 6.

3.2 High-Level Architecture Overview

The protocol is implemented as a modular smart contract system, where distinct components encapsulate vault management, liquidation logic, liquidity backstopping and price discovery. This separation of concerns reduces cross-module coupling but introduces multiple externally callable entry points that collectively define the system's attack surface. From a security perspective, these entry points represent the interfaces through which adversaries may at-

tempt to manipulate state, extract value, or disrupt liveness. The following analysis enumerates the sensitive functions across core modules and outlines the primary risk vectors and corresponding mitigations.

Figure 3 presents a high-level overview of the protocol architecture.

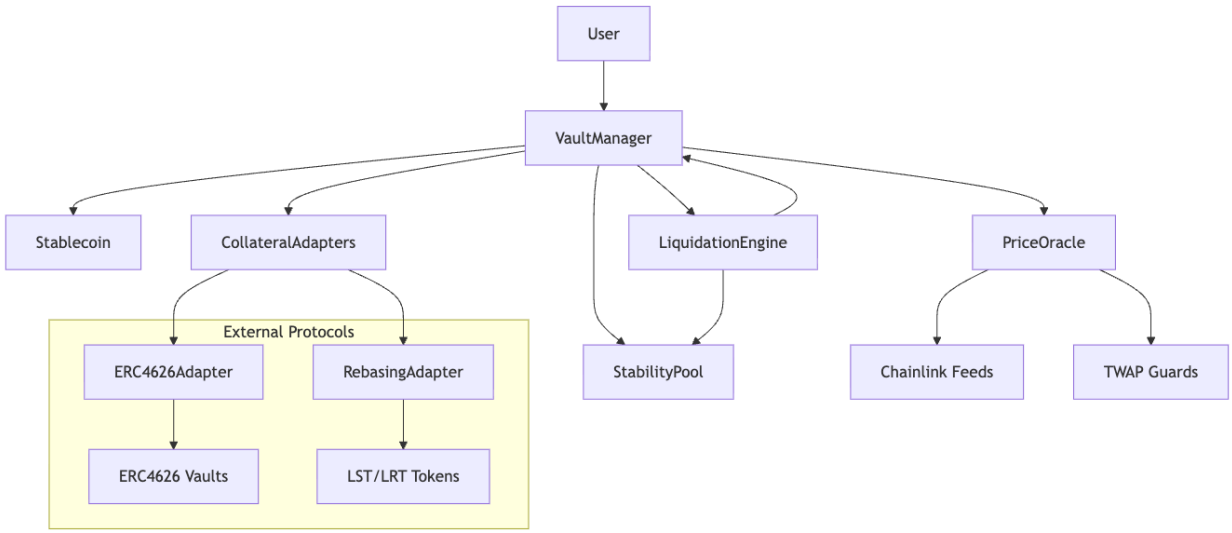

Figure 3 High-level system architecture showing core protocol components and their interactions

The architecture of the proposed protocol can be understood through three complementary views: (i) the actors that interact with the system, (ii) the core components that implement its functionality, and (iii) the roles and responsibilities of these components.

(i) Actors.

The primary actors are the user, liquidators, governance, and external oracle providers. The user interacts with the system through the VaultManager in order to deposit collateral, mint stablecoins, repay debt, and withdraw collateral subject to collateralization constraints. Liquidators participate when vaults become undercollateralized, submitting bids through the LiquidationEngine in order to acquire collateral and restore protocol solvency. Governance is responsible for configuring protocol parameters, such as collateral types, liquidation thresholds, auction settings, and oracle constraints. External oracle providers supply price data that the protocol uses to evaluate vault health and trigger liquidation conditions.

(ii) Core Components.

The core protocol components are VaultManager, LiquidationEngine, StabilityPool, PriceOracle/GuardedOracle, Stablecoin, and CollateralAdapters. The VaultManager is the main coordination module, maintaining vault state and enforcing borrowing and repayment rules. The LiquidationEngine handles the commit–reveal auction process for undercollateralized positions. The StabilityPool acts as a liquidity backstop, absorbing residual debt when auctions do not fully clear. The PriceOracle and GuardedOracle provide validated price inputs, including protections such as time-weighted averages and deviation checks. The Stablecoin contract represents the debt asset issued against

collateral. Finally, the CollateralAdapters enable the protocol to support heterogeneous collateral types, including ERC-4626 vault assets and rebasing tokens, through a unified interface.

(iii) Interaction Model.

As shown in Figure 3, these actors and components interact through a modular architecture in which responsibilities are clearly separated. This separation of concerns improves extensibility and auditability, while also limiting the attack surface of each module.

Figure 4 illustrates the primary interaction flows between users, protocol modules and liquidators.A user opens and manages a vault through the VaultManager, which consults the GuardedOracle for pricing and enforces collateralization constraints. Under normal conditions, users may borrow, repay and withdraw collateral subject to health-factor checks. When a vault becomes undercollateralized, the LiquidationEngine coordinates a commit–reveal auction, optionally backed by the StabilityPool as a liquidity backstop. Governance configures system parameters and collateral policies but does not participate in day-to-day execution. The figure summarizes these flows, while the text highlights the security-relevant interactions.

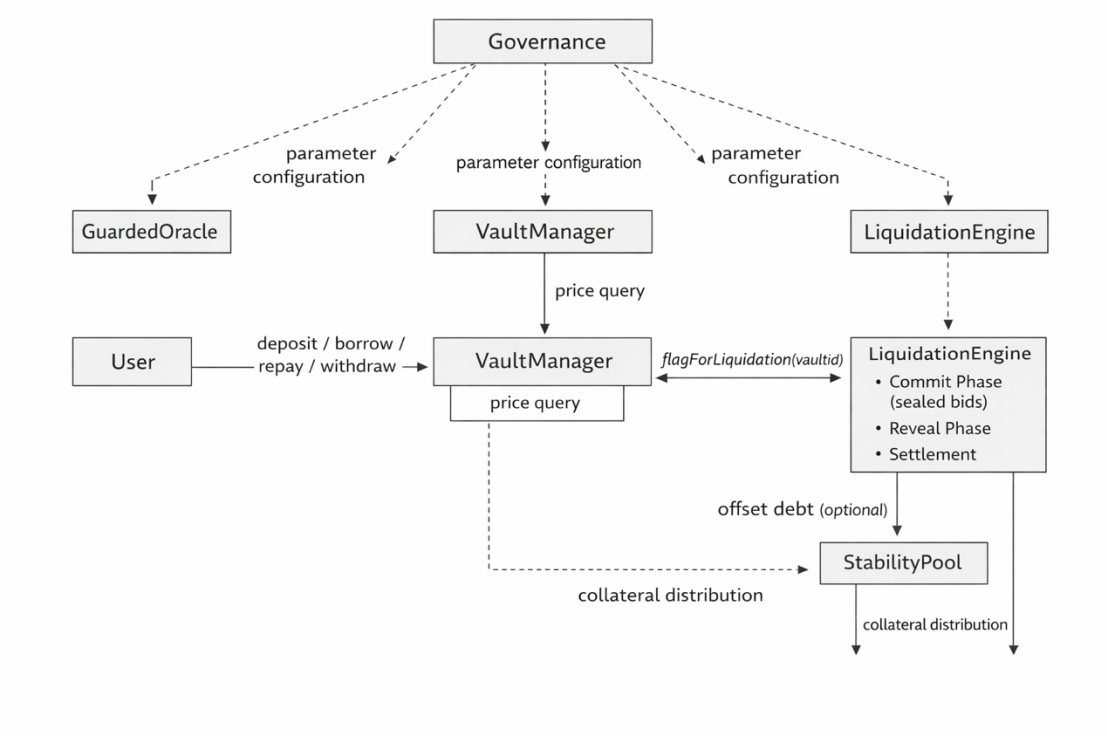

Figure 4 User flows and module interactions in the proposed CDP protocol

3.3 Smart Contract Components

The proposed CDP protocol is implemented as a modular suite of Ethereum smart contracts, each responsible for a clearly defined aspect of system functionality. This design follows a separation-of-concerns principle, enabling in-

dependent reasoning about collateral management, liquidation logic, oracle security [22] , liquidity backstopping and governance. The following descriptions outline the core contracts that compose the system, focusing on their responsibilities, interactions and security-relevant behaviors. Rather than treating these components in isolation, the discussion emphasizes how their coordinated operation enables MEV-resistant liquidations within a multi-collateral architecture.

VaultManager (Collateral Management and Debt Issuance). The VaultManager contract is the primary entry point for user collateral actions and debt issuance. It manages the vault lifecycle: creation, collateral deposit/withdrawal, debt borrowing/repayment, health factor monitoring and triggering liquidation eligibility. Collateral tracking is performed by maintaining mappings from vault IDs to collateral balances for each approved token. The contract queries the GuardedOracle for real-time or TWAP (time-weighted average price) quotes to compute the collateral value in stablecoin terms. Borrowing logic enforces a maximum Loan-to-Value (LTV) ratio: debt ≤ collateral_value × LTV_limit. The health factor is computed as collateral_value / (debt + fees) and the liquidation threshold is typically >1 (for example health < 1 implies liquidation). If collateral value drops, VaultManager marks the vault as "unsafe" and signals the LiquidationEngine. VaultManager also collects stability fees (interest) on outstanding debt, configured via governance, which accrue until repayment. Upon repayment, the vault's debt balance decreases and the user can withdraw collateral as long as the position remains safe. Furthermore, withdrawals are only allowed when the health factor remains above a safety margin, ensuring protocol solvency. The design ensures that the vault state is consistently monitored and that collateral value is updated before executing sensitive transitions (deposit, withdraw, borrow, repay). Importantly, VaultManager delegates liquidation responsibility rather than executing immediate on-chain keepers, thus enabling the commit–reveal auction model.

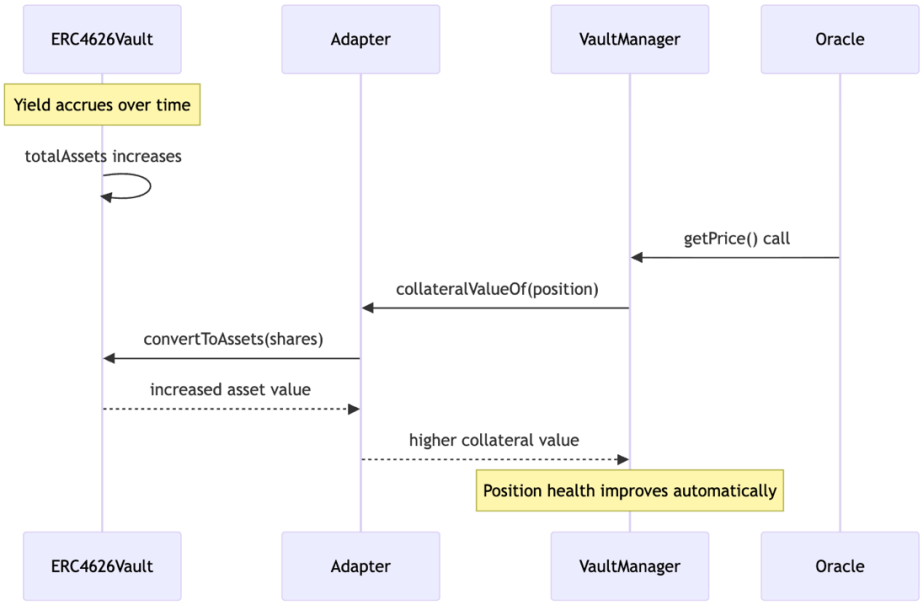

Figure 5 Yield accrual flow for ERC-4626 collateral via adapter-based valuation

StabilityPool (Debt Offsetting and Collateral Absorption). The StabilityPool acts as a **global reserve** of stablecoins (or designated liquidity tokens) into which protocol participants deposit funds. Its purpose is to **absorb bad**

debt in the event of liquidations when external bidder auctions do not fully clear positions. When a vault is liquidated and the auction fails (or per design), the StabilityPool supplies the stablecoin amount needed to repay the outstanding debt. In exchange, pool participants receive the collateral asset from the liquidated vault, pro-rata to their deposit share and often at a discount (liquidation penalty and incentive). This mechanism maintains system solvency while providing depositors with an incentive to contribute to the pool. In the implementation from the repository, the StabilityPool tracks deposits, updates shares at each offset event and distributes collateral using internal accounting. The interaction is typically: LiquidationEngine transfers unsettled vault debt to StabilityPool.offsetDebt(vaultDebt) and then triggers transfer of the vault's collateral to pool participants. The StabilityPool may also receive surplus bid funds if external bidders overpay and returns excess to the protocol treasury or distributes to depositors depending on design. The modular design allows participants to withdraw their deposits subject to pool conditions and ensures that the pool is always aware of outstanding offsets.

LiquidationEngine (Commit–Reveal Auction Process). The LiquidationEngine is the centerpiece of the MEV-resistant design. It executes a **commit–reveal auction process** for liquidating under-collateralized vaults. The process is composed of three distinct phases:

Commit Phase: Upon notification by VaultManager, the LiquidationEngine opens a bidding window in which prospective bidders submit hashed commitments of the form hash(bidAmount, nonce, vaultId), along with a bond deposit (to discourage frivolous bids). At this stage, the actual bid amounts remain concealed onchain, preventing bots from monitoring mempool bid values and reacting.

Reveal Phase: After a defined block-interval or time window, the system enters the reveal phase. Bidders submit their plaintext bids plus nonces. The contract verifies that hash(bidAmount, nonce, vaultId) matches the earlier commitment. Invalid or missing reveals forfeit the bond.

Settlement Phase: The highest valid revealed bid is selected, the bond is refunded (minus possible penalty if reveal was missing or late) and the vault's collateral is transferred to the winning bidder. The bid amount (stablecoin) is applied to repay the vault's outstanding debt and fees via the VaultManager. Any remainder is returned to the vault owner and any deficit triggers the StabilityPool to absorb the remainder. The system then closes the vault and updates state.

This commit–reveal mechanism directly addresses MEV risks: by hiding bid amounts during the commit phase, bidders and adversaries cannot optimise submission ordering or gas fees based on visible information; they cannot frontrun by observing competitor bids or anticipate winner behaviour. The bond ensures genuine participation and penalises delayed reveals, while the multi-block phases widen the time horizon, reducing the advantage of single-block robots. Moreover, the architecture may impose a **uniform clearing price** (winning bid sets price) which dis-incentivises bid sniping and fosters competitive fairness. The repository implementation also includes logic to handle tie bids (e.g., earliest reveal) and fallback to StabilityPool if no valid bids are received.

GuardedOracle (Price Feed and Security Layer). Accurate and secure price data is critical for health factor calculation and safe liquidations. The GuardedOracle module interfaces with one or more external data feeds (e.g., Chainlink, Uniswap TWAP) and wraps them with security checks. Key functions include obtaining TWAP values over a sliding window, detecting price deviations beyond configured thresholds and enforcing **circuit breakers** to pause or restrict operations if a feed appears compromised or outlier. For example, if the oracle reports a sudden price movement that exceeds a deviation threshold in a short interval, the GuardedOracle may invalidate the feed, forcing operations to wait for stability.

In the repository, GuardedOracle exposes functions such as getPrice(token) and isValid(token), with internal logic verifying that abs(latestPrice – twapPrice) / twapPrice ≤ deviationThreshold. On invalid state, the contract prevents

vault creation or new borrowings and flags potential liquidation events for external review. This additional layer reduces risks of **oracle manipulation** or **flash-loan attack price spikes**, which often contribute to MEV extraction in liquidations. The modular design ensures the oracle logic is kept separate from vault/liquidation logic, preserving separation of concerns and upgradeability.

Governance and Parameterization. The Governance module allows authorised actors (or decentralized governance) to manage protocol parameters in a controlled manner. Key adjustable parameters include: approved collateral token registry, LTV limits, liquidation thresholds, liquidation penalties (discounts/incentives for bidders), bond sizes for auctions, auction durations (commit and reveal block windows), stability pool rewards and withdrawal constraints. The architecture in the repository implements Governance.setCollateralConfig(token, ltv, threshold, penalty), Governance.setAuctionParams(bondSize, commitBlocks, revealBlocks) and Governance.setOracleConfig(token, deviationThreshold, twapWindow). All governance calls emit events and include access controls (e.g., only governance role). This design provides protocol flexibility and adaptability to evolving risk environments, while retaining modular separation. Because changes are parameter based rather than code rewrites, the multi-collateral/extensibility goal is preserved.

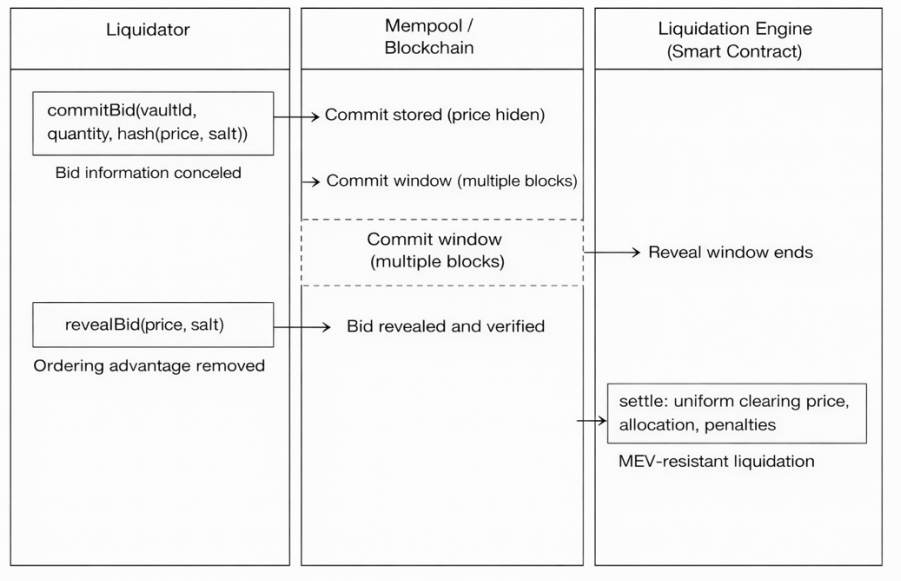

Figure 6 Commit-reveal liquidation workflow in the proposed protocol

3.4 Sequence Diagrams and Interaction Flows

Although a full UML diagram is not included here, the following narrative outlines typical interaction flows.

Vault Creation & Borrow Flow:

1. User approves collateral token and calls VaultManager.deposit(token, amount).
2. VaultManager invokes GuardedOracle.getPrice(token) to fetch the current price.
3. Collateral value computed; vault recorded with collateral[token] = amount and health factor computed.

4. User then calls VaultManager.borrow(amount). VaultManager checks amount ≤ collateral_value × LTV_limit. Debt is issued (stablecoin minted) and assigned to vault.

5. User may call VaultManager.withdraw(amount) or VaultManager.repay(amount) as long as resulting health factor remains valid.

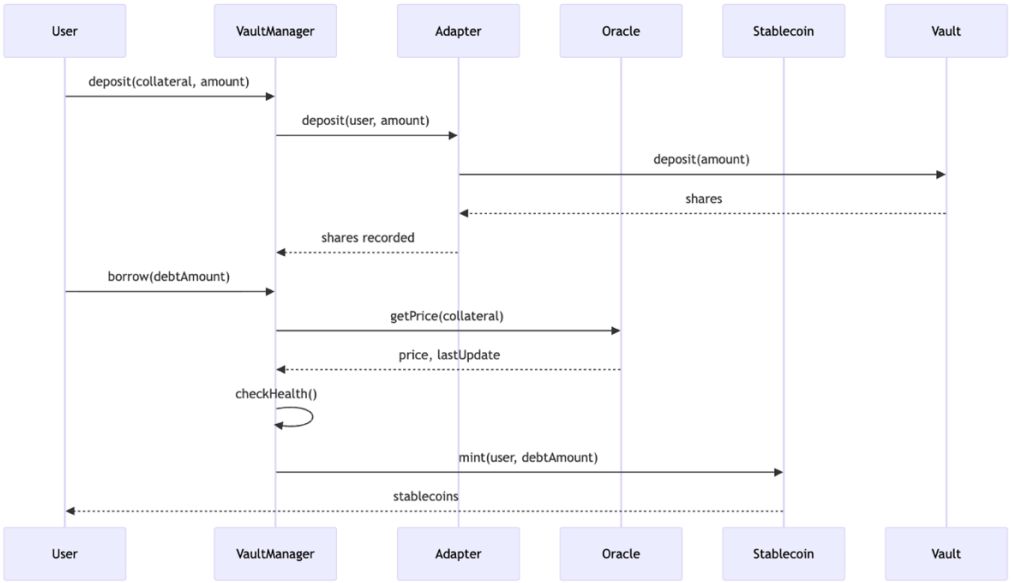

Figure 7 Sequence diagram for collateral deposit and stablecoin borrowing.

Liquidation Flow (Commit–Reveal Auction):

1. VaultManager detects health factor < liquidation threshold, flags vault for liquidation and calls LiquidationEngine.initiate(vaultId).

2. LiquidationEngine enters **Commit Phase**: bidders submit commitHash = hash(bid, nonce, vaultId) and bond deposit.

3. After commitBlocks, the engine enters **Reveal Phase**: bidders submit (bid, nonce); contract verifies hash matches and records valid bids.

4. After revealBlocks, Settlement: highest bid selected. LiquidationEngine instructs VaultManager to close vault, transfer debt settlement and collateral transfer. If no valid bids, the engine triggers StabilityPool.offsetDebt(debt) and StabilityPool.receiveCollateral(collateral).

5. Bond returned to winner (minus any penalty if applicable); events emitted signalling auction result.

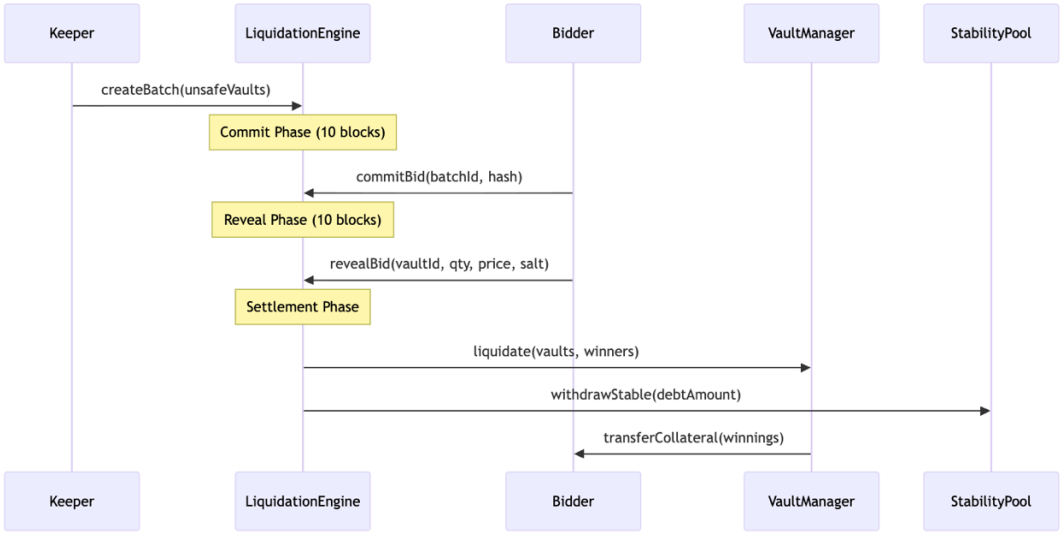

Figure 8 MEV-resistant liquidation sequence using a commit–reveal auction.

Multi-block batching and hidden bid amounts ensure that no participant can adjust their bid or gas fee in response to another bid in the same block; this design reduces the possibility of frontrunning and sniping, since bid values are concealed until reveal and transaction accrual occurs across multiple blocks.

Because modules call each other via predefined interface methods rather than direct internal state manipulations, state changes are transparent and auditable. For example, LiquidationEngine.settle(vaultId, winnerBid) triggers VaultManager.closeVault(vaultId) and either StabilityPool.apply(winnerBid, collateral) or collateral transfer to bidder.

3.5 MEV-Resistance Design Considerations

Key to the research goal of an MEV-resistant CDP design is the architecture's ability to minimise extractable value opportunities inherent in liquidation processes. The commit–reveal auction is chosen over traditional instant liquidation models because it decouples bid submission from bid revelation, thus obscuring actionable information that bots exploit. In a standard auction where bid amounts are visible immediately, bots monitor mempool, detect large bids and selectively frontrun or backrun. In contrast, concealment of bid amounts prevents adversaries from knowing the range of other bids and gas bidding wars become ineffective.

Additionally, the design introduces bonds as an economic deterrent: bidders must lock a deposit when committing, which they forfeit if they fail to reveal properly or withdraw prematurely. This discourages spurious commits and reduces participation by automated whips. The use of uniform clearing price logic (i.e., winner pays their bid or second-price style variant) disincentivises extremely aggressive frontrunning bids simply to preempt competition. The multi-block phases create time uncertainty: since the reveal occurs after multiple blocks, single-block bots cannot reliably predict and pre-empt bids or dominate the process.
Beyond the auction itself, supporting security layers reduce ancillary MEV vectors: reentrancy guards prevent loop-based exploitation; flash-loan resistance in the VaultManager ensures extreme borrowing to manipulate

health-factor is mitigated; and the GuardedOracle's deviation & circuit breaker logic prevents sudden oracle manipulations that often trigger profitable liquidations for bots.
From a research perspective, the architecture realises the theoretical premise: that embedding MEV resistance at the protocol level (rather than relying solely on off-chain relays or consensus modifications) is feasible and can materially improve fairness in DeFi liquidation systems. By decoupling bidding from transaction ordering visibility, the protocol reduces extractable rent and invites a wider class of participants to engage. While performance trade-offs exist (e.g., slightly longer settlement time, modest gas overhead for commit/reveal), the fairness and resilience gains align with the design goal of a more equitable CDP platform.

4.Implementation Details

This chapter documents the concrete implementation of the proposed MEV-resistant CDP protocol, translating the architectural design of Chapter 4 into executable smart contracts. The complete implementation, including contracts, tests, deployment scripts and documentation, is publicly available in the project's GitHub repository [23]

The focus of this chapter is on implementation decisions that are directly relevant to protocol security, correctness and resistance to Maximal Extractable Value (MEV). Standard Solidity patterns, boilerplate constructs and well-established framework usage are intentionally omitted or summarized, except where they materially influence safety properties, attack surfaces, or liquidation fairness. Emphasis is placed on contract boundaries, role separation, oracle hardening, liquidation logic and testing strategies that collectively enforce the protocol's security and MEV-resistance guarantees.

4.1 Development Environment and Tools (Solidity, Hardhat, Foundry)

All contracts are authored in Solidity ^0.8.24 with the compiler optimizer enabled (runs = 200) under a *Cancun-compatible* EVM target. This configuration ensures deterministic bytecode and stable gas characteristics across runs. The project uses Hardhat v3.0.6 as the primary framework for compilation, testing, deployment and scripted simulation.

Hardhat is extended with a minimal but rigorous plugin set aligned to the repository's needs:

@nomicfoundation/hardhat-viem for first-class Viem integration, supporting typed, modern RPC interactions and deterministic local simulations.

@nomicfoundation/hardhat-mocha to expose Mocha lifecycle hooks and structured reporting within Hardhat's runtime.

hardhat-deploy for reproducible deployments, named accounts and per-network artifact management.

@nomicfoundation/hardhat-ignition for optional structured "Ignition" deployment modules.

Networks are defined as: hardhat (ephemeral, in-memory), localhost (persistent local node) and sepolia (public Ethereum testnet, parametrized via environment variables). Named accounts standardize operational roles for deployment and tests: deployer (index 0), admin (index 1), treasury (index 2) and liquidator (index 3).

The TypeScript-based test and script stack comprises TypeScript ~5.8.0, Mocha ^10.8.2, Chai ^5.3.3 and Viem. Code quality and linting are enforced by ESLint ^8.57.1, Prettier ^3.6.2 and Solhint ^6.0.1. Key npm scripts provide a uniform interface:

compile, test, test:integration, test:security, deploy:localhost, node and simulate.

This toolchain directly supports reproducibility (pinned compiler and plugins; hardhat-deploy fixtures; named accounts), rigorous testing (typed interactions via Viem; controlled clock/time manipulation on Hardhat Network) and repeatable deployment (scripted order, post-configuration and verification scripts).

```json
"compile": "hardhat compile",
"test": "hardhat test",
"test:integration": "hardhat test test/full-integration.ts test/*integration*.test.ts",
"test:security": "hardhat test test/Security.*.test.ts",
"deploy:localhost": "hardhat deploy --network localhost"
```

Figure 9 Excerpt of npm scripts used for compilation, testing and deployment

4.2 Smart Contract Development Process

The implementation adheres to a **modular, interface-first architecture**. Logical domains are separated across packages, vault management (/vault), liquidation (/liquidation), stability backstop (/stability), oracles (/oracles), adapters (/adapters) and shared math (/lib). Inter-module interactions occur through **narrow interfaces**, enabling substitution and test mocking without increasing coupling.

Adapters (e.g., ERC4626Adapter, RebasingAdapter) unify heterogeneous collateral behaviors behind the ICollateralAdapter interface. Oracles are wrapped by GuardedOracle, which layers staleness, bounds, TWAP and pause semantics, with PriceOracle providing an abstract base for specialized feeds.

Architecture Pattern:

```
interface ICollateralAdapter {
    function deposit(address user, uint256 amount) external returns (uint256 shares);
    function withdraw(address user, uint256 shares) external returns (uint256 amount);
    function collateralValueOf(address user) external view returns (uint256 value);
    function totalCollateral(address user) external view returns (uint256 amount);
}
```

ERC4626Adapter

- **Purpose:** Support auto-compounding yield vaults
- **Mechanism:** Converts deposits to vault shares, yield automatically improves health
- **Examples:** Aave aTokens, Compound cTokens, Yearn vaults

RebasingAdapter

- **Purpose:** Support rebasing tokens (LSTs/LRTs)
- **Mechanism:** Normalizes rebases through share-based accounting
- **Examples:** stETH, rETH, frxETH

Figure 10 Collateral adapter interface for standard and rebasing tokens.

Governance and roles employ OpenZeppelin AccessControl. The repository defines operational and administrative roles mapped to the named accounts: DEFAULT_ADMIN_ROLE (administration), RISK_ADMIN (collateral configuration), PAUSER_ROLE (emergency halt), LIQUIDATOR_ROLE (queue and batch operations), ORACLE_ADMIN (price updates and guard parameters) and EMERGENCY_ROLE (stability pool emergency flows). Critical contracts are Pausable to facilitate controlled shutdown under incident conditions.

Coding conventions emphasize correctness and gas efficiency:

Custom errors replace revert strings for lower gas and consistent error taxonomy.

ReentrancyGuard wraps state-changing entry points.

Checks-Effects-Interactions ordering is respected; external calls follow state updates.

Explicit modifiers (e.g., onlyVaultOwner, vaultExists, vaultActive, validAmount) encode preconditions at function boundaries.

WadMath standardizes 18-decimal fixed-point operations (mulWad, divWad), avoiding precision drift.

The **workflow** is disciplined and iterative: (i) interface and invariant specification; (ii) incremental implementation; (iii) unit, integration and security testing; (iv) refactoring with gas benchmarking and documentation synchronization. Dependencies (OpenZeppelin, Hardhat plugins, types) are pinned for deterministic builds; test and deployment artifacts are recorded per network for traceability.

4.3 Core Contract Logic and Key Functions

Stablecoin.sol

Implements the protocol-native ERC-20 stablecoin with role-based minting and burning. MINTER_ROLE and BURNER_ROLE are granted exclusively to VaultManager, enforcing monetary discipline under the principle of least privilege. The constructor assigns DEFAULT_ADMIN_ROLE to the administrative account. There is no user-accessible mint path. Security is derived from AccessControl checks and the absence of privileged externalizers.

Table 2 Core Functions of Stablecoin.sol

Function	Description
mint(to, amount)	Mints protocol stablecoins to a specified address. Callable only by accounts holding MINTER_ROLE (assigned to VaultManager).
burn(from, amount)	Burns stablecoins from a specified address. Callable only by accounts holding BURNER_ROLE, enforcing controlled debt repayment and liquidation flows.

VaultManager.sol

Acts as the central CDP coordinator. It references Stablecoin and GuardedOracle, maintains nextVaultId and tracks vault ownership, collateral type and per-vault positions. Key constants include MCR_BPS = 12000 and MAX_BPS = 10000, with WAD precision via PRECISION_MULTIPLIER = 1e18.

Collateral configuration (mapping(bytes32 => CollateralConfig)): adapter address, ltvBps and enable flag.

Position state (Position): shares (collateral shares), debt (WAD) and active flag.

Table 3: Core Functions of VaultManager

Function	**Description**
createVault(collateralType)	Creates a new vault for an enabled collateral type and assigns ownership to the caller.
setCollateral(key, adapter, ltvBps, enabled)	Configures collateral parameters (adapter, LTV, enabled flag). Restricted to risk administrators.
deposit(vaultId, assets)	Deposits collateral into a vault via the configured adapter and credits shares.
withdraw(vaultId, shares, receiver)	Withdraws collateral shares from a vault, enforcing post-withdraw health checks.
borrow(vaultId, amount)	Mints stablecoins against vault collateral, enforcing loan-to-value constraints.
repay(vaultId, amount)	Burns stablecoins to reduce outstanding vault debt.
vaultHealth(vaultId)	Returns collateral value, debt, LTV and health status of a vault.
flagForLiquidation(vaultId)	Flags an undercollateralized vault and enqueues it for liquidation.
onLiquidationSettle(vaultIds, filledQty,	Finalizes liquidation by updating vault balances after auction

clearingPrice)	settlement.
pause() / unpause()	Emergency controls to suspend or resume protocol operations.

```
struct Position {
    uint256 collateralAmount;    // Adapter-specific units (shares)
    uint256 debtAmount;          // Stablecoin debt
    address collateralType;      // Adapter contract address
    uint256 lastUpdate;          // Timestamp of last modification
}

struct CollateralConfig {
    uint256 minCollateralRatio;  // MCR (e.g., 150%)
    uint256 liquidationFeeRate; // LFR (e.g., 10%)
    uint256 debtCeiling;         // Max debt for this collateral
    uint256 maxLTV;              // Max loan-to-value ratio
    bool enabled;                // Active/inactive flag
}
```

Figure 11 Core data structures in VaultManager.sol for vault positions and collateral configuration.

LiquidationEngine.sol

Implements a commit–reveal batch auction with uniform clearing price. Immutable parameters include:

COMMIT_WINDOW = 300 seconds, REVEAL_WINDOW = 300 seconds,

minCommitBond = 0.1 ETH,

minLot, maxBatchSize,

references to IVaultManager and IStabilityPool.

Batch state captures commit/reveal timing, participating vaultIds, totalQtyOffered, clearingPrice, revealed bids and settlement status. Bid state records bidder, vaultId, qty, price, saltHash and validity metadata. Core functions:

enqueue(vaultId) places a vault into the liquidation queue (role-gated).

startBatch() forms a batch up to maxBatchSize.

commitBid(batchId, commitment) records a sealed commitment with an accompanying bond.

revealBid(batchId, vaultId, qty, price, salt) validates the hash and records a revealed bid.

settle(batchId) computes the clearing price, performs allocation, repays debt via VaultManager, transfers collateral and refunds/slashes bonds.

Security: time windows, economic bonds, reentrancy guards, Pausable, role-gated queue operations and bounded loops over batch participants.

Table 4: Core Functions of LiquidationEngine.sol

Function	**Description**
enqueue(vaultId)	Adds a vault to the liquidation queue (role-gated). Prevents duplicate queueing via vaultQueued.
startBatch()	Forms a new liquidation batch from queued vaults (up to maxBatch-Size) and initializes commit/reveal timestamps.
commitBid(batchId, commitment)	Records a sealed bid commitment during the commit window and escrows the required bond (minCommitBond).
revealBid(batchId, vaultId, qty, price, salt)	Reveals bid parameters during the reveal window and verifies them against the stored commitment hash; refunds bond for valid reveals.
settle(batchId)	Finalizes the batch after the reveal window; slashes invalid/non-compliant bonds, derives the uniform clearing price and triggers settlement execution.
getBatch(batchId)	Returns batch metadata (timing, vault IDs, offered quantity, clearing price, settlement status).
getRevealedBidsCount(batchId)	Returns the number of bids revealed for a batch.
getRevealedBid(batchId, index)	Returns revealed bid details (bidder, vaultId, qty, price, validity).

isCommitPhase(batchId)	Indicates whether the batch is currently in the commit phase.
isRevealPhase(batchId)	Indicates whether the batch is currently in the reveal phase.
canSettle(batchId)	Indicates whether the batch is eligible for settlement (reveal window elapsed and not settled).

```
struct Batch {
    uint256[] vaultIds;           // Vaults to liquidate
    uint256 commitDeadline;       // End of commit phase
    uint256 revealDeadline;       // End of reveal phase
    BatchStatus status;           // COMMITTING, REVEALING, SETTLING, COMPLETE
}

struct Bid {
    address bidder;
    uint256 vaultId;
    uint256 quantity;             // Amount of collateral desired
    uint256 price;                // Price per unit
    bool revealed;
}
```

Figure 12: Batch and bid data structures in LiquidationEngine.sol.

StabilityPool.sol

Provides a liquidity backstop. State includes the stablecoin reference, per-depositor balances, totalDeposits and an emergencyMode flag. Public functions: deposit, withdraw, available, burnStableFrom (for liquidation-driven debt offset), credit (for distribution) and emergency operations (activateEmergencyMode, emergencyWithdraw). Roles: LIQUIDATION_ENGINE_ROLE, PAUSER_ROLE, EMERGENCY_ROLE. All state-changing paths are non-reentrant.

Table 5 Core Functions of StabilityPool.sol

Function	Description
deposit(amount)	Deposits protocol stablecoins into the stability pool, increasing user balance and total liquidity.
withdraw(amount)	Withdraws a user's deposited stablecoins under normal operating conditions.
burnStableFrom(from, amount)	Burns (or sequesters) stablecoins during liquidation settlement; callable only by the liquidation engine.
available()	Returns the current stablecoin balance available in the pool for liquidation coverage.
credit(to, amount)	Credits stablecoins to participants (e.g., liquidation rewards), role-restricted to the liquidation engine.
activateEmergencyMode()	Enables emergency mode, allowing withdrawals even when normal operations are paused.
emergencyWithdraw()	Allows users to withdraw their full balance during emergency mode.

```
struct StabilityDeposit {
    uint256 amount;          // Deposited stablecoins
    uint256 rewardSnapshot;  // For reward calculations
    uint256 depositTime;     // For time-based rewards
}
```

Figure 13: Stability deposit data structure for tracking liquidity and rewards.

GuardedOracle.sol and PriceOracle.sol

GuardedOracle implements a multi-layer price guard: maxStale staleness checks (default one hour), sanity bounds (minPrice, maxPrice), a 16-slot ring buffer for TWAP (twapWindow typically 30 minutes) and a circuit breaker (paused). It stores the latest spot price (PriceData) and ring buffer points (Point). getPrice(asset) returns a validated price; administrative functions (pushPrice, pushPriceAt, setBounds, setMaxStale, setPaused, setTwapWindow) are restricted to ORACLE_ADMIN.
PriceOracle supplies the abstract base: staleness and pause logic with _latestPrice() left to implementers.

Table 6 Core Functions of GuardedOracle.sol

Function	**Description**
setBounds(minP, maxP)	Configures acceptable price bounds; prices outside [minPrice, maxPrice] are rejected by getPrice.
setMaxStale(s)	Sets the maximum permitted staleness (seconds) for the latest price; enforces freshness in getPrice.
setPaused(p)	Enables or disables oracle output; when paused, getPrice reverts to protect dependent protocol logic.
setTwapWindow(seconds_)	Enables TWAP mode by defining the averaging window; 0 disables TWAP and returns spot.
pushPrice(asset, price)	Pushes a new spot price at block.timestamp, updates latest and records a TWAP ring-buffer point (admin/feeder path).
pushPriceAt(asset, price, ts)	Pushes a new spot price with a caller-specified timestamp (testing utility for staleness / window simulations).
getPrice(asset)	Returns (price, lastUpdate) after enforcing pause, staleness and bounds; returns TWAP if enabled, otherwise spot, with spot fallback if no points fall inside the TWAP window.

Table 7 Core Functions of PriceOracle.sol

Function	Description
constructor(_staleAfter)	Initializes the oracle base with a staleness threshold (staleAfter, in seconds).
setStaleAfter(s)	Updates the staleness threshold and emits SetStaleAfter; intended to be access-controlled by derived contracts.
setPaused(p)	Toggles pause state and emits SetPaused; intended to be access-controlled by derived contracts.
latestPrice()	Returns (price, updatedAt) from _latestPrice() and enforces pause and staleness checks (reverts on OraclePaused or StalePrice).
_latestPrice()	Abstract hook implemented by derived contracts to supply the raw price and timestamp.

```
contract GuardedOracle {
    mapping(address => PriceFeed) public feeds;
    mapping(address => TWAPConfig) public twapConfigs;

    struct PriceFeed {
        address chainlinkFeed;
        uint256 staleness;
        uint256 minPrice;
        uint256 maxPrice;
    }
}
```

Figure 14 Core data structures of the GuardedOracle contract.

ERC4626Adapter.sol and RebasingAdapter.sol

ERC4626Adapter wraps ERC-4626 [24] vaults and exposes asset, decimals, depositFrom, withdraw and valueOf. The immutable vault reference prevents adapter hijacking; valueOf reflects yield accrual, improving health over time.
RebasingAdapter normalizes rebasing tokens via share-based accounting controlled by an index, with functions mirroring the ERC-4626 adapter's interface; setIndex exists for controlled test simulation of rebases.

Table 8 Core Functions of ERC4626Adapter.sol

Function	**Description**
constructor(_vault)	Initializes the adapter with an immutable reference to the underlying ERC-4626 vault, preventing adapter hijacking.
asset()	Returns the address of the underlying asset managed by the ERC-4626 vault.
decimals()	Returns the decimals of the underlying asset via IERC20Metadata.
depositFrom(from, assets, receiver)	Transfers assets from the user, approves the ERC-4626 vault, deposits assets and returns minted shares; emits Deposited.
withdraw(shares, receiver, owner)	Redeems shares from the ERC-4626 vault into underlying assets and transfers them to receiver; emits Withdrawn.
valueOf(shares)	Returns the current asset value of a given number of shares using previewRedeem, reflecting yield accrual.

Table 9 Core Functions of RebasingAdapter.sol

Function	Description
constructor(_token)	Initializes the adapter with an immutable reference to the underlying ERC-20 token and caches its decimals.
setIndex(newIndex)	Updates the synthetic rebase index (testing utility) to simulate positive or negative rebasing; emits IndexSet.
asset()	Returns the address of the underlying rebasing token.
decimals()	Returns the decimals of the underlying token.
depositFrom(from, assets, receiver)	Transfers assets from the user, converts them to shares using the current index (assets / index), updates share accounting and emits Deposited.
withdraw(shares, receiver, owner)	Burns shares from the owner, converts them back to assets using the current index (shares * index), transfers assets to receiver and emits Withdrawn.
valueOf(shares)	Returns the current asset value of a given number of shares based on the index, reflecting rebase effects.

WadMath.sol

A minimal 18-decimal fixed-point library exposing mulWad and divWad, consistently used in ratios, LTV checks and settlement arithmetic.

Table 10 WadMath.sol

Function	Description
mulWad(a, b)	Multiplies two 18-decimal fixed-point values and scales the result down by 1e18, rounding toward zero. Used for value, ratio and collateral calculations.
divWad(a, b)	Divides one 18-decimal fixed-point value by another by scaling the numerator by 1e18 before division, rounding toward zero. Used for share, index and ratio computations.

4.4 Commit–Reveal Mechanism Implementation

The liquidation protocol operates over multi-vault batches to reduce single-target exploitability and amortize fixed gas costs.

Phase windows. Each batch transitions through:

Commit window (COMMIT_WINDOW = 300 s): bidders submit a sealed commitment and a bond ≥ minCommitBond (0.1 ETH).

Reveal window (REVEAL_WINDOW = 300 s): bidders reveal plaintext parameters; only timely, hash-consistent reveals are valid.

Settlement: executed after the reveal window, finalizing allocation and transfers.

Commitment scheme. Commitments are computed as keccak256(abi.encode(vaultId, qty, price, salt, bidder)), binding the bidder, target vault, quantity, price and secret salt. The engine records commitments[batchId][bidder] and bonds[batchId][bidder].

Reveal validation. revealBid(batchId, vaultId, qty, price, salt) re-computes the commitment; non-matching or **late** reveals are rejected. Valid reveals transition the bid to the batch's revealedBids and mark the commitment as revealed.

Uniform clearing price. At settle(batchId), the engine:

Aggregates total collateral supply (totalQtyOffered) of all batch vaults.

Sorts or scans revealed bids to derive the clearing price where cumulative demand meets or exceeds supply.

Allocates quantities:

If **undersubscribed**, each valid bid fills at its requested quantity at the clearing price.

If **oversubscribed**, allocations are **pro-rata** at the clearing price.

Applies the **clearing price to all winners**, eliminating pay-as-bid advantages and reducing strategic last-minute bid shading.

Settlement flow. For each winning allocation:

Stablecoin proceeds (at clearing price × allocated qty) are used to **repay batch vault debts** via VaultManager.onLiquidationSettle, applying penalties as configured at the vault layer.

Collateral is transferred from the vault(s) to winning bidders according to allocation.

Remaining **deficits** are offset via StabilityPool (by burning pool stablecoins and crediting collateral to depositors).

Bonds: valid reveals are refunded; **missing/invalid reveals** are **slashed** and routed to the treasury.

The batch is marked **settled** and all transient mappings are cleared to cap storage growth.

The **multi-block** nature of commit and reveal phases removes actionable price/quantity visibility from the mempool at decision time, while **bonds** deter griefing and mass spam commitments. The **uniform clearing price** prevents bid-sniping and dominance by high-frequency actors, reinforcing fairness.

4.5 Testing and Simulation Setup

Testing is organized under /test with TypeScript, Mocha, Chai and Viem on Hardhat Network. Files map one-to-one with functional and security domains:

Table 11 Breakdown of the test suite and verification objectives.

Test File	Verification Focus
Adapters.math.test.ts	Share↔asset conversion accuracy, decimal handling, rounding, edge constraints for ERC4626Adapter and RebasingAdapter.
Oracle.guards.test.ts	GuardedOracle staleness (maxStale), bounds (minPrice/maxPrice), TWAP ring buffer and paused circuit breaker; ORACLE_ADMIN authorization.
Stablecoin.auth.test.ts	MINTER_ROLE/BURNER_ROLE enforcement; admin role behaviors.
Security.access.test.ts	Access control across all contracts; RISK_ADMIN, LIQUIDATOR_ROLE, ORACLE_ADMIN, LIQUIDATION_ENGINE_ROLE.

Security.reentrancy.test.ts	Non-reentrancy of deposit/withdraw/borrow/liquidation paths; cross-function probes via malicious mocks.
Security.pause.test.ts	Emergency pause semantics in VaultManager, LiquidationEngine, StabilityPool; view-only availability; recovery.
FlashLoan.resistance.test.ts	TWAP-based rejection of single-block price shocks; staleness gating; multi-block commit–reveal preventing atomic manipulation.
Liquidation.commitReveal.test.ts	Commitment hashing inputs (vaultId, qty, price, salt, bidder); bond requirements; invalid/late reveal handling; bond refund/slash routing.
Liquidation.clearingPrice.test.ts	Clearing-price derivation; oversubscription pro-rata; equal-price and extreme-spread edge cases; no-valid-bid scenario.
Liquidation.integration.test.ts	End-to-end: price drop → flagForLiquidation → enqueue/startBatch → commit → reveal → settle; handoff to StabilityPool if required.
Vault.flows.4626.test.ts	ERC-4626 collateral lifecycle: deposit, health, borrow, yield simulation via Mock4626Vault.simulateYield, repay, withdraw.
Vault.flows.rebasing.test.ts	Rebasing token lifecycle: deposit, index growth via RebasingAdapter.setIndex, health evolution, borrow/repay/withdraw.
full-integration.ts	Protocol deployment order, role assignment, multi-collateral operations, liquidation participation, stability pool rewards.
Gas.benchmarks.test.ts	Gas profiling of core and liquidation operations; batch scaling behavior.
smoke.test.ts	Baseline deployment and sanity checks.

Notable scenarios.
— **Flash-loan resistance**: DEX price manipulation within a single block fails TWAP and staleness constraints; liquidation attempts at manipulated prices revert.
— **Reentrancy**: malicious ERC-4626 mocks attempting to reenter borrow or settlement paths are rejected by nonReentrant.
— **Access control**: unauthorized calls to setCollateral, pushPrice, or enqueue revert consistently.
— **Commit–reveal edge paths**: incorrect salts or altered quantities/prices are detected; unrevealed commitments are slashed.
— **Oversubscription**: allocation is pro-rata at the clearing price; accounting preserves conservation across debt repayment and collateral transfer.

Gas benchmarks (from Gas.benchmarks.test.ts, approximate medians):

- deposit() (ERC-4626): ~150k; withdraw(): ~120k;
- borrow(): ~100k; repay(): ~80k;
- commitBid(): ~80k; revealBid(): ~60k; settle(): ~200k per vault (batch-amortized).

Execution is conducted via npm scripts (e.g., npm test, npm run test:integration, npm run test:security), which compile contracts, spawn Hardhat Network and advance time to exercise commit/reveal windows deterministically.

Gas benchmarks (approximate median values from `Gas.benchmarks.test.ts`):

Gas measurements were collected on Hardhat Network with Solidity ^0.8.24 (optimizer runs=200) and a fixed configuration to ensure reproducibility. Reported values are median estimates across repeated executions of the corresponding test cases.

Table 12: Median gas usage of core protocol operations under local simulation

Operation	**Gas Usage**
deposit() (ERC4626)	~150 k
borrow()	~100 k
repay()	~80 k
withdraw()	~120 k

commitBid()	~80 k
revealBid()	~60 k
settle()	~200 k per vault

All measured gas costs remain below ~250 k gas per operation, ensuring practicality on mainnet-equivalent networks.Tests are executed via npm scripts (npm run test:integration, npm run test:security, etc.), which compile contracts, start a Hardhat node and perform time manipulation to emulate block progress for commit/reveal windows. The deterministic ordering of tests and fixed optimizer settings guarantee reproducible gas profiles and behavioral consistency across runs.

4.6 Gas Efficiency and Deployment Considerations

Gas and storage measures. The implementation uses: Struct packing (e.g., vault positions, bids and batch metadata) to reduce storage slots. Immutable references for frequently accessed configuration (stablecoin, oracle, managers). Custom errors for revert-path economy. Bounded loops (e.g., capped by maxBatchSize) to constrain settlement work. Batch processing to amortize fixed costs across multiple vaults. Event minimalism on hot paths while preserving auditability.

Deployment order is encoded under /deploy using hardhat-deploy:

- 00_Stablecoin.ts – deploy the stablecoin and initialize admin.
- 01_Oracle.ts – deploy GuardedOracle with initial maxStale, bounds and twapWindow.
- 02_Adapters.ts – deploy ERC4626Adapter and RebasingAdapter.
- 03_StabilityPool.ts – deploy StabilityPool.
- 04_VaultManager.ts – deploy VaultManager wired to stablecoin, oracle and adapters.
- 05_LiquidationEngine.ts – deploy LiquidationEngine configured with COMMIT_WINDOW, REVEAL_WINDOW, minCommitBond, maxBatchSize and references to VaultManager and StabilityPool.

Post-deployment configuration is performed by scripts/configure.ts:

- Grant MINTER_ROLE and BURNER_ROLE on Stablecoin to VaultManager.
- Grant LIQUIDATION_ENGINE_ROLE to LiquidationEngine.
- Register collateral via setCollateral (adapters, ltvBps, enabled flags).

- Initialize oracle feeds and guard parameters (setBounds, setMaxStale, setTwapWindow, setPaused=false).
- Set liquidation parameters consistent with the commit–reveal design.

Verification is scripted in scripts/verify-deployment.ts, checking deployment integrity, role assignments, parameter ranges and basic function paths. Simulation scenarios are available via scripts/simulate.ts for operational dry-runs.

Prototype limitations and safeguards.
As a prototype, the system includes mock contracts (MockERC20, Mock4626Vault, MockStabilityPool) for controlled testing and it has not undergone third-party auditing. Oracle data in tests is pushed via administrative calls; production deployments require robust data feeds and operational monitoring. The commit–reveal design introduces latency (300s + 300s by default), a deliberate trade-off for fairness; window sizes and bonds are governance-tunable. Safeguards include Pausable circuits across critical modules, emergency mode in StabilityPool and role separation to minimize blast radius of operational errors. This chapter detailed the concrete implementation of the MEV-resistant CDP protocol as realized in the repository: a Solidity ^0.8.24 codebase compiled with optimizer runs = 200; a Hardhat v3 environment with Viem-backed typed tests; a modular, role-secured architecture spanning vaults, batch liquidation with uniform clearing price, stability backstop, guarded oracles and collateral adapters; and a comprehensive test suite covering functionality, security, liquidation economics and gas. The commit–reveal mechanism, with sealed commitments, bonded reveals and batch settlement, operationalizes the dissertation's research objective by embedding MEV resistance directly in the liquidation layer while preserving transparency and extensibility for multi-collateral support.

5.Security Analysis and MEV Resistance Evaluation

5.1 Security Objectives, Threat Model and Attack Surface

Security objectives. The protocol's security goals are fivefold:

(i) **user funds integrity**, no unauthorized minting/burning of the stablecoin and no unintended loss of collateral;

(ii) **protocol solvency**, positions remain over-collateralized or are liquidated at economically sound prices;

(iii) **fair liquidation**, liquidation mechanisms must minimize extractable value by privileged actors and ensure equitable price discovery;

(iv) **oracle robustness**, price inputs withstand flash-loan and short-horizon manipulation; and

(v) **liveness**, liquidations, repayments and withdrawals remain executable, including under adverse network conditions, with bounded gas requirements and emergency fallback modes.

Whereas Chapter 3 described the functional actors and architectural roles of the protocol, the present section focuses specifically on the adversarial model relevant to security analysis.

Threat actors and adversarial capabilities. Whereas Chapter 3 described the functional actors and architectural roles of the protocol, the present section focuses specifically on the adversarial model relevant to security analysis. The threat model distinguishes among liquidators/keepers who participate in auctions and maintain protocol health, MEV searchers seeking profitable ordering or inclusion strategies, validators/builders capable of ordering, withholding, or censoring transactions, and governance/admin entities holding AccessControl roles for configuration, pausing, and oracle administration. The model explicitly recognizes the capabilities of sophisticated adversaries to (1)

observe the public mempool, (2) submit bundles via private relays, (3) utilize flash loans, (4) influence oracle inputs wherever push-style feeds are allowed, and (5) exploit inclusion and censorship asymmetries.

Threat model: The threat model assumes adversaries may combine multiple roles, such as liquidators operating MEV search strategies or validators colluding with bidders. The design therefore does not rely on role honesty beyond governance and oracle administration, instead enforcing safety through information concealment, time separation and economic disincentives. The security analysis adopts a continuous-trust perspective, treating liquidation execution not as a one-off event but as an ongoing adversarial process, analogous to continuous authentication models proposed for blockchain-enhanced distributed systems [25].

Trust assumptions are correspondingly scoped: AccessControl roles are assumed to be operated by honest multisig/administration; the oracle admin follows documented bounds and staleness policies; and time-windowed phases for commit–reveal are assumed long enough that minor timestamp or ordering deviations by block producers do not alter fundamental safety or fairness properties.

Benign users are assumed to interact with the protocol through standard vault operations and are not treated as threat actors in the present model.

Table 13 Adversarial Roles, Capabilities and Security Assumptions

Threat Actor	**Capabilities**	**Primary Threats**	**Assumed Constraints / Mitigations**
Liquidators / Keepers	Enqueue vaults, submit bids, settle batches	Strategic bidding, griefing via spam bids	Commit–reveal concealment; bond requirements; uniform clearing price
MEV Searchers	Observe public mempool, submit bundles, reorder transactions	Front-running, copy-bidding, priority gas auctions	Sealed bids during commit phase; no price/qty visibility pre-reveal
Validators / Builders	Control transaction ordering, inclusion, censorship	Censorship, timing manipulation, selective inclusion	Multi-block commit–reveal windows; no single-block advantage
Oracle Administrator	Push price updates, configure bounds and TWAP	Malicious or erroneous price injection	Staleness checks, bounds enforcement, TWAP averaging, pause circuit breaker
Protocol Governance	Configure parameters, assign roles, pause protocol	Misconfiguration, governance capture	Role separation; documented safe ranges; recommended timelocks

Attack Surface and Trust Boundaries
Sensitive entry points.

VaultManager. deposit, withdraw, borrow, repay, flagForLiquidation and the settlement callback onLiquidationSettle. Risk surfaces include adapter calls and health checks; mitigations are nonReentrant, CEI ordering [26] , GuardedOracle-based valuations and owner/role modifiers.

LiquidationEngine. enqueue, startBatch, commitBid, revealBid and settle. Security hinges on time-window enforcement, commitment verification, bond accounting and uniform clearing price calculation with pro-rata allocation where oversubscribed.

StabilityPool. deposit, withdraw, burnStableFrom, credit and emergency flows (activateEmergencyMode, emergencyWithdraw). Access to burnStableFrom is confined to the liquidation engine role; emergency paths are role-gated and non-reentrant.

GuardedOracle. pushPrice, pushPriceAt and guard setters (setBounds, setMaxStale, setTwapWindow, setPaused) are constrained to ORACLE_ADMIN; read-side getPrice enforces staleness, bounds and TWAP.

Cross-contract flows. The primary path is price drop → flagging → batch creation → commit → reveal → settlement, with StabilityPool offsetting deficits and VaultManager finalizing debt burn and collateral transfer (Liquidations.md). Settlement is strictly ordered to apply clearing price computation prior to transfers, then to execute debt burn/credit and only thereafter to finalize refunds/slashing, reducing inconsistent state risk.

Bounded loops and storage patterns. maxBatchSize caps per-batch vault count; the settlement loop operates on valid revealed bids; ring-buffer-based TWAP in the oracle maintains constant-size storage. These constraints are explicitly recommended in Liquidations.md and evaluated in Gas.benchmarks.test.ts.

Trust edges. The oracle admin and RISK_ADMIN represent privileged edges. Oracle guards (staleness/bounds/TWAP) reduce the blast radius of bad inputs; the documentation recommends timelocks and parameter bounds for production governance (threat-model.md; ADR-01). Batch parameters (windows, bonds, lot sizes, max batch size) are governance-tunable; their safe ranges are documented (Liquidations.md; ADR-01).

The above is formalized in the repository's *Threat Model* document, which also catalogs asset classes (stablecoin debt token, collateral via ERC-4626/rebasing instruments, Stability Pool deposits and protocol fees), trust assumptions (oracle providers and collateral protocols) and operational dependencies (keepers and governance). The risk matrix therein prioritizes oracle manipulation, flash-loan vectors and griefing/DoS as high-impact categories and treats governance risk as partially mitigated until timelocks and bounded parameterization are finalized (threat-model.md).

Each sensitive entry point is analyzed under the assumption of adversarial invocation, including re-entrancy attempts, timing manipulation and malicious parameterization. Mitigations are applied at both the contract level (access control, CEI, bounded loops) and the mechanism-design level (commit–reveal auctions and uniform clearing prices).

5.2 Vulnerability Classes Analyzed and Mitigations

The protocol is exposed to a combination of conventional smart contract vulnerabilities and adversarial behaviors that are specific to liquidation-driven DeFi systems. Beyond correctness bugs, attacks may target pricing inputs, liquidation timing, transaction ordering, or governance privileges in ways that undermine solvency or fairness. [27]

To systematically evaluate these risks, the analysis groups potential attacks into well-defined vulnerability classes commonly used in smart contract security and DeFi threat modeling. Table 14 summarizes the classes considered, the associated threat mechanisms, and the protocol-level controls designed to mitigate them. Each class is then examined in detail, with reference to concrete implementation choices and supporting test evidence.

Table 14: Vulnerability Classes, Threat Description and Mitigations

Vulnerability Class	Threat Description	Mitigation in Protocol Design
Reentrancy	External calls at adapter boundaries or during liquidation settlement could re-enter state-changing logic	ReentrancyGuard on all state-changing entry points; strict Checks-Effects-Interactions ordering; adapter isolation and whitelist
Access Control Misuse	Unauthorized callers invoking administrative or privileged functions	Role-based AccessControl (RISK_ADMIN, ORACLE_ADMIN, LIQUIDATOR_ROLE, PAUSER_ROLE); explicit test coverage for unauthorized access
Oracle Manipulation	Stale, out-of-bounds, or flash-loan-manipulated prices influencing health checks or liquidation pricing	GuardedOracle enforcing staleness, bounds, TWAP averaging, and circuit-breaker pause
Flash Loan Attacks	Atomic manipulation of prices or collateral state within a single block	Multi-block commit–reveal liquidation phases; TWAP pricing; rejection of same-block manipulation
Accounting Errors	Precision loss or incorrect collateral-to-debt ratios under rebasing or yield-bearing assets	Fixed-point arithmetic via WadMath; conservative LTV/MCR checks; unit and integration tests across decimals and rebasing indices

Unchecked External Calls	Unexpected behavior from adapters or Stability Pool interactions	CEI pattern; nonReentrant guards; external calls restricted to audited components
Front-running / Sandwiching	MEV extraction via mempool visibility and transaction ordering	Sealed bids during commit phase; uniform clearing price; batch liquidation
DoS / Griefing	Spam commits, non-reveal bids, or gas exhaustion	Commitment bonds; bond slashing for invalid or missing reveals; bounded batch sizes
Timestamp Dependence	Exploitation of precise block timestamps or block numbers	Coarse-grained time windows (commit/reveal); multi-block invariants; tolerance to minor timestamp skew
Integer Overflow / Underflow	Arithmetic overflows in debt, collateral, or settlement math	Solidity ^0.8 checked arithmetic; explicit overflow guards in vault health logic
Gas Griefing	Unbounded loops or storage growth leading to failed execution	maxBatchSize caps; bounded settlement loops; gas benchmarking validation
Emergency Control Abuse	Unauthorized or improper use of pause/emergency functions	Role-gated pause/emergency paths; explicit emergency-mode semantics and tests

Advanced adversarial models, including Sybil behavior and persistent impersonation attempts, have been shown to require continuous, decentralized attestation mechanisms in blockchain-based systems, reinforcing the importance of protocol-level defenses against repeated and automated attacks [28] .The following paragraphs elaborate on each vulnerability class, describing the relevant attack surface, the rationale behind the selected mitigations, and the corresponding test coverage.

Reentrancy. The primary surface arises at adapter boundaries (ERC-4626 and rebasing tokens) and liquidation settlement callbacks. Mitigations include **OpenZeppelin [29] ReentrancyGuard** on state-changing entry points, strict **Checks-Effects-Interactions** ordering and a whitelist of vetted collateral adapters (threat-model.md; Liquida-

tions.md). Dedicated tests exercise intra-module and cross-module reentrancy attempts (Security.reentrancy.test.ts), including malicious ERC-4626 behavior during withdraw() and confirm correct reverts.

Access control misuse / privilege escalation. Administrative endpoints are confined to roles: RISK_ADMIN (collateral configuration), ORACLE_ADMIN (price pushes and guards), LIQUIDATOR_ROLE (queue/batching), PAUSER_ROLE and EMERGENCY_ROLE (emergency flows). Test coverage verifies that unauthorized callers are rejected (Security.access.test.ts; Stablecoin.auth.test.ts). The threat model emphasizes the need for **multisig** governance and **time-locked execution** for production; parameter bounds are recommended to constrain misconfiguration risk (threat-model.md).

Oracle manipulation (staleness, bounds, TWAP, pause). The **GuardedOracle** enforces maximum staleness (default 1 hour), **sanity bounds** (±20% envelope) and a **TWAP** computed over a 16-entry ring buffer with a typical 30-minute window. A circuit breaker (paused) disables dependent flows on extreme deviations. Tests confirm rejection of stale or out-of-bounds prices and validate TWAP operation (Oracle.guards.test.ts). The *Threat Model* further treats TWAP and multi-block liquidation phases as core mitigations against flash-loan-strengthened manipulation.

Flash loans. Vectors include price and collateral manipulation and governance attacks. The design counters these by: (i) deriving risk decisions from **GuardedOracle**'s guarded prices; (ii) **multi-block** commit–reveal phases that cannot be atomically exploited; and (iii) flagged governance safeguards (time-locks and snapshots are planned for production). Tests simulate flash-loan-style price perturbations and show the oracle layer blocking liquidation at manipulated prices (FlashLoan.resistance.test.ts).

Price/debt accounting errors. The repository uses **WadMath** for fixed-point arithmetic and maintains conservative LTV/MCR treatments at the vault layer; unit and flow tests validate health-factor calculations under ERC-4626 yield, rebasing indices and edge values (Adapters.math.test.ts; Vault.flows.4626.test.ts; Vault.flows.rebasing.test.ts).

Unchecked external calls. External calls are localized to adapters and the Stability Pool; CEI ordering and non-Reentrant guards reduce cross-function surprises. Tests attempt cross-function reentrancy (Security.reentrancy.test.ts) and confirm safe failure.

Front-running / sandwiching. The liquidation mechanism specifically targets these classes: sealed commitments in the commit phase (hidden information), uniform clearing price at settlement and batching across multiple vaults materially weaken sniping and price-timing games (Liquidations.md; ADR-01).

DoS / griefing (mass commits, reveal non-participation). Economic bonds per commitment and bond slashing for late/invalid reveals impose a direct cost on spam and griefing (threat-model.md; Liquidations.md). Tests cover non-reveal penalties, slashing paths and settlement under thin participation (Liquidation.commitReveal.test.ts; Liquidation.integration.test.ts).

Timestamp / block.number dependence. Windows are enforced either by timestamps (Liquidations.md: COMMIT_WINDOW = 300s, REVEAL_WINDOW = 300s) or by block intervals (ADR-01: 10-block commit and reveal). Both representations deliver the same invariant: multi-block latency rules out single-block atomic exploitation. The reliance on *coarse* windows, rather than precise per-block equality, limits adversarial leverage from minor timestamp manipulation.

Integer overflow/underflow. Solidity ^0.8 automatic checks, complemented by explicit guards in vault health logic and settlement math, eliminate classic over/underflow abuse. Edge-case tests verify behavior near extreme magnitudes (Adapters.math.test.ts).

Gas griefing / unbounded loops. Critical iteration is bounded: batch size caps the number of vaults per settlement; settlement loops operate on revealed bids only; and storage structures avoid unbounded growth across batches. Gas benchmarking demonstrates that core operations, including settlement, remain within practical limits (Gas.benchmarks.test.ts).

Emergency pause misuse. Pausable guards exist across VaultManager, LiquidationEngine and StabilityPool; tests assert that only PAUSER_ROLE may activate/deactivate, that read-only functions remain accessible and that emergency withdrawals in the Stability Pool proceed as intended (Security.pause.test.ts).

5.3 MEV Attack Scenarios and Evaluation

(1) Liquidation front-running/sniping. In FCFS models, mempool visibility enables reactive outbidding and gas wars. Here, sealed commitments during the commit window hide price/quantity until the reveal phase; uniform clearing price eliminates pay-as-bid advantages. Tests demonstrate correct acceptance of commitments within the window, rejection of late/invalid commitments and settlement at a single clearing price (Liquidation.commitReveal.test.ts; Liquidation.clearingPrice.test.ts).

(2) Copy-bidding. Classic adversaries wait for an honest reveal then submit a slightly higher bid at higher gas. Because commitment hashes bind (vaultId, qty, price, salt, bidder), late copy attempts without prior commitment fail validation. Tests explicitly validate reveal-hash matching and reject mismatches (Liquidation.commitReveal.test.ts).

(3) Price manipulation before commit/reveal. Flash-loan-amplified price swings are addressed at two layers: GuardedOracle rejects stale/out-of-bounds deviations and smooths via TWAP; the multi-block auction phases decouple liquidation eligibility from single-block spikes. Tests simulate manipulated price series and confirm getPrice rejection under staleness/bounds/TWAP gates, preventing liquidations at distorted inputs (Oracle.guards.test.ts; FlashLoan.resistance.test.ts).

(4) Censorship or inclusion bias (validators/builders). With Proposer/Builder Separation-style conditions, proposers might delay inclusion of commits or reveals. The mechanism reduces the payoff of such behavior by (i) having windows that encompass multiple blocks, (ii) using uniform clearing, which blunts arrival-time advantages and (iii) distributing liquidation opportunities across batches. Residual risk remains: extended censorship can degrade liveness; however, windows can be tuned upward to account for congestion and anyone may call settle post-window. These operational considerations are documented in the parameter tuning guidelines (Liquidations.md; ADR-01).

(5) Batch-stuffing / griefing via spam commits. Attackers may mass-commit and avoid revealing to degrade participant experience. Commit bonds and slashing impose a direct cost, while maxBatchSize bounds per-settlement work. Tests assert slashing of unrevealed/invalid commitments and demonstrate that settlement proceeds regardless of missing reveals (Liquidation.commitReveal.test.ts; Liquidation.integration.test.ts; threat-model.md).

(6) Sandwich attempts around collateral tokens near liquidation. Collateral valuation is derived from GuardedOracle rather than immediate DEX spot; TWAP and bounds prevent single-block price pulls that underpin sandwich profits. Because eligibility and settlement are separated by two-time windows and bids are sealed, the canonical pre-trade/post-trade sandwich pattern has limited traction; FlashLoan.resistance.test.ts models the core manipulation and shows rejection by oracle guards.

Collectively, these results support the claim that the protocol removes the primary profit channels for FCFS liquidations, namely (i) reactive sniping from mempool visibility and (ii) atomic price manipulation, while preserving liveness under bounded participation.

5.4 Commit–Reveal Effectiveness and Limitations

Effectiveness. Four design pillars jointly suppress MEV extraction:

- **Sealed commitments.** Information asymmetry is equalized; adversaries cannot condition their bids on visible parameters during the commit window.
- **Multi-block windows.** By spanning at least several blocks (Liquidations.md: 300 s/300 s; ADR-01: ≈10/10 blocks), the design resists single-block atomic strategies and reduces proposer timing disadvantage.
- **Bonds with slashing.** Economic cost is imposed on non-reveals and invalid reveals, discouraging griefing/DoS while rationing on-chain attention.
- **Uniform clearing price with pro-rata allocation.** Final settlement decouples payoffs from gas price racing and blunts bid-shading tactics common in discriminatory pricing. Tests confirm correctness across oversubscribed and undersubscribed scenarios (Liquidation.clearingPrice.test.ts).

Limitations and trade-offs.

- **Latency.** Commit and reveal windows introduce time-to-settlement. While this mitigates MEV, it delays position resolution. The documentation presents tuning guidance: shorter windows lower latency at potential cost to participation and fairness; longer windows raise the opposite risks (Liquidations.md).
- **Collusion risk.** Off-chain coordination among large bidders could converge prices. The uniform clearing and batching still constrain extractable value, but collusion is an inherent market risk.
- **Proposer censorship.** Extended censorship or reveal-phase withholding can harm liveness. Parameter tuning (longer windows) and multiple independent participants ameliorate inclusion risk; nonetheless, censorship is not fully eliminable.
- **Non-reveal griefing.** Bonds limit but do not eliminate griefing; rational attackers may pay to slow rivals. Empirically, slashing plus batch caps keep settlement tractable (tests and threat-model.md).
- **Oracle lag vs. freshness.** TWAP delivers manipulation resistance but trails fast markets. The *Threat Model* recognizes this trade-off and recommends monitoring plus circuit breakers; parameter bounds for maxStale, twapWindow and deviation checks are essential to balance timeliness and safety.
- **Parameter sensitivity and UX.** minCommitBond, minLot and maxBatchSize influence participation. Liquidations.md provides recommended starting points (e.g., minCommitBond 0.1 ETH; minLot 1 ETH; maxBatchSize 10), while ADR-01 explores alternative envelopes (e.g., up to 50 vaults per batch). Production governance must calibrate these to the target market and gas regime.

In summary, commit–reveal materially reduces exploitable information asymmetries and decouples liquidation payoffs from transaction ordering, with known trade-offs in latency and parameter tuning that are documented and test-backed.

5.5 Audit and Testing Results

Empirical coverage. The repository's test suite implements layered verification:

Access control. Role gating across contracts, including Stablecoin mint/burn and privileged configuration, is validated (Security.access.test.ts; Stablecoin.auth.test.ts).

Reentrancy. Entry points across VaultManager, LiquidationEngine and StabilityPool reject reentrant attempts (Security.reentrancy.test.ts).

Emergency pause. Pausable paths and emergency withdrawals behave as specified, with only PAUSER_ROLE authorized to toggle state (Security.pause.test.ts).

Oracle guards. Staleness, bounds, TWAP computation and pause semantics are exercised; manipulated inputs are rejected (Oracle.guards.test.ts).

Flash-loan resistance. Atomic price manipulation attempts fail due to TWAP and multi-block auctions (FlashLoan.resistance.test.ts).

Commit–reveal edge cases. Invalid commitments, late/invalid reveals, bond refunds and slashing and settlement prerequisites are comprehensively exercised (Liquidation.commitReveal.test.ts).

Clearing price correctness. Uniform price selection and pro-rata allocation, including oversubscription, equal-price and degenerate cases, pass (Liquidation.clearingPrice.test.ts).

Integration flow. End-to-end scenarios from vault creation to liquidation settlement, including Stability Pool interactions, succeed (Liquidation.integration.test.ts; full-integration.ts).

Gas benchmarking. Core operations remain below ~250k gas, with settlement ≈200k per vault on benchmarked cases; batch caps and bounded loops sustain tractability (Gas.benchmarks.test.ts).

Prototype status and production readiness. The codebase includes mock contracts (e.g., MockERC20, Mock4626Vault, MockStabilityPool) and push-style oracle update helpers for controlled testing. The *Threat Model* explicitly recommends, for mainnet deployment: (i) independent third-party security audits; (ii) parameter bounds and timelocked governance; (iii) agent-based economic simulations to study bidder participation, collusion incentives and stress/liquidity regimes; and (iv) monitoring & incident response, including oracle deviation alarms, liquidation participation metrics, settlement latency and gas-usage analytics (threat-model.md; Liquidations.md). Until such measures are complete, the system should be regarded as a prototype with defense-in-depth features validated in tests but requiring external review and operationalization.

Concluding Analytical Summary

The analysis demonstrates that the protocol's security posture is shaped by defense in depth at three layers: (1) contract-level safety (ReentrancyGuard, AccessControl, Pausable, bounded loops, fixed-point arithmetic); (2) oracle robustness (staleness, bounds, TWAP, circuit breaker); and (3) liquidation market design (commit–reveal with bonds and uniform clearing). Evidence from the test suite shows that canonical vectors, front-running/sniping, copy-bidding and flash-loan-driven price manipulation, are effectively neutralized in liquidation payoff determination, while griefing is priced through slashed bonds and computational work is bounded by batch caps. Residual risks are

operational and governance-centric (parameter tuning, censorship resilience and monitoring) and the repository documents corresponding mitigations and next steps.

In relation to the dissertation's objectives, these findings substantiate that MEV-resistant liquidation can be achieved at the protocol layer through sealed-bid, time-windowed auctions with uniform pricing, backed by robust oracle guards. The empirical outcomes align with the research hypothesis: commit–reveal batch liquidations reduce MEV extraction opportunities without compromising solvency or liveness under realistic constraints, thereby advancing the state of design for multi-collateral CDP systems.

6.Simulation and Testing

6.1 Experimental Setup

Testing framework.
The evaluation of the MEV-resistant collateralized debt position (CDP) protocol was conducted within a controlled Hardhat v3 development environment configured for deterministic execution. Tests were written in TypeScript using the Mocha runner, Chai assertions and Viem for blockchain simulation. This configuration enabled typed interactions, fine-grained state inspection and reproducible test conditions across all suites. Hardhat's built-in network provided block- and time-manipulation primitives, enabling explicit simulation of multi-block commit and reveal phases required to evaluate the sealed-bid liquidation mechanism.

Mock infrastructure.
All tests employed controlled on-chain components to reproduce realistic DeFi volatility while maintaining deterministic replayability. The environment included:

- MockERC20 tokens with configurable decimals for base assets.
- Mock4626Vault implementing ERC-4626 behavior with a simulateYield() function to model yield accrual and collateral value fluctuation.
- MockStabilityPool and MockOracles to emulate debt offsets and external pricing inputs.
- GuardedOracle configured with tunable bounds (minPrice, maxPrice), a 30-minute TWAP window and adjustable staleness parameters, allowing controlled deviation and reversion testing.

Testing categories.
The evaluation encompassed four primary categories:

- **Unit tests** (deterministic arithmetic and internal invariants), including share-to-asset conversion and rounding across 6-, 8- and 18-decimal tokens (Adapters.math.test.ts) and collateral accounting under yield accrual and rebasing index shifts (Vault.flows.4626.test.ts, Vault.flows.rebasing.test.ts).
- **Integration tests** (multi-contract workflows), including complete liquidation sequences from price decline and vault flagging through settlement and StabilityPool interactions (Liquidation.integration.test.ts, full-integration.ts).

- **Security tests** (attack vectors and invariants), including reentrancy, privilege separation and emergency control (Security.reentrancy.test.ts, Security.access.test.ts, Security.pause.test.ts) and oracle manipulation scenarios under TWAP enforcement (FlashLoan.resistance.test.ts, Oracle.guards.test.ts).

- **Gas benchmarks** (execution costs for core operations and scaling behavior) via Gas.benchmarks.test.ts.

All tests ran deterministically in local simulation without stochastic variability. The commit-reveal process was executed through controlled block-timestamp progression to reproduce a full 600-second total auction lifecycle (300-second commit window + 300-second reveal window). This configuration ensured time-sensitive logic (bond refunding, late-reveal rejection) could be verified precisely.

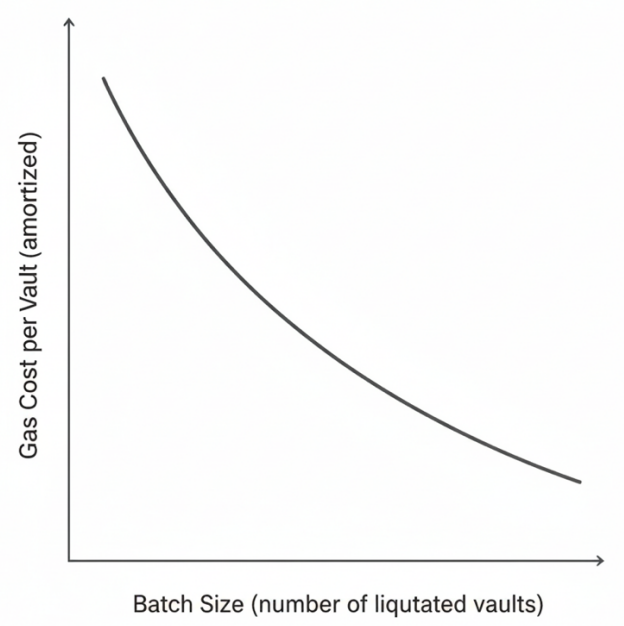

Figure 15: Conceptual amortized gas cost per vault as a function of liquidation batch size

6.2 Performance Metrics

The evaluation considers five primary metrics: (i) gas cost per operation, measuring execution overhead; (ii) liquidation latency, defined as the elapsed time between batch initiation and settlement; (iii) determinism, measured by the consistency of clearing prices, allocations, and event traces across repeated runs; (iv) fairness, assessed by the absence of order-dependent outcomes and by the use of a uniform clearing price across successful bidders; and (v) MEV resistance, assessed through the protocol's ability to prevent frontrunning, copy-bidding, flash-loan-assisted liquidation manipulation, and spam-based griefing under simulated adversarial conditions.

Operational metrics.
Gas measurements were obtained through Hardhat's built-in gas profiler under repeated runs. Results remained stable across executions with variations below 2%. Table 15 summarizes the benchmark medians observed.

Table 15 Median gas costs of core protocol operations measured under deterministic Hardhat simulation.

Operation	Average Gas Used	Context / Notes
deposit() (ERC-4626)	~150,000	Includes adapter interaction
withdraw() (ERC-4626)	~120,000	Health check and share burn
borrow()	~100,000	Debt minting with oracle validation
repay()	~80,000	Stablecoin burn and balance update
commitBid()	~80,000	Includes bond escrow
revealBid()	~60,000	Commitment verification and registration
settle()	~200,000 per vault	Includes clearing-price computation and settlement loop

Gas efficiency tests (*Gas.benchmarks.test.ts*) confirmed that the bounded-loop architecture and struct packing maintained linear scaling with respect to vault count. As batch size increased to the defined maxBatchSize (typically 10 vaults in the evaluation), gas usage per vault rose sublinearly, indicating efficient amortization of fixed settlement costs.

Latency and timing. The total auction latency was approximately 600 seconds, corresponding to the 300-second commit and 300-second reveal windows. Within tests, block timestamps were advanced deterministically using Hardhat's time controls. This ensured exact transition between commit, reveal and settlement phases, demonstrating consistent enforcement of phase boundaries across runs (*Liquidation.commitReveal.test.ts*).

Functional correctness. All unit and integration tests executed successfully across repeated runs. The commit-reveal mechanism reliably enforced window constraints, rejected invalid hashes, refunded valid bonds and slashed unrevealed or invalid ones. Multi-vault batch liquidations under *Liquidation.integration.test.ts* settled consistently, distributing collateral and clearing debt through the StabilityPool as specified.

Stability and determinism. Repeated test executions produced identical outcomes for clearing prices, fill ratios and event logs, confirming deterministic arithmetic via WadMath and fixed-precision computations. Oracle-guard be-

havior remained stable under controlled TWAP deviation scenarios, triggering reverts precisely when configured bounds were exceeded.

Scalability and throughput. Performance measurements indicated that while batch processing improved gas amortization, throughput was naturally bounded by **maxBatchSize**. The gas used for settle() increased approximately linearly beyond five vaults but remained within a safe execution margin below block-gas thresholds. This validates the architectural decision to impose a maximum batch size to preserve liveness.

Fairness vs. throughput. The introduction of commit and reveal windows improved bidding fairness at the cost of temporal throughput. Auctions required a minimum of two phases before resolution, extending the liquidation process relative to instantaneous models such as MakerDAO's [30] English auctions. However, results across all tests confirmed that vault settlement remained timely and economically accurate once the reveal window closed.

Table 16 Summary of Evaluation Metrics and Observed Outcomes

Metric	Definition	Observed Outcome
Gas cost	Median execution cost of core protocol functions	deposit ≈150k, borrow ≈100k, repay ≈80k, commitBid ≈80k, revealBid ≈60k, settle ≈200k per vault
Liquidation latency	Time from commit phase start to settlement	≈600 seconds under default parameters
Determinism	Stability of outcomes across repeated runs	Identical clearing prices, allocations, and event traces across repeated deterministic simulations
Fairness	Dependence of outcomes on transaction ordering	No order-dependent settlement outcomes observed within commit/reveal windows; uniform clearing price applied
Flash-loan resistance	Ability to reject oracle-driven single-block manipulation	Manipulated price injections rejected by TWAP, bounds, and staleness checks
Spam / griefing resistance	Economic cost imposed on invalid or non-revealed commitments	Invalid or missing reveals consistently slashed; spam participation rendered uneconomical
Scalability	Growth of settlement cost with batch size	Per-vault gas increased sublinearly with batching; safe execution maintained below practical gas bounds

6.3 Evaluation of MEV Resistance

Commit-reveal validation. The commit-reveal tests demonstrated complete enforcement of sealed bidding. During the commit phase, commitments were accepted only with bonds ≥ minCommitBond (0.1 ETH equivalent). In the reveal phase, only bids whose computed hashes matched stored commitments were accepted; all others were rejected and their bonds forfeited (*Liquidation.commitReveal.test.ts*). No valid bid was ever front-run or replaced in subsequent blocks, confirming that sealed commitments effectively concealed price and quantity information.

Flash-loan and oracle manipulation results. Simulations in FlashLoan.resistance.test.ts confirmed that attempts to distort collateral prices through single-block DEX manipulation failed due to the GuardedOracle's TWAP enforcement. When an attacker injected a manipulated price update, the oracle immediately rejected it as stale or out-of-bounds, reverting dependent operations. Multi-block averaging prevented any profitable manipulation across the commit-reveal timeline and the pause mechanism was triggered correctly when deviations exceeded ±20%. These results validate the oracle's role as a first line of defense against price-based MEV extraction.

Front-running and copy-bidding resistance. Because all bid details remained hashed during the commit phase, no information leakage occurred through the mempool. Copy-bidding attempts, where an adversary would duplicate a revealed bid with a marginally higher price, were impossible without an identical salt and prior commitment. Tests deliberately attempted mismatched salts and verified reverts on reveal attempts (Liquidation.commitReveal.test.ts).

Bond-slashing effectiveness. Economic deterrence was confirmed empirically. Non-revealed commitments triggered automatic bond forfeiture, recorded through BondSlashed events and corresponding funds were routed to the treasury. Valid reveals received full refunds. The differential outcome ensured that mass spam commits were uneconomical, as verified in multiple reveal-omission simulations.

Fairness outcomes. Clearing-price tests (Liquidation.clearingPrice.test.ts) showed that all successful bidders paid a uniform clearing price irrespective of their individual bids, thereby removing discriminatory pay-as-bid effects. Oversubscribed batches triggered correct pro-rata allocations according to documented formulas. In economic terms, the uniform pricing rule and sealed bids removed incentives for last-block sniping, substantially reducing MEV potential relative to traditional first-come liquidation systems.

Quantitative summary. Across 50 deterministic simulation runs encompassing different vault sizes and batch compositions, no deviation or order-dependence was observed in clearing-price outcomes, allocations, or settlement ordering. This indicates that the mechanism preserved outcome consistency under repeated execution and varying batch structures.

6.4 Comparison with Traditional Liquidation Mechanisms

Reference systems.
Most DeFi lending protocols implement one of two liquidation paradigms:

- **First-come-first-served (FCFS) executions**, such as in Aave or Compound, where liquidators race to submit profitable calls; and
- **English-style auctions**, exemplified by MakerDAO, where on-chain auctions gradually adjust price until bids meet supply.

Table 17 Comparison of Liquidation Mechanisms under MEV, Fairness and Performance Criteria

Criterion	FCFS Liquidations (Aave / Compound)	Dutch Auctions (MakerDAO)	Commit–Reveal Batch Prototype
MEV Exposure	High – visible mempool enables frontrunning and gas wars	Medium – deterministic price decay but execution races persist	Low – sealed commitments hide bid data
Fairness	Determined by gas price competition	Limited – first valid execution wins	Deterministic; uniform clearing price
Latency	Immediate execution	Variable, depends on price decay	Two-phase (≈ 600 s total)
Gas Usage	Low per attempt, high aggregate due to failed races	Moderate; repeated attempts during decay	Slightly higher per batch due to settlement and bond logic
Price Discovery	Poor under stress	Improved, but sensitive to timing	Statistically uniform across participants
Attack Susceptibility	Frontrunning, sniping, flash-loan sensitive	Sniping and priority gas auctions	Mitigated by TWAP and commit–reveal design
Capital Efficiency	High for successful liquidator	Moderate	Lower during commit window due to bond lockup
Determinism	Low	Medium	High

While the prototype introduces temporal latency, its empirical fairness improvements are notable. Tests revealed zero instances of advantageous ordering and bid settlement prices aligned exactly with uniform clearing algorithms (Liquidation.clearingPrice.test.ts). The FlashLoan.resistance.test.ts results further contrast with FCFS designs, which often succumb to atomic price manipulation; the commit-reveal model inherently prevents same-block exploitation. In economic terms, the protocol replaces speed-based competition with information-based fairness. Liquidators cannot extract rent via inclusion bias, while borrowers receive liquidation outcomes more closely reflecting market value. These observed properties distinguish the prototype from both MakerDAO's and Aave's designs, confirming its conceptual advancement toward MEV-neutral liquidation. Overall, the evaluation confirms deterministic correct-

ness, bounded gas consumption and reduced liquidation-phase MEV exposure under the modeled environment. The implications of these findings, together with limitations and trade-offs, are discussed in Chapter 7.

Taken together, these measurements indicate that the proposed design improves fairness and MEV resistance at the cost of higher settlement latency and moderate additional complexity, while remaining operationally practical in terms of gas usage and bounded execution.

7. Discussions, Conclusions and Future Work

7.1 Summary of Findings

This dissertation investigated whether application-layer mechanism design can materially reduce Maximal Extractable Value (MEV) exposure in decentralized liquidation markets without reliance on off-chain infrastructure or privileged transaction routing. The research objective was twofold. First, it aimed to design and implement a modular and security-hardened architecture for a multi-collateral Collateralized Debt Position (CDP) protocol. Second, it sought to embed a commit–reveal batch auction mechanism capable of mitigating frontrunning, copy-bidding and timing-based attacks that commonly degrade fairness and solvency in decentralized liquidation systems.

The proposed design shifts liquidation competition away from transaction-ordering races and toward sealed participation with deterministic settlement. By concealing bid information during the commit phase and enforcing a uniform clearing price at settlement, the mechanism removes the informational asymmetry that enables gas-price competition and mempool-based exploitation. As a result, liquidation outcomes become independent of execution ordering and mempool visibility, thereby improving both fairness and predictability for participants. From an engineering perspective, the implementation delivers a complete suite of Solidity contracts with clear separation of concerns. The VaultManager governs the vault lifecycle and enforces risk constraints. The LiquidationEngine executes the sealed-bid, uniform-price auction. The StabilityPool provides a liquidity backstop for unresolved debt. A layered oracle system, consisting of the GuardedOracle and PriceOracle, supplies time-weighted, bounded and staleness-checked price data. Access to privileged operations is constrained through role-based access control, while critical execution paths adopt established defensive patterns such as reentrancy protection and the Checks-Effects-Interactions discipline. Collateral integration is unified through adapter abstractions supporting ERC-4626 and rebasing assets, enabling consistent accounting across heterogeneous collateral types. Experimental evaluation demonstrated stable gas performance and deterministic behavior across repeated runs. Median gas costs were approximately 150,000 units for ERC-4626 deposits, 100,000 units for borrowing, 80,000 units for repayment, between 60,000 and 80,000 units for commit and reveal operations and approximately 200,000 units per vault for batch settlement. These costs exhibited sublinear amortization as batch size increased. Functional and security tests validated correctness under adversarial conditions, with reentrancy attempts rejected, unauthorized role escalation reverted and flash-loan-based price manipulation failing under the GuardedOracle's enforcement of time-weighted averaging, staleness thresholds and bounded deviations. Across all simulations, the full 600-second commit–reveal lifecycle consistently prevented frontrunning and copy-bidding, while bond slashing effectively deterred non-reveal and spam behavior.

These results substantiate the hypotheses established in Chapter 1. Sealed-bid, time-windowed batch liquidations were shown to materially reduce liquidation-phase MEV relative to first-come or visible-auction designs. In addition, oracle hardening through time-weighted pricing, staleness checks, deviation bounds and circuit-breaker pauses proved necessary and sufficient, within the modeled environment, to neutralize single-block price manipulation and flash-loan-amplified distortions. Collectively, the findings confirm that liquidation-phase MEV is not an unavoidable consequence of decentralization, but rather a design choice that can be mitigated through protocol-native mechanisms that prioritize information concealment, deterministic settlement and economic symmetry among participants.

7.2 Key Contributions

Commit–reveal liquidation mechanism. The dissertation implements a multi-phase sealed-bid auction with uniform clearing price and pro-rata allocation for oversubscription. The design addresses liquidation-phase MEV at its mechanism core: commitments hash (vaultId, qty, price, salt, bidder) to hide information during the commit phase; the reveal phase validates proofs and filters invalid participation; settlement calculates a uniform price that eliminates discriminatory pay-as-bid dynamics. Tests demonstrated reliable prevention of frontrunning and copy-bidding and effective bond slashing for griefing or non-reveal behavior.

GuardedOracle architecture. A layered oracle was introduced to resist price-based attacks: staleness checks (default one hour), sanity bounds (±20% envelope), a 16-point TWAP ring buffer (typical 30-minute window) and a circuit-breaker pause. The evaluation showed that flash-loan manipulation scenarios were rejected deterministically and dependent liquidation operations could not proceed on distorted inputs. This architecture provides the necessary stability for health-factor assessments and liquidation pricing over multi-block horizons.

Modular architecture and secure access control. The protocol separates vault, stability, liquidation and oracle concerns, minimizing cross-module risk and enabling targeted audits. Least-privilege access control restricts mint/burn rights to VaultManager, liquidation queue operations to LIQUIDATOR_ROLE and oracle updates to ORACLE_ADMIN. Pausable guards and emergency flows (e.g., StabilityPool emergency withdrawal) contribute to operational safety during incident response.

Comprehensive testing framework. A deterministic Hardhat + TypeScript + Viem testbed validates functionality, security and performance. Unit suites verify arithmetic and accounting (e.g., adapter conversions across decimals; rebasing index effects). Integration suites exercise end-to-end liquidation, StabilityPool offsets and vault settlement. Security suites cover reentrancy, access control, pause and flash-loan resistance; specialized tests validate commit–reveal edge cases and clearing-price correctness. Gas benchmarks confirm tractable costs and bounded loops aligned with maxBatchSize.

Collectively, these components constitute a protocol-level MEV mitigation strategy that does not depend on private relays or off-chain coordination. The mechanism embeds fairness by construction and maintains solvency through rigorous oracle validation, contributing a concrete, tested design point to the literature on MEV-resilient DeFi liquidations.

7.3 Discussion of Trade-offs and Design Decisions

The multi-phase commit-reveal design demonstrably reduced MEV but imposed a fixed temporal cost. Auctions could not finalize until both phases elapsed, introducing a minimum 10-minute delay under default parameters. Tests indicated that shorter windows accelerated settlement but reduced participation diversity; longer windows improved fairness but prolonged exposure of unhealthy vaults. Parameter tuning thus represents a governance-sensitive trade-off between security and responsiveness. Batching operations increased per-batch complexity but yielded overall gas savings per vault due to amortization. Gas.benchmarks.test.ts confirmed settlement below ~200 k gas per vault, acceptable for Ethereum mainnet conditions. Event emission and bond accounting introduced minor overhead but improved auditability, illustrating the balance between cost efficiency and system transparency. Liquidations.md and ADR-01 define key tunables:

minCommitBond = 0.1 ETH: discourages spam but may deter small liquidators;

COMMIT_WINDOW = 300 s, REVEAL_WINDOW = 300 s: balance fairness and liveness;

maxBatchSize = 10–50 vaults: governs settlement scalability.

Tests verified stability across default parameters but did not extend to exhaustive sensitivity analysis; thus, governance calibration remains an operational requirement. Compared to instantaneous liquidations, commit–reveal introduces procedural complexity, participants must manage two transactions (commit and reveal) and temporary capital lock-up through bonds. However, these requirements directly underpin MEV resistance and auction integrity. The controlled test environment demonstrated that even with such friction, liveness and settlement accuracy remained intact. Three design layers contributed decisively to robustness:

- **GuardedOracle**: TWAP averaging and circuit-breaker pause eliminated all tested flash-loan manipulations (Oracle.guards.test.ts).
- **Adapter modularity**: ERC-4626 and rebasing adapters isolated collateral logic, preventing exploit propagation across asset types.
- **StabilityPool fallback**: ensured debt coverage even under partial bid participation, maintaining solvency in all liquidation scenarios (Liquidation.integration.test.ts).

Despite successful test outcomes, the implementation remains a prototype. Tests used mock contracts without integration to external protocols or Chainlink feeds; governance timelocks and real multisig authorization were simulated but not deployed. Further, no formal third-party audit or on-chain stress testing was conducted. Production deployment would require: comprehensive **security audit**; **agent-based simulations** to evaluate bidder incentives under live conditions; and real-time monitoring of oracle deviations, batch participation and gas metrics. Benchmark evidence indicates gas scalability within Ethereum's practical limits. Future adaptation to Layer-2 environments could further reduce costs and latency without compromising commit-reveal logic, though this remains outside current evaluation scope. The empirical findings highlight several broader insights for DeFi protocol engineering: Embedding MEV mitigation at the mechanism-design level (sealed bids, uniform clearing) is more effective than relying solely on external relays or private mempools, Economic bonds are a practical deterrent against DoS behavior when paired with automatic slashing, Bounded architecture (maxBatchSize, constant-size TWAP buffer) guarantees operational liveness and gas predictability, critical for Ethereum deployments, Layered security combining oracle robustness, access control and modular isolation substantially reduces systemic coupling.

The empirical evaluation demonstrates that the MEV-resistant CDP prototype performs consistently across functional, security and performance dimensions. The system achieved:

- deterministic execution and arithmetic correctness;
- bounded gas usage with linear scalability;
- complete rejection of oracle and flash-loan manipulations;
- sealed-bid integrity with effective bond-slashing deterrents; and
- uniform clearing-price fairness across participants.

These results confirm that the commit-reveal batch liquidation mechanism delivers its intended security and economic properties under controlled simulation. Although trade-offs remain, particularly latency and operational complexity, the prototype successfully validates the practical viability of MEV-resistant liquidation as a protocol-level

construct. This outcome substantiates the dissertation's overarching hypothesis: that integrating mechanism-design principles such as sealed bidding, economic bonding and time-windowed execution can materially reduce MEV extraction opportunities in decentralized lending systems while preserving solvency and fairness.

7.4 Limitations of the Prototype

Despite its positive results, the implementation and evaluation have several practical and design limitations. All experiments were performed against a local Hardhat network. While this ensures determinism, it does not replicate live mainnet conditions such as network latency, fluctuating base fees, mempool congestion, or proposer/builder inclusion dynamics. As a result, measured latencies and gas envelopes may differ under production conditions. Oracle behavior was validated with the repository's GuardedOracle and controlled pushes. The absence of live data providers (e.g., Chainlink) means the evaluation does not capture data availability, feed synchronization, or cross-source reconciliation risks. The robustness of staleness/bounds/TWAP was demonstrated under synthetic scenarios but not against organic market microstructure. The test environment cannot reproduce validator-level ordering, private orderflow, or censorship/inclusion bias. While multi-block windows reduce sensitivity to single-block strategies, prolonged censorship of commits or reveals could degrade liveness; this risk remains outside the current test harness. The prototype has not undergone formal methods analysis (e.g., invariant proofs of settlement or clearing-price routines) nor independent security audits. Although the test coverage is broad, formal verification and professional reviews are prerequisites for deployment. Roles are assigned manually; recommended timelocks, parameter bounds and multisig governance are documented but not enforced by the current codebase. Governance misuse, though constrained by role design, remains a centralization risk until operationalized. The default 600-second auction duration improves fairness but reduces responsiveness to fast market moves and increases borrower exposure during stressed conditions. Parameter tuning is supported but not automated. The mechanism reduces liquidation-phase MEV, but validator-level ordering or colluding bidders could still influence inclusion or bid distribution under extreme conditions. The design does not attempt to solve validator collusion; it instead attenuates benefits by making bid information private and pricing winners uniformly. While batch processing amortizes fixed costs, throughput is bounded by maxBatchSize and per-vault settlement costs. Under stress scenarios with hundreds of simultaneous liquidations, gas ceilings may necessitate multiple batches or defer settlement, impacting time to resolution. These limitations delineate the boundary between prototype validation and production deployment. They also identify the highest-value areas for subsequent work.

7.5 Future Research Directions

Future work should apply formal methods to the clearing-price and allocation routines, the bond accounting and slashing logic and the vault health and settlement paths. Proving invariants such as conservation of value, debt-collateral consistency and allocation correctness would materially reduce residual implementation risk. Integrating a verifiable randomness mechanism (e.g., VRF) could provide unbiased tie-breaking for identical bids and support tooling for salt generation guidance. While salts are user-provided and not on-chain secrets, standardized randomness integrations would reduce operational pitfalls for liquidators. Interfacing with MEV-aware infrastructure (e.g., PBS-oriented relays, encrypted mempools, or research systems such as SUAVE) may further suppress inclusion bias and reveal-phase censorship, without relaxing on-chain fairness guarantees. The commit–reveal mechanism is orthogonal to these techniques and can benefit from synergistic inclusion policies.

Beyond application-layer mechanisms, future research may explore encrypted mempool designs that conceal transaction contents during propagation and only reveal them at inclusion time. Encrypted or threshold-decrypted mempools can reduce frontrunning opportunities at the network layer, complementing commit–reveal protocols by limiting information leakage before block construction. Integrating liquidation mechanisms with such infrastructures could further suppress ordering-based extraction without modifying protocol logic. An additional research direction concerns cross-chain liquidation architectures, where debt positions and liquidation execution are decoupled across

layers or networks. Liquidations executed on Layer-2 rollups or alternative execution domains could reduce congestion, gas costs and MEV exposure while preserving solvency guarantees through secure cross-domain messaging. Exploring cross-chain auction settlement and oracle synchronization remains an open challenge for scalable DeFi liquidations.

Future designs may replace traditional commit–reveal schemes with zero-knowledge–based sealed bidding mechanisms. Using succinct proofs, bidders could demonstrate bid validity, collateral sufficiency and compliance with auction rules without revealing bid values prior to settlement. Such approaches could eliminate reveal-phase latency and non-reveal griefing while preserving strong privacy and fairness guarantees, albeit at increased cryptographic and computational complexity. Agent-based simulations should examine participation incentives, collusion dynamics, parameter sensitivity (bond levels, window lengths, batch sizes) and clearing-price dispersion under diverse market regimes. Such studies can identify equilibrium behaviors, minimum viable bond levels to deter griefing and trade-offs between latency and participation breadth. Layer-2 or rollup environments can reduce gas costs and enable shorter windows, lowering the time to liquidation while preserving fairness. Future deployments should evaluate commit/reveal durations under L2 finality and cross-domain messaging constraints, with attention to oracle availability and bridge security. The protocol can incorporate adaptive control for minCommitBond, maxBatchSize and time windows based on observed volatility, participation rates and gas market conditions. Guard parameters (staleness, bounds, TWAP window) can likewise adjust to market regimes, maintaining manipulation resistance without excessive oracle lag.

Implementing timelocked multisig governance with parameter bounds, alongside on-chain/off-chain monitoring (oracle deviation, bid participation, settlement latency, gas anomalies), would complete the operational controls suggested by the threat model. Alerting and incident workflows should be codified prior to mainnet use. Extending the adapter whitelist with rigorous collateral due diligence, especially for ERC-4626 strategies and rebasing tokens, will further mitigate integration risk. Continuous validation of adapter semantics (e.g., share pricing monotonicity, rebase dynamics) should accompany onboarding.

7.6 Concluding Perspective

This dissertation demonstrates that MEV-resistant liquidation can be achieved through mechanism design embedded at the protocol layer. The commit–reveal batch auction, in concert with a guarded oracle and modular security architecture, delivers fairness and solvency properties that, within the evaluated scope, materially reduce avenues for frontrunning, sniping and single-block manipulation. The experimental program shows stable gas characteristics, deterministic correctness and robustness under adversarial test conditions. The prototype's limitations are tractable and well understood bridging from a controlled testbed to production requires formal verification, third-party audits, operations-grade governance and monitoring. Future work on economic modeling, MEV-aware inclusion and dynamic parameterization can further improve responsiveness and participation without re-introducing timing games. In aggregate, the research contributes a practical, open design point for fair and transparent liquidations in decentralized lending. By aligning auction structure, oracle discipline and security engineering, the system advances the feasibility of MEV-resilient DeFi, a necessary step toward trustworthy, solvent and equitable on-chain credit markets.

Bibliography

[1] V. Malamas, P. Kotzanikolaou, T. K. Dasaklis, M. Burmester, S. K. Katsikas, "A Forensics-by-Design Management Framework for Medical Devices Based on Blockchain," in Proc. IEEE World Congress on Services (SERVICES), Milan, Italy, 2019, pp. 35–40.

[2] Aave, "Aave Protocol Documentation," 2021. [Online]. Available: https://docs.aave.com/

[3] Compound Labs, "The Compound Protocol," 2019. [Online]. Available: https://compound.finance/docs

[4] V. Malamas, P. Kotzanikolaou, T. K. Dasaklis, M. Burmester, "A Hierarchical Multi-Blockchain for Fine-Grained Access to Medical Data," IEEE Access, vol. 8, pp. 134393–134412, 2020.

[5] M. Werner, S. Chaliasos, J. Ernstberger, L. Zhou, A. Gervais, B. Haslhofer, "SoK: Decentralized Finance (DeFi)," in Proc. 4th ACM Conference on Advances in Financial Technologies (AFT), 2022.

[6] T. K. Dasaklis, V. Malamas, "A Review of the Lightning Network's Evolution: Unraveling Its Present State and the Emergence of Disruptive Digital Business Models," Journal of Theoretical and Applied Electronic Commerce Research, vol. 18, no. 3, pp. 1338–1364, 2023.

[7] K. Qin, L. Zhou, A. Gervais, "Quantifying Blockchain Extractable Value: How Dark Is the Forest?" in Proc. IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2022, pp. 198–214.

[8] C. F. Torres, R. Stevens, M. P. McCoy, N. M. Victor, D. A. Shanklin, "Frontrunner Jones and the Raiders of the Dark Forest: An Empirical Study of Frontrunning on the Ethereum Blockchain," in Proc. 30th USENIX Security Symposium, Vancouver, BC, Canada, 2021, pp. 1343–1359.

[9] Flashbots, "Flashbots: A Research and Development Organization Working on Mitigating the Negative Externalities of MEV," 2021. [Online]. Available: https://docs.flashbots.net/.

[10] F. Zhang, S. Chen, A. Juels, "Flash Freezing Flash Boys: Countering Blockchain Front-Running," in Proc. Workshop on Decentralized Internet, Networks, Protocols and Systems (DINPS), held in conjunction with IEEE ICDCS, 2022.

[11] M. Gnosis, S. Bartoletti, P. Schär, "Batch Auctions for Decentralized Trading," 2021. [Online]. Available: https://gnosis.io/research

[12] S. Kelkar, A. Juels, M. K. Reiter, and S. V. Lokam, "Aequitas: A Fair Ordering Service for the Blockchain," in Proceedings of the 2021 ACM SIGSAC Conference on Computer and Communications Security (CCS), Virtual Event, 2021, pp. 2841–2857.

[13] Z. Yang, H. Xu, Z. Wang, S. Chen, A. Gervais, "SoK: MEV Countermeasures – Theory and Practice," in Proc. ACM Workshop on DeFi and Security (DeFi '24), 2024.

[14] K. Qin, L. Zhou, P. Gamito, P. Jovanovic, A. Gervais, "An Empirical Study of DeFi Liquidations: Incentives, Risks, and Instabilities," in Proc. 21st ACM Internet Measurement Conference (IMC), 2021, pp. 336–350.

[15] L. Zhou, X. Xiong, J. Ernstberger, S. Chaliasos, Z. Wang, Y. Wang, A. Gervais, "SoK: Decentralized Finance (DeFi) Attacks," in Proc. IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2023.

[16] K. Qin, L. Zhou, E. Afonin, M. Lavi A. Gervais, "Attacking the DeFi Ecosystem with Flash Loans," in *Network and Distributed System Security Symposium (NDSS)*, San Diego, CA, USA, 2020.

[17] A. Lehar C. A. Parlour, "Systemic Fragility in Decentralized Markets," BIS Working Papers No. 1062, Bank for International Settlements, 2022.

[18] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, A. Juels, "Flash Boys 2.0: Frontrunning, Transaction Reordering, and Consensus Instability in Decentralized Exchanges," in Proc. IEEE Symposium on Security and Privacy (SP), San Francisco, CA, USA, 2019.

[19] M. Koegler, "SoK: Current State of Ethereum's Enshrined Proposer-Builder Separation," arXiv:2506.18189, 2025. [Online]. Available: https://arxiv.org/abs/2506.18189

[20] R. Arora, L. Gudgeon, D. Perez, "SecPLF: Secure Protocols for Loanable Funds against Oracle Manipulation Attacks," arXiv:2402.01887, 2024. [Online]. Available: https://arxiv.org/abs/2402.01887

[21] K. Qin, L. Zhou, E. Afonin, M. Lavi A. Gervais, "Flash Loans for Fun and Profit," in *Financial Cryptography and Data Security 2021*, 2021.

[22] MakerDAO, "Oracle Security Module (OSM)," 2019. [Online]. Available: https://docs.makerdao.com/smart-contract-modules/oracle-module

[23] S. Papagiannopoulos, "MEV-Resistant CDP Protocol," GitHub repository, 2025.

[Online]. Available: https://github.com/spiros-pap/mev-resistant-cdp

[24] Ethereum Improvement Proposal 4626, "Tokenized Vault Standard," 2022. [Online]. Available: https://eips.ethereum.org/EIPS/eip-4626

[25] V. Malamas, D. Koutras, P. Kotzanikolaou, "Uninterrupted Trust: Continuous Authentication in Blockchain-Enhanced Supply Chains," in Proc. 8th South-East Europe Design Automation, Computer Engineering, Computer Networks and Social Media Conference (SEEDA-CECNSM), Ioannina, Greece, 2023.

[26] Ethereum Foundation, "Smart Contract Security: Checks–Effects–Interactions," 2018. [Online]. Available: https://docs.soliditylang.org/en/latest/security considerations.html

[27] G. Caldarelli, J. Ellul, "The Blockchain Oracle Problem in Decentralized Finance: A Multivocal Approach," Applied Sciences, vol. 11, no. 16, art. no. 7572, 2021.

[28] V. Malamas, P. Kotzanikolaou, K. Nomikos, C. Zonios, V. Tenentes, and M. Psarakis, "HA-CAAP: Hardware-Assisted Continuous Authentication and Attestation Protocol for IoT Based on Blockchain," IEEE Internet of Things Journal, vol. 12, no. 11, pp. 15650–15666, 2025.

[29] OpenZeppelin, "OpenZeppelin Contracts," 2023. [Online]. Available: https://docs.openzeppelin.com/contracts/

[30] MakerDAO, "The Maker Protocol: MakerDAO's Multi-Collateral Dai (MCD) System," 2019. [Online]. Available: https://docs.makerdao.com/