Security Analysis of Encrypted Mempool

Prabal Das

Security Analysis of Encrypted Mempool

Dissertation Submitted in Partial Fulfillment of the requirements for the degree of

Master of Technology in Cryptology and Security

Submitted by

Prabal Das

Roll: CrS2313
MTech Cryptology and Security
Indian Statistical Institute, Kolkata

Supervised by

Prof. Dr. Ir. Bart Preneel
Department of Electrical Engineering
KU Leuven, Belgium

Prof. Mridul Nandi
Applied Statistics Unit
Indian Statistical Institute, Kolkata,
India

Daily Supervisor

Roozbeh Sarenche
Doctoral Researcher
COSIC, Department of Electrical Engineering, KU Leuven

July, 2025

Declaration

I hereby declare that this thesis entitled **Security Analysis of Encrypted Mempool**, submitted in partial fulfillment of the requirements for the award of the degree of Master of Technology in Cryptology and Security, and completed under the supervision of Roozbeh Sarenche, Prof. Dr. Ir. Bart Preneel, and Prof. Mridul Nandi at the Katholieke Universiteit Leuven, is an original and authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Signature and name of the student with date

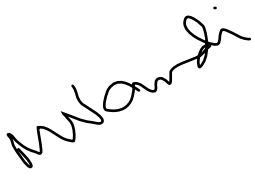
Probal Das

Prof. Dr. Ir. Bart Preneel

Signature

Prof. Mridul Nandi

Signature



Acknowledgements

I would like to express my heartfelt gratitude to everyone who has supported and guided me throughout the course of this thesis.

First and foremost, I am deeply thankful to my daily supervisor, Roozbeh Sarenche, for his continuous support and encouragement from the very first day. His insightful guidance, patience, and unwavering belief in my potential have been instrumental in shaping both the direction and quality of this work. I am especially grateful for the intellectually stimulating discussions and the motivating academic environment he provided throughout this journey.

I am also sincerely grateful to my supervisors, Prof. Bart Preneel (COSIC, KU Leuven, Belgium) and Prof. Mridul Nandi (Indian Statistical Institute, Kolkata), for their invaluable guidance and for offering me this wonderful opportunity to work in such a rich academic setting. Their mentorship and vision have been crucial to the development of this thesis.

I would like to extend my heartfelt thanks to my friends, whose presence gifted me countless beautiful days and cherished memories throughout this journey. Their constant support, encouragement, and companionship helped me stay focused and motivated.

Finally, I would like to acknowledge all the authors, researchers, and contributors whose work has been cited and referenced in this thesis. Their valuable contributions have provided the foundation upon which this research is built.

Abstract

With the rapid growth of Decentralized Finance (DeFi), the challenge of Maximum Extractable Value (MEV) has become increasingly significant-particularly on Ethereum. MEV allows malicious actors to manipulate transaction order within blocks, enabling exploitative strategies such as frontrunning and sandwich attacks. In response, recent research has proposed encrypted mempools, which conceal transaction content until after ordering is finalized, thereby reducing the exploitable surface for MEV.

This thesis investigates encrypted mempool, specifically the Shutter protocol, a threshold encryption-based approach designed to mitigate MEV by hiding transaction contents during the mempool phase. We analyze its core architecture, underlying cryptographic mechanisms, and proposed extensions, including its integration into Ethereum's Proposer-Builder Separation (PBS) framework.

Through this study, we identify several vulnerabilities in both the base Shutter protocol and its proposed Ethereum integration. We demonstrate how certain behaviors of proposers, builders, and smart contracts can be exploited to launch front-running attacks on encrypted transactions. As a proof of concept, we have also implemented this attack on a local blockchain environment to showcase its feasibility in practice. In addition, we propose mitigation strategies to address these issues and highlight open problems that need further investigations.

Contents

Chapter 1

Introduction

1.1 Decentralized Finance

Blockchain technology, with its decentralized, transparent, and tamper-resistant structure, has revolutionized the concept of trust in digital systems. One of its most impactful innovations is Decentralized Finance (DeFi) - a fast-growing movement that aims to replicate traditional financial services such as lending, borrowing, trading, and asset management on public blockchains like Ethereum, without relying on centralized intermediaries like banks or brokers.

Ethereum supports a wide variety of decentralized applications (dApps) powered by smart contracts—self-executing programs that operate on-chain and enforce protocol-specific rules without intermediaries. These dApps span financial services such as token exchanges (e.g., Uniswap [28]) and lending protocols (e.g., Aave [1]), as well as NFT marketplaces, oracle-based feeds, staking platforms, and cross-chain bridges. While these systems offer transparency and composability, they also introduce new attack surfaces-particularly related to transaction visibility and ordering. Although Maximum Extractable Value (MEV) is often associated with DeFi, it also emerges in a variety of other contexts, such as NFT minting, oracle price updates, cross-chain bridge relaying, and liquid staking reward cycles. In all these cases, adversaries exploit Ethereum's public mempool, where pending transactions can be observed and strategically manipulated, giving rise to the broader problem of MEV. According to DeFiLlama [12], Figure 1.1 illustrates the rising trend of DeFi activity on Ethereum.[1]

Figure 1.1: TVL in Ethereum based DeFi(source: [12])

[1] https://defillama.com/chains

1.2 MEV Problem

In the Ethereum network, transactions are initially sent to the *mempool* - a public waiting area where pending transactions are broadcast before they are included in a block. While this design promotes transparency and decentralization, it also creates a new class of vulnerabilities. One of the most critical is MEV[19], which refers to the profit that can be extracted by reordering, including, or excluding transactions within a block.

Who Benefits from MEV?
MEV is most often exploited by specialized actors known as *searchers*, who run bots to monitor the mempool and identify profitable opportunities. These bots craft and submit custom transactions to extract value using strategic timing. In proof-of-stake Ethereum, MEV can also be captured by *validators*, who have the power to determine the final ordering of transactions in blocks. Searchers typically pay validators (via tips or block bids) to prioritize their bundles, creating a shadow auction-like mechanism for transaction inclusion. The diagram 1.2 below illustrates the transaction workflow from the user's end to its inclusion in a block [2].

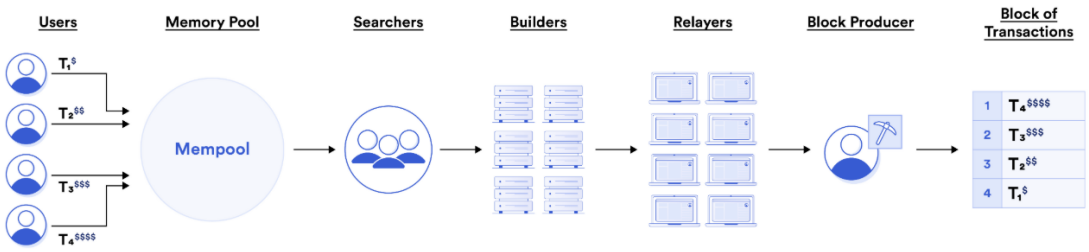


Figure 1.2: Transaction workflow(source: Chainlink)

Common MEV Strategies:

- **Front-running:** A searcher observes a profitable transaction (e.g., a large token swap on Uniswap) and submits a similar transaction with a higher gas fee to be included first. This causes the original transaction to execute at a worse price.
- **Back-running:** The searcher submits a transaction to run immediately *after* a profitable one. For example, if a liquidation is expected to cause a token price change, a back-runner might trade on the expected price shift.
- **Sandwich attack:** This combines front-running and back-running. The attacker places one transaction before and one after a user's swap to manipulate the price momentarily and capture profit from the price slippage.
- **Censorship and Exclusion:** In rare cases, validators may exclude specific transactions to protect or benefit others, violating Ethereum's neutrality assumptions.

How Ethereum's Mempool Enables MEV:
The root cause of MEV lies in the public visibility of transactions in the mempool.

[2] https://chain.link/education-hub/maximal-extractable-value-mev

Any user or bot can observe pending transactions and simulate their impact on DeFi protocols. Combined with Ethereum's *first-price auction* gas model - where transactions paying the highest gas fees are prioritized - this creates incentives for bots to bid aggressively to front-run others. Additionally, validators can choose which transactions to include and in what order, allowing them to collaborate with searchers to extract MEV.

Why Regular Users Lose
MEV harms users by increasing transaction costs and reducing the fairness of execution. In front-running, for instance, a user's trade is executed at a worse price because someone else got there first. In sandwich attacks, users suffer from artificially widened slippage. Furthermore, gas wars between searchers congest the network and raise base fees, indirectly impacting all users. These manipulative strategies are not just theoretical. According to Qin et al. [26], MEV yielded approximately $540.54 million USD in profit over a 32-month period from December 1, 2018, to August 5, 2021. As reported by Forbes [18] and Flashbots [16], over 500,000 ETH (exceeding $1.8 billion USD) has been extracted through MEV since the Merge.

1.3 Problem Statement and Our Contributions

Recent research efforts have proposed a variety of mitigation techniques, among which **encrypted mempools** have gained attention as a promising direction. These systems aim to preserve transaction confidentiality by temporarily hiding transaction contents from the public mempool until a specific condition (e.g., consensus or block inclusion) is met. While such mechanisms may reduce the opportunities for MEV extraction, their security and practical implementation remain complex and underexplored.

In this thesis, we study one such implementation: the **Shutter protocol**, a threshold encryption-based approach designed to build encrypted mempools for Ethereum. Our work investigates its architecture, security assumptions, and integration into Ethereum's Proposer-Builder Separation (PBS) pipeline. PBS separates block building from block proposal: instead of having the block's proposer perform both tasks, specialized builders are responsible for creating blocks. Based on this study, our contributions are as follows:

- We conduct an in-depth analysis of the **Shutter protocol**, focusing on its use of threshold encryption to hide transaction contents during the mempool phase, and identify several **vulnerabilities and potential attacks** related to the ordering, decryption, and execution of encrypted transactions.
- We implement a **proof-of-concept attack** that demonstrates how one of these vulnerabilities can be exploited in practice to do a front-running attack on encrypted transaction. This serves to highlight the real-world risks involved in deploying such privacy-preserving solutions.
- We propose a **rough conceptual solutions** to mitigate the identified vulnerabilities by introducing a fairness-based modification to the transaction ordering and slashing mechanism.
- We analyze the recent whitepaper released by the Shutter proposing an encrypted mainchain design, and uncover additional design limitations and security gaps in that architecture as well.

Our study highlights the **open challenges and trade-offs** inherent in encrypted mempool systems, emphasizing how subtle design choices in smart contracts can render them **vulnerable to manipulation or exploitation**, thereby contributing to both the evaluation of such systems and the broader discourse on secure and fair transaction ordering in Ethereum.

1.4 Thesis Structure

The rest of the thesis is organized as follows:

- **Chapter 2: Literature Review**

  This chapter presents two broad schools of thought addressing MEV problem and existing solutions. It then introduces the concept of encrypted mempools, followed by a detailed discussion of the Shutter protocol, including the cryptographic primitives it employs. Finally, it reviews the proposal for a Shutterized Ethereum Mainchain, which integrates Shutter within Ethereum's proposer-builder separation (PBS) architecture.

- **Chapter 3: Attack and Vulnerabilities**

  This chapter identifies several vulnerabilities and potential attacks within the Shutter protocol and its mainchain integration. It also presents possible mitigation strategies to address the weaknesses uncovered.

- **Chapter 4: Implementation: Front-Running Encrypted Transactions**

  This chapter focuses on the implementation of a specific attack-front-running of encrypted transactions. The attack exploits a smart contract working flows and execution pipeline that enables adversaries to front-run even encrypted transactions.

- **Chapter 5: Conclusion and Future Work**

  The final chapter summarizes key findings, outlines proposed mitigation approaches, and highlights promising directions for future research and protocol improvement.

Chapter 2

Background and Literature Review

2.1 Two Schools of Thought on MEV

The issue of MEV has sparked significant debate within the blockchain community, leading to two dominant and contrasting schools of thought on how MEV should be addressed in permissionless systems.

1. Accept and Democratize MEV

This perspective recognizes MEV as an inevitable byproduct of transparent and permissionless blockchain architectures. Rather than attempting to eliminate it, this approach focuses on designing mechanisms to make MEV extraction more equitable and less harmful. The goal is to enable:

- *Efficiency* - Preventing gas wars through out-of-band bidding mechanisms,
- *Decentralization* - Reducing censorship risks by separating roles among participants,
- *Transparency* - Maintaining user trust by clearly exposing extraction mechanisms.

2. Eliminate or Minimize MEV

This school of thought treats MEV not as an economic opportunity, but as a critical vulnerability that threatens fairness, decentralization, and user trust. Proponents advocate for minimizing or eliminating MEV through technical innovations that reduce visibility and manipulability of pending transactions.

Notable strategies in this category include:

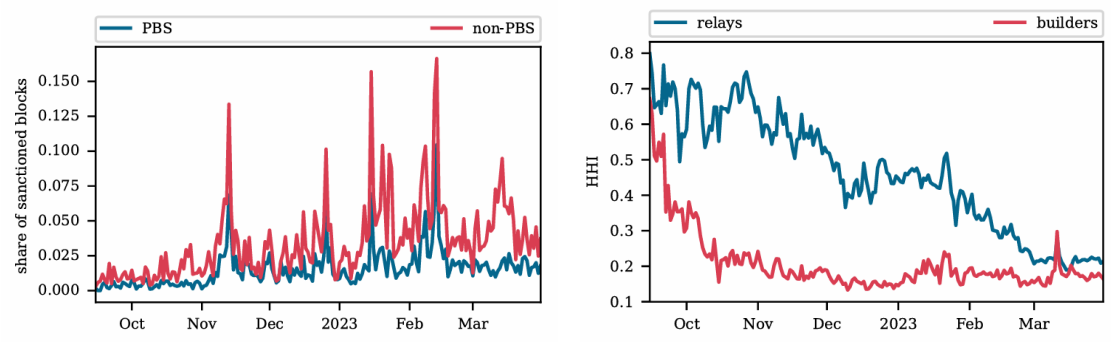
- *Encrypted mempools*, which conceal transaction details until after ordering is fixed,
- *Threshold encryption*, where a group collaboratively controls transaction decryption,
- *Fair ordering protocols*, which aim to order transactions based on external, unbiased metrics.

2.2 Existing Solutions

Proposer-Builder Separation

Proposer-Builder Separation (PBS) [14] is a design paradigm proposed not to eliminate MEV, but to **fairly distribute** it among participants by decoupling the roles of block proposing and block building in Ethereum's consensus protocol. This separation aims to mitigate centralization and censorship risks by preventing any single party-particularly validators-from unilaterally controlling the block contents and extracting MEV exclusively. Instead of having the validator (proposer) construct the block themselves, PBS enables external entities called builders to assemble blocks and submit them to the proposer through relayers. This separation aims to reduce the technical burden on validators, promote specialization, and create a fair and transparent MEV extraction process. **MEV-Boost** [17] is an implementation of PBS developed by Flashbots, which acts as a middleware allowing proposers to receive pre-built blocks from builders via trusted relays.

However, recent analysis by Heimbach et al. [22] suggests that these systems may not fully realize their goals. In practice, a small number of builders and relays dominate the ecosystem, reintroducing centralization and censorship risks. Additionally, proposers may not always receive the promised MEV share, undermining incentives and trust. Herfindahl-Hirschman Index (HHI) [29] is a statistical measure of the concentration of an industry and the higher the HHI, the higher the concentration of the industry. Figure 2.2a shows even though the concentration of the relay and builders decrease with time, it remains concentrated. Also, figure 2.1a shows how PBS currently falls short of its second design goal i.e. censorship resistance.



(a) Daily share of PBS and non-PBS blocks that include transactions that do not comply with OFAC sanctions

(b) Relay and builder HHI over time

Figure 2.1: Current state of PBS(source: [22])

Trusted Execution

One approach to mitigate MEV is to leverage Trusted Execution Environments (TEEs) [30] to protect transaction confidentiality. In such a design, the TEE enclave holds the secret key of a public-key encryption scheme, while users encrypt their transactions using the corresponding public key. These encrypted transactions are submitted to the network and can only be decrypted inside the enclave. The TEE then decrypts and publishes an ordered block of transactions, which is subsequently included on-chain. The idea is to prevent external actors, including validators and searchers, from accessing transaction contents before block finalization.

However, this approach comes with serious limitations. Despite their theoretical appeal, TEEs-such as Intel SGX [10]-have repeatedly been shown to suffer from side-channel vulnerabilities and hardware-based attacks[1]. Moreover, placing such a critical trust assumption on a single hardware component introduces a central point of failure. In the context of MEV, where over $1 billion USD has already been extracted[2], the economic incentive to compromise or exploit such enclaves is significant. Therefore, while TEE-based designs offer an interesting direction, they are unlikely to provide a secure long-term solution unless hardware security improves substantially or is combined with additional layers of decentralization and verification.

Cryptographic Approaches

In this category, clients rely on cryptographic techniques to hide the contents of their transactions until the execution is finalized and the transaction ordering is fixed-thereby preventing front-running and other MEV-driven exploitation. Three major approaches have been proposed in this direction, with the most notable being timelock encryption and threshold encryption.

In timelock encryption[[8]], a transaction is encrypted in such a way that it can only be decrypted after a certain period has passed. This provides a time-limited privacy window during which the transaction remains hidden from the public mempool. However, if the transaction is not included in a block within this window, its confidentiality may be compromised. Additionally, timelock encryption often relies on computational puzzles (like Verifiable Delay Functions), which are resource-intensive and may introduce latency or computational overhead-making them less ideal for time-sensitive or high-throughput environments.

In contrast, threshold encryption [8, 9, 5, 25] enables a group of designated participants - typically called a "committee"- to collectively control the decryption process. Transactions are encrypted using a public key derived from the committee, and only when a threshold number of members cooperate can the transaction be decrypted. When built with adaptive chosen ciphertext security, threshold encryption schemes offer strong privacy guarantees even against powerful adversaries. However, a couple of key limitations are the communication complexity grows linearly with both the number of transactions and the size of the committee as well as the honesty of the committee members. This creates significant scalability challenges, especially for high-volume chains like Ethereum, where thousands of transactions may need to be decrypted in each block.

Order-Fairness

An alternative strategy for mitigating MEV focuses on establishing a consensus over the ordering of transactions as they arrive in the mempool. Instead of relying on individual validators or miners to determine transaction order—which enables manipulative behaviors like front-running—these approaches aim to enforce a form of order-fairness at the protocol level. The core idea is that if all participants agree on the sequence in which transactions were received, then the ability to reorder them for MEV extraction is significantly reduced.

Protocols like and Aequitas [24] exemplify this principle of decentralized order-fairness. They seek to construct a globally consistent transaction order that reflects

[1] https://sgx.fail/

[2] https://explore.flashbots.net/

when transactions were first observed by the network. However, achieving strict total ordering in decentralized systems is challenging due to network latency and asynchronous communication. As a result, these protocols typically provide only a weaker guarantee known as batch order-fairness, where fairness is enforced over groups (batches) of transactions rather than individual ones. While this reduces opportunities for MEV extraction, it does not fully eliminate them-particularly in high-frequency trading scenarios or when transactions within a batch can still be manipulated.

Our Perspective and Research Focus

While both perspectives acknowledge that unregulated MEV extraction harms the ecosystem, they diverge in their solutions - one seeks to reduce MEV to the greatest extent possible, while the other aims to control and fairly redistribute it.

Our work aligns with the second school of thought. We focus on preventing MEV at its root by exploring **encrypted mempool designs**, particularly those based on threshold encryption schemes. To this end, we are studying and evaluating various implementations - such as the Shutter protocol - to identify limitations, propose improvements, and assess the feasibility of integrating these solutions into Ethereum's future architecture.

2.3 Workflow for Encrypted Transactions

In Ethereum, the mempool (short for memory pool) is where pending transactions wait to be included in a block by validators. In case of Encrypted Mempools, all transactions are encrypted so that it can hide the transaction data to prevent early access. There are different frameworks for encrypted mempool design, and follow is widely accepted:

- Users encrypt their transactions using a shared public key and submit the ciphertext to the public mempool, optionally alongside minimal metadata (e.g., gas info).
- Validators include encrypted transactions in a block without knowing their actual contents and propose the block in the network.
- After block proposal, a decryption process is triggered (e.g., via threshold decryption). Sometime there are special entities who are responsible for key generation as well as decryption.
- Once decrypted, the transactions are executed in the blockchain, making their effects and data visible on-chain.

2.4 Shutter Network

Shutter network[3] is a practical implementation of encrypted mempool that enables fairness in blockchain through threshold encryption and Distributed Key Generation (DKG). We will discuss Shutter protocol, it's underline cryptographic protocol and lastly discuss the implementation on testnet, as described in their whitepaper [13].

[3] https://www.shutter.network/

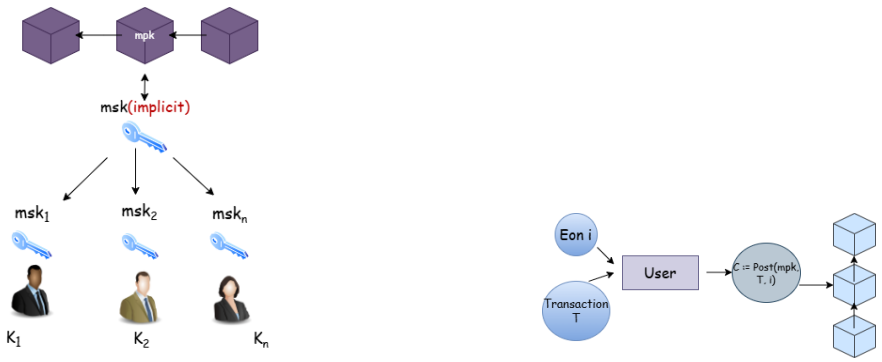
2.4.1 Application of Shutter Protocol

The Shutter system was initially created to mitigate MEV attacks on public blockchains and is actively employed for this purpose on the Gnosis Chain[4]. Beyond this use case, Shutter has broader applicability. For instance, it is currently used to enable shielded voting for DAOs, where maintaining vote privacy is essential. Through Shutter, the content of each vote stays confidential. This functionality is already integrated into the Snapshot[5] voting platform. One such use is condition-based decryption, where messages are encrypted and only revealed upon fulfilling specific criteria—such as the passage of time or occurrence of an event. Another potential application is censorship resistance on both Layer 1 and Layer 2 networks: by encrypting transaction content, it becomes difficult for miners to selectively censor or be coerced into censoring particular transactions.

2.4.2 High level idea of the Shutter protocol

Shutter operates on the principle that users encrypt the contents of their transactions, which are then collectively decrypted at a later stage. The protocol includes a group of users $\{u_1, u_2, \cdots, u_m\}$ and a set of designated key holders known as keypers $\{k_1, k_2, \cdots, k_n\}$. All parties interact through a public ledger L, such as Ethereum, which supports smart contract functionality. It is assumed that the majority of the keypers behave honestly. The protocol works in following steps:

- **Setup Phase:** At the beginning of the protocol, a setup phase is carried out in which the keypers collaboratively generate the system parameters. This process produces a master public key mpk which is published on the ledger, and a master secret key msk which is never exposed to any individual. Instead of existing explicitly, the msk is implicitly present in the system as it is secret-shared among the keypers.



(a) Setup (b) Encrypted transaction on ledger

Figure 2.2

- **Encryption and Submission:** The protocol runs in eon, $i \in \mathbb{N}$. User selects an eon, encrypts their payload T using the master public key mpk and send as a standard Ethereum transaction to the mempool. Threshold encryption scheme with Distributed Key Generation (DKG) is used as a cipher, which we will discuss shortly. The encrypted transactions $C := Post(mpk, T, i)$ are then

[4] https://www.gnosischain.com/
[5] https://snapshot.box

included in a block by a proposer, who has full discretion over its placement and ordering within the block.

- **Key Share Publication and Voting:** After the eon finishes, decryption process is started by the keyper nodes. They do communicate with themselves in an another Tendermint based chain, called Shuttermint chain. Each keyper has a secret share of msk, from which they generate corresponding share to the eon and broadcast it in this chain. After getting everyone's share, each construct the eon's secret key and do the partial decryption. They are not fully decrypting C, instead compute σ from which anyone can recover the transaction. Since everyone individually compute σ for each transaction T, there is a voting phase among the keypers which decide the majority σ for a transaction.

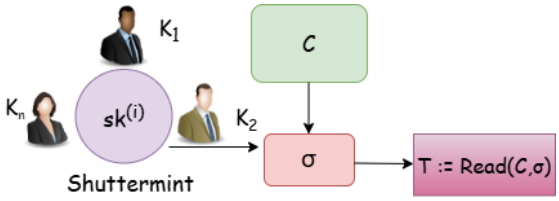


Figure 2.3: Eon key generation and decryption

- **Decryption and Execution:** After proof σ is sent to the ledger, it will be used in $T := Read(C, \sigma)$ for decryption. One keyper is selected to complete the decryption of all partially decrypted transactions, which are then sent to the ledger. After decryption, it will be sent to the target contract for execution. We will see shortly how exactly this execution process happens and functionality of different smart contracts.

The Voting Procedure

Let t denote the maximum number of corrupt parties. A contract on the main chain includes a function called `vote` that accepts two parameters:

- **vote_id** – a unique identifier for the specific voting process, and
- a value v.

Whenever a keyper K_i invokes `vote` with parameters `(vote_id,` v `)`, the function executes the following steps:

1. If K_i has previously called `vote` with the same `vote_id`, the current call is disregarded.
2. Otherwise:
   (a) If no keyper has previously called `vote` with parameters `(vote_id,` v `)`, the contract records `(vote_id,` v`)` with an initial label $i := 1$.
   (b) If another keyper has already called `vote` with `(vote_id,` v `)`, the contract increments the label of this entry by 1. If this label exceeds t then this function changes this label to **"agreed"** and ignores any further calls of vote with parameter *vote_id*.

2.4.3 Encryption Mechanism

To ensure confidentiality and prevent MEV exploitation, Shutter leverages advanced cryptographic primitives. At its core, it employs a distributed Boneh-Franklin identity-based encryption (IBE) scheme, which offers CCA-security and enables threshold-based decryption of transactions. This section outlines the cryptographic components and security guarantees underlying Shutter's design.

Bilinear maps

A **bilinear map** is a function $e : \mathbb{G}_1 \times \mathbb{G}_2 \to \mathbb{G}_T$, where G_1 and G_2 are additive cyclic groups of prime order q, and $\mathbb{G}_T$ is a multiplicative cyclic group of the same order, satisfying the following:

1. *bilinearity*: for all $m, n \in \mathbb{Z}_p$ and $A \in \mathbb{G}_1$, $B \in \mathbb{G}_2$, we have
$$e(mA, nB) = e(A, B)^{mn}$$
2. *non-degeneracy*: $e(A, B) \neq 1$ if A and B are generators of $\mathbb{G}_1$ and $\mathbb{G}_2$
3. *efficient computability*: there exists an efficient algorithm to compute $e(A, B)$.

Distributed Boneh-Franklin encryption

We now present the distributed version of the Boneh-Franklin identity-based encryption scheme, initially outlined in [5] and later developed more thoroughly in [23]. To maintain consistency with the terminology used throughout this paper, we refer to the parties executing the protocol as keypers, denoted $\{K_1, K_2, ..., K_n\}$.

We employ Shamir's (t, n)-threshold secret sharing scheme [27] to guarantee that at least $t + 1$ keypers are required to reconstruct the master secret key msk, while any group of t or fewer keypers learns nothing about it. In this scheme, a secret s is shared by choosing a random polynomial ϕ with $\phi(0) = s$, and each keyper K_i receives the share $\phi(i)$.

The distributed Boneh-Franklin identity-based scheme is a pair (`DistrSetup`, `DistrExtract`, `Encrypt`, `Decrypt`) of distributed algorithms, where `DistrSetup` is for key generation algorithm in distributed setup, `DistrExtract` is the distributed key extraction algorithm, and `Encrypt` and `Decrypt` are encryption and decryption algorithms resp.

- `DistrSetup()`: Each keyper K_i does the following in parallel:
    1. Choose $s^i \xleftarrow{\$} \mathbb{Z}_q$
    2. Execute `FeldmanVSS(`K_i, s^i`)` [15], where Shamir secret shares of s^i will be generated along with proof for verifiability $(\pi_0, \pi_1, \cdots, \pi_n)$, $\pi_0 = s^i \times P_2, \pi_j = \phi(x_j) \times P_2$ (P_2 is a generator of $\mathbb{G}_2$).
    3. Once all execution is over, each keyper K_i computes its private output $msk_i = s_i^1 + \cdots s_i^n$ mod q, where s_k^i is the private input of K_k of share s_i. The public output of each keyper is $(mpk_1, \cdots, mpk_n)$, where $mpk_j = \pi_j^1 + \cdots + \pi_j^n$, and master public key is $mpk := \pi_0^1 + \cdots + \pi_0^n$.

- `DistrExtract(`i`)`: For each identity $i \in \mathbb{N}$, the private key is $sk_i = msk \times H_1(i)$, where $H_1 : \mathbb{N} \to \mathbb{G}_1$ be a hash function. Now, the secret key msk is generated implicitly, so to generate the identity's private key we need each keyper's masked secret key $msk_i \times H_1(i)$ and use Lagrange interpolation for computing the key.

- `Encrypt(`T, ID, mpk`)`: Users will encrypt transaction $T = T_1||T_2||\cdots||T_n$ for epoch id ID with master public key mpk by following way:

$$r := H_3(\sigma, T), \sigma \xleftarrow{\$} \{0,1\}^k$$

Then, compute

$$C_1 := r \times P, P \in \mathbb{G}_2$$

and

$$C_2 := \sigma \oplus H_2(e(ID, mpk)^r)$$

and

$$C_3 := (H_4(\sigma||1) \oplus T_1, \cdots, H_4(\sigma||n) \oplus T_n)$$

So, the cipher text is the tuple $(C_1, C_2, C_3 = (C_3^1, \cdots, C_3^n))$.

- `Decrypt(`$sk_{ID}, (C_1, C_2, C_3)$`)`: Eon secret key $sk_{ID} = msk \times H_1(i)$ is generated from threshold many share from the keypers. Keypers mask their share msk_i with $H_1(i)$ and broadcast it to Shuttermint chain. Then they individually generate the eon secret key sk_{ID} and do the decrytion as follows:

$$\sigma := C_2 \oplus H_2(e(sk_{ID}, C_1)),$$

and

$$T := (C_3^1 \oplus H_4(\sigma||1))||\cdots||(C_3^n \oplus H_4(\sigma||n))$$

and

$$r := H_3(\sigma, T)$$

Security

Security follows from the underline hard problem, *Bilinear Diffie-Hellman (BDH) problem*. [5] showed in their paper that BF-IBE scheme is IND-ID-CCA secure against any poly-time adversary that corrupts at most t-many keypers. Non-malleability follows from the fact that every IND-ID-CCA secure scheme is non-malleable.

2.4.4 Implementation

To demonstrate the practicality of the protocol, a production-ready implementation of Shutter[6] has been developed for the on-chain setting. The keyper nodes coordinate over a customized Tendermint blockchain, while an Ethereum-compatible chain is used for transaction ordering and execution.

The block sequence on this Ethereum-compatible chain is partitioned into eons of configurable length. Each eon is associated with a unique key pair, and the keypers collaboratively generate the corresponding decryption key at the end of the eon.

Throughout its lifecycle, a Shutter transaction interacts with three smart contracts on the underlying chain: the batcher, the executor, and the target contract.

[6] https://github.com/shutter-network/shutter

To send a transaction, a user selects an eon and encrypts their payload using the respective public encryption key. They then wrap the ciphertext in a standard Ethereum transaction, along with some metadata, and send it to the batcher contract. The batcher logs the transaction's arrival, provided the batch is within size limits and the eon is valid (i.e., not expired or too far in the future).

At the end of an eon, keypers publish their key shares to the Tendermint chain. They collect shares from others and, once enough are gathered, reconstruct the decryption key. They then decrypt the transactions submitted to the batcher contract for that eon, sign the results, and post their signatures as votes on the Tendermint chain. A designated keyper submits the decrypted transactions to the executor contract. If they fail to do so within a timeout, the next in line takes over.

The executor contract forwards the decrypted transactions to the target contract, which represents the application logic (e.g., a DEX). This contract handles tasks like authentication and replay protection. Transactions are protected against front-running by other transactions handled by the sequencer, ensuring that a DEX which accepts only sequencer-originated transactions remains secure from front-running attacks.

If a keyper submits incorrect results, they may be challenged to present signed votes from peers as proof of honest behavior. Failure to do so results in a penalty, such as the forfeiture of a previously deposited bond. This optimistic verification model-where checks are only triggered on disputes-is more efficient than verifying every step, and acceptable in many practical scenarios.

Ideally, keypers would only provide key shares, leaving decryption entirely to the blockchain. However, current gas costs make this infeasible. Upcoming improvements to the Ethereum Virtual Machine (EVM) may enable such a shift in the future.

2.5 Further improvement of Shutter protocol

Although the Shutter protocol ensures strong confidentiality for pending transactions within an eon, it is not without limitations. A notable concern arises with encrypted transactions that are submitted but not included in a block before the eon concludes. Once the decryption key is revealed at the end of the eon, these unexecuted transactions remain in the mempool and become decryptable-exposing potentially sensitive information despite never being processed. This is known as **Pending Transaction Privacy**. Recent research has explored ways to mitigate this issue:

Mempool Privacy via Batched Threshold Encryption by Garg et al.[8]

This approach combines an MPC-based setup with a per-epoch configuration and leverages the **KZG polynomial commitment scheme** for confidentiality. At its core, the construction relies on a commitment scheme that enables a user to commit to a polynomial $p(X)$ by generating a commitment com. Later, the user can provide a succinct proof that opens com to specific input-output pairs (x, y), ensuring that $p(x) = y$.

During encryption, each user samples a point (x, y), and for a batch of B transactions, the committed polynomial is constructed to interpolate through the selected points. After the batch is formed, the commitment is revealed and proof of opening is used for decryption. For any ciphertext associated with a point (x', y') not

included in the committed batch, and assuming (x', y') is randomly sampled, it will-with overwhelming probability-not satisfy $p(x') = y'$. Due to the **binding property** of the KZG scheme, the commitment com, once tied to $p(X)$, cannot be opened to an unrelated pair (x', y'). This guarantees that ciphertexts encrypted with such unmatched pairs remain hidden. To decentralize the setup and eliminate the need for a trusted server, a **multi-party computation (MPC)** protocol is employed to securely generate and manage the per-epoch commitment parameters.

Unlike the naive approach to threshold encryption, which requires $\mathcal{O}(nB)$ communication to decrypt B transactions with a committee of n parties, the batched-threshold encryption scheme reduces the communication cost to just $\mathcal{O}(n)$ as well as provide pending transaction privacy. Rather than relying on distributed key generation, this construction employs an MPC-based per-epoch key generation process. While this enables decentralization and enhances security, it introduces additional overhead due to the inherent complexity of the MPC setup.

Practical Mempool Privacy via One-time Setup Batched Threshold Encryption by Garg et al.[9]

When a user wishes to encrypt a message for a given epoch eid, it utilizes a **witness encryption scheme** (cf. equation (2.1)) in conjunction with the **pairing product equations** (cf. equation (2.2)) to produce the ciphertext. Alongside the ciphertext, the user provides a *non-interactive zero-knowledge proof of knowledge* to attest that the encryption was performed correctly.

Once a batch of B ciphertexts is selected, the committee extracts the associated tuples (x_i, tg_i) and uses them to interpolate a polynomial. They then compute the corresponding **KZG polynomial commitment** com. The committee signs the pair $(H(\mathsf{eid}), \mathsf{com})$ using a *threshold BLS signature* scheme and broadcasts the resulting signature.

Any user can now locally recompute com, derive the necessary *opening proofs* for the selected ciphertexts, and finally use the broadcast signature to recover the witnesses required for decryption. Since only a single signature is communicated, this approach achieves the desired communication efficiency outlined in the protocol's design goals.

Moreover, the committee performs a **distributed key generation (DKG)** only once at the beginning of the protocol to obtain Shamir shares of the BLS signing key. Intuitively, for an adversary to break this system, it must either forge a BLS signature on a commitment com' of its choice or break the *binding property* of the KZG commitment by opening a fixed com to a forged point (x', tg').

$$e(g^A, h^x) = g_T^b \tag{2.1}$$

$$e(\sigma, h) \cdot e(\mathsf{com}, \mathsf{pk}) = e(H(\mathsf{eid}), \mathsf{pk}) \tag{2.2}$$

2.6 Shutterized Ethereum Mainchain

A group of researchers from Shutter Network[7], Gnosis Chain [8], MEV Blocker[9], Nethermind[10], Chainbound[11], and others have proposed a practical roadmap for integrating threshold-encrypted mempools into Ethereum's Proposer-Builder Separation (PBS) architecture [20]. Their work explores how the Shutter protocol can be incorporated into the Ethereum mainchain by leveraging proposer commitments and aligning with the PBS transaction supply chain. The proposal also discusses integration into existing systems such as RPC endpoints (e.g., MEV Blocker) to support encrypted transaction submission, along with the inclusion of relays in the encrypted transaction pipeline.

2.6.1 Transaction Workflow in the PBS Setup

- **Encrypted Transaction Submission:** Users encrypt transactions and encrypted transactions are submitted to the Bolt protocol [4] through one of the available RPC interfaces. Bolt coordinates with the next block proposer to manage their inclusion. To support Shutter integration, Bolt introduces a dedicated "proposer commitment" type, which preserves transaction confidentiality by withholding decryption keys until the proposer has finalized their commitment.

- **Proposer Commitment:** The proposer signs and publishes a commitment to a specific ordering of encrypted transactions, ensuring their inclusion at the top of the block (ToB). This commitment is broadcast as a public promise that the prioritized transactions will be included in the specified order.

- **Relay Participation:** The relayers also receive encrypted transactions submitted by users and securely stores them along with their corresponding decryption information and intended inclusion order. It then generates a commitment to these transactions, guaranteeing their inclusion in a specific order within the block. This commitment is broadcast to the network to ensure transparency and accountability.

- **Key Release and Decryption:** Once the proposer's commitments as well as relayer's commitments are confirmed, the Shutter Network's Keypers release the decryption key. The key is sent securely to the Bolt as well as relayers and they start decryption. Then Bolt will send the decrypted transactions to block builders.

- **Block Construction by Builders:** Builders construct blocks by respecting the proposer's committed transaction ordering. They can also include additional transactions of their own after the proposer's committed transactions which can be an effective backrunning. Then builders send the block to the relayers for bidding.

[7] https://www.shutter.network/
[8] https://www.gnosischain.com/
[9] https://cow.fi/mev-blocker
[10] https://www.nethermind.io/
[11] https://chainbound.io/

- **Relay Appending:** Upon receiving a block proposal from a block builder, relays append their committed transactions to the block and then participate in the PBS bidding process.

- **Block Proposal:** Finally, the proposer selects the most profitable block from the bids and proposes it to the chain. It can reject block if it does not follows the proposer's commitment.

Although this architecture aims to preserve privacy and fairness in transaction ordering while aligning with Ethereum's future direction, it has some inherent limitations like:

- **Latency:** The decryption of transactions and the subsequent recalculation of the state root introduce a minor delay in the process. Both the commitment and decryption phases contribute to this latency.

- **Handling rejected blocks:** There is no guarantee that the proposer will accept the modified block. If the proposer rejects the relay's modified block, the relay is required to hold onto the decrypted transactions and try to include them in a future block.

- **Top of the Block Commitment:** The top of the block is a highly valuable position for transaction inclusion, and builders are generally unwilling to reserve it for specific transactions ahead of time. However, since proposers only accept blocks that match their commitments, this enforces the placement of decrypted transactions at the top of the block.

These are the some of limitations that the proposed architecture inherently holds. However, in our study we have figured out some potential attacks and vulnerabilities of this scheme, which we will discuss in the following chapter.

Chapter 3

Proposed MEV Attacks and Vulnerabilities

3.1 Toward a Formal Definition of MEV Attack

Smart contracts are autonomous programs on blockchains that offer strong security guarantees-such as integrity, transparency, and censorship resistance-which centralized systems often lack. These properties have enabled a massive ecosystem including stablecoins, lending protocols, decentralized exchanges, and NFTs.

However, blockchains cannot guarantee transaction ordering. This opens the door for powerful actors to manipulate order for profit. For instance, if Alice sees Bob's transaction and submits hers just before it, she can frontrun him. This type of profit extraction is known as Miner (or Maximal) Extractable Value (MEV), a term introduced by Daian et al. [11], referring to the gains from selectively including, excluding, or reordering transactions apart from the standard block fees.

MEV raises serious concerns. It can directly harm users—as seen in sandwich attacks that have cost victims millions—and leads to inefficiencies, like on-chain bidding wars that congest networks and increase gas fees. In some cases, MEV competition shifts off-chain, resulting in latency wars.

Beyond performance, MEV threatens consensus security. Research by Carlsten et al. [7] has shown that when transaction fees outweigh block rewards, miners are incentivized to fork the chain in pursuit of higher profits, rather than following the protocol honestly. MEV also encourages centralization, as discovering profitable strategies often requires resources only large players possess. This can lead to opaque, vertically integrated systems that undermine the transparency and decentralization of blockchains.

Given the growing practical significance of MEV, numerous research efforts have emerged to better understand its mechanics and implications. However, unlike traditional cryptographic research-which relies on rigorous formal definitions of security properties and adversarial capabilities-the formalization of MEV remains a complex and evolving challenge. This difficulty stems primarily from the need to characterize a powerful and multifaceted adversary: (i) one who can influence block construction by crafting and inserting their own transactions, and (ii) whose identity or wealth is largely irrelevant to their ability to extract MEV. Capturing the full range of such adversarial capabilities, knowledge, and attack vectors requires a nuanced and comprehensive formal model. While some initial attempts have been made to formalize MEV[[3], [6]], they are still in early stages and fall short of fully encapsulating the intricacies of the problem.

3.2 Attacks and Vulnerabilities

In this section, we present a series of attacks targeting both the original Shutter protocol and the recently proposed shutterized Ethereum architecture. Our work includes one implemented attack on the Shutter protocol, which serves as a proof of concept for the feasibility of exploiting certain design weaknesses. In addition to this, we propose several theoretical attacks against the shutterized Ethereum architecture. While these have not been implemented-due to the architecture still being in the design and proposal phase-they highlight critical security concerns and practical limitations that could arise if the architecture were deployed in its current form. Our analysis serves both to demonstrate practical vulnerabilities and to motivate more robust cryptographic and system-level safeguards in future iterations of Shutter-based systems.

3.2.1 Proposer Abandonment Attack

The **Proposer Abandonment Attack** targets the mechanism by which decrypted transactions are included in a block in the shutterized Ethereum architecture. In this design, proposers are expected to commit to a batch of encrypted transactions and subsequently reveal their decrypted versions for inclusion at the top of a block, as mandated by their earlier commitment. However, this process introduces an opportunity for strategic misbehavior.

In this attack, a malicious proposer prepares a valid block containing a batch of correctly decrypted transactions, as per their prior commitment, but then deliberately chooses not to publish the block. This strategic abandonment breaks the intended guarantee that the committed decrypted transactions will appear at the top of the chain. The consequences of this action diverge based on how the network handles the now-revealed decrypted transactions:

- **If the decrypted batch is picked up and included in a subsequent block**, it no longer occupies the top position it was originally guaranteed. The new proposer would have already committed to a different transaction ordering, meaning the previously top-ordered transactions could be demoted within the block. This exposes them to frontrunning and other forms of reordering-based manipulation, defeating the purpose of shutter-based confidentiality and priority enforcement.
- **If the decrypted transactions are not included in the next block**, they continue to linger in the public mempool, now fully decrypted and visible to all participants. In this state, they become vulnerable to front-running and other MEV attacks.

Real-World Scenario: Front-Running a Large DeFi Swap

Consider a user submitting a large token swap on a decentralized exchange (DEX) such as Uniswap, intending to exchange 200 ETH for DAI. To preserve confidentiality and avoid MEV, the transaction is submitted through the encrypted mempool as designed in the shutterized Ethereum architecture. The encrypted transaction is committed for inclusion at the top of the next block by the current block proposer.

The shutterized system guarantees that this transaction will be decrypted and placed at the top of the block, making it resistant to frontrunning under normal operation. However, the malicious proposer observes that the decrypted swap will

significantly impact the ETH/DAI price on Uniswap due to slippage. This creates a clear MEV opportunity: if another swap could be inserted before this large trade, it would result in an arbitrage profit.

Rather than publishing the decrypted block, the proposer abandons the block, allowing the committed transactions to linger in the mempool. In the meantime, the malicious proposer submits their own transaction (or shares the information with a collaborating MEV bot) that performs a smaller ETH-to-DAI trade before the large swap, benefiting from the price impact.

In the next block, a new proposer includes the original user's transaction-now decrypted-but it no longer occupies the top position. The front-run transaction executes first, and the original user suffers worse execution (i.e., poor slippage), while the attacker profits.

Slashing as a Potential Mitigation

One possible mitigation for the proposer abandonment attack is to introduce slashing penalties-economic disincentives that penalize proposers for failing to publish blocks after committing to decrypted transactions. This mechanism aligns incentives by making abandonment financially costly.

Additionally, the reputation of a proposer can serve as a deterrent against repeated instances of such attacks, as consistent misbehavior may lead to diminished trust or exclusion from future protocol participation.

Challenges of Slashing Mechanism

While slashing is effective in theory, it is not always a reliable deterrent in practice-particularly in decryption-based systems where timing is critical. A proposer might fail to publish a block not out of malicious intent but due to unavoidable issues such as network latency, delayed decryption outputs from keypers, or clock desynchronization between nodes. In such scenarios, penalizing proposers could result in unjust slashing, harming honest participants.

These limitations highlight the need for a fault-tolerant design that can distinguish between malicious and unintentional failures. Designing such a balanced mechanism remains an open challenge.

3.2.2 Builder Exploitation Attack

A notable vulnerability arises from the deterministic ordering of transactions within blocks in the shutterized Ethereum architecture, where the block typically includes the proposer's committed transactions first, followed by those from the builder, and finally the relays. While this ordering is intended to preserve commitment integrity, it can be subtly exploited by adversaries-particularly builders-in the Proposer-Builder Separation (PBS) paradigm.

In this setting, builders are incentivized to maximize the profitability of blocks by continuously building blocks with new opportunities. Given that the transactions committed by the proposer and relays are already fixed, a builder can simulate various combinations of blocks around these transactions to identify the most lucrative opportunities. In particular, they may insert their own transactions between those of the proposer and the relays in a way that front-runs the relay-submitted transactions.

The severity of this attack scenario depends significantly on how transactions are submitted to relays:

- **Public Relay Submission:** If users submit encrypted transactions to relays via the public mempool, then builders naturally gain access to these transactions from the relay's commitment as soon as the decryption key released. In such cases, simulating and front-running relay transactions does not involve any breach of trust or protocol violation. The attack remains feasible and may even be expected under adversarial assumptions.

- **Private Relay Submission:** If transactions are submitted to relays via private channels or private mempools (e.g. Flashbots [11]), the situation becomes more subtle. Ideally, private submission should conceal transaction content from anyone except the relay. However, relays may collude with builders by leaking the contents of their private transaction flow, allowing the builder to strategically position transactions in a way that front-runs the very users they are meant to serve. Although this behavior may not technically violate any formal duty of the relay in current designs, it can also lead to front-run it's own committed transactions.

The use of private relays thus introduces a tension between privacy and trust: while it can prevent public frontrunning by hiding transaction content, it also places implicit trust in the relay not to leak information. Relays might also censor or exclude some transactions from committing those, which will be difficult to verify. Consequently, the reliance on relays as honest intermediaries in the transaction flow introduces a centralization and trust vector that could be abused under adversarial conditions.

3.2.3 Proposer-Controlled Ordering Attack

Despite the confidentiality guarantees provided by Shutter through encryption of user transactions, our analysis reveals a fundamental vulnerability: the block proposer maintains full control over the ordering of encrypted transactions within the block. This ordering freedom exists because, although transactions are encrypted when submitted, the protocol does not enforce or verify any canonical order-the encrypted payloads are opaque to others but entirely subject to the proposer's discretion in terms of placement.

This opens up a significant vector for manipulation. A malicious or economically incentivized proposer could choose to:

- Insert encrypted transactions from bribing users at the top of the batch(scenarios like auction, voting etc).

- Reorder user-submitted transactions in ways that optimize for private profit.

- Apply custom sorting heuristics based on off-chain metadata or submission patterns to infer likely transaction types (e.g., swaps or NFT mints), and rearrange accordingly.

Although keypers will eventually release decryption shares for the batch, they operate without access to the contents or original ordering intent. Thus, even after decryption, there is no way to verify whether the order in which transactions were

included reflects a fair or unbiased submission process. This breaks the perceived neutrality of the proposer role and could lead to trust erosion.

Importantly, this attack does not require breaking the encryption or colluding with any keypers. Instead, it leverages the lack of ordering commitments or proofs in the protocol design. As such, this vulnerability exists even if all keypers behave honestly, and even if the encryption scheme itself is secure.

In practical terms, this implies that wealthy or well-connected actors could gain consistent priority access to blockspace, reducing fairness for average users. This is especially concerning in high-MEV environments-such as those involving decentralized exchanges or NFT launches-where even encrypted transactions can be valuable if their placement can be manipulated.

Mitigation: Fair Transaction Ordering

One possible direction to address the proposer's control over transaction ordering is to introduce a consensus mechanism on the ordering of encrypted transactions. Since keypers are aware of the arrival times of transactions, it may be feasible for them to collectively agree on a fair ordering before decryption and execution. A promising foundation for this idea is Aequitas, a suite of fair-ordering consensus protocols proposed by Kelkar et al [24]. These protocols are built on FIFO-based Byzantine agreement, where each participant maintains a directed acyclic graph (DAG) of observed transactions. The final transaction order is then derived from this shared DAG structure. The main limitation of this solution is that it has a lot of communication overhead and quiet infeasible to directly apply this in Shutter architecture.

Another time-based ordering protocol that was implemented by Hedera [2]. To establish a consistent and fair arrival time for each transaction, keyper nodes individually record the time at which they observe the transaction. Since these observations can vary slightly due to network latency or clock discrepancies, the protocol determines the final arrival time by taking the median of all recorded timestamps. This approach ensures robustness against outliers and reflects a consensus-based estimate of when the transaction was received by the network. This protocol is quite simpler and can be implemented in Shutter network.

3.2.4 Front-Running Encrypted Transactions

This attack exploits a structural asymmetry in how Shutter (or similar encrypted mempool mechanisms) integrates with underlying smart contracts. Although it is illustrated here using the Shutter protocol's deployed deposit contract[1], the vulnerability is not protocol-specific. Any smart contract that exhibits similar functionality-i.e., one that accepts and immediately processes unencrypted transactions outside the Shutter pipeline without any conditional check or ordering-can be susceptible to this form of exploitation.

Attack Setup

In the Shutter architecture, target smart contracts (such as DEXs) are designed to only accept transactions that originate from keyper nodes, ensuring that user-submitted transactions undergo encryption and ordering enforcement before execu-

[1] https://github.com/shutter-network/shutter/blob/main/contracts/contracts/DepositContract.sol

tion. Users must encrypt their transaction payloads and broadcast them on-chain. These encrypted transactions are eventually decrypted by the keyper committee, followed by the previous ordering, and published to the mempool as a batch, from which they are then executed.

However, Shutter's deposit contract deviates from this pipeline. It accepts standard (unencrypted) Ethereum transactions and executes them immediately without keyper intervention as well as any other security checks or orderings of this transaction. This design creates a loophole that attackers can exploit.

Attack Strategy

An adversary wishing to execute a front-running attack can proceed as follows:

1. The attacker crafts a swap transaction (e.g., to front-run a large trade) but may not yet possess sufficient tokens to execute it. This transaction is encrypted and published on-chain, appearing indistinguishable from normal Shutter transactions at this stage.

2. A victim subsequently submits a large swap transaction, also encrypted, and broadcasts it through the same Shutter channel.

3. Upon decryption and batch publication by the keypers, both transactions-attacker's and victim's-enter the public mempool in the original ordering determined by encryption time.

4. At this point, the attacker monitors the decrypted mempool and assesses the profitability of front-running the victim's trade. If a profitable arbitrage or front-running opportunity is confirmed, the attacker immediately submits a deposit transaction to the deposit contract to acquire the tokens needed for the earlier encrypted swap.

5. Since the deposit transaction is unencrypted and bypasses the keyper, it executes instantly. Thus, by the time the attacker's encrypted front-running transaction is picked up and executed, the required funds are already available in the attacker's account.

6. If no profitable opportunity exists after decryption, the attacker simply refrains from submitting the deposit transaction. In that case, the earlier encrypted transaction will fail due to insufficient balance, costing only a transaction fee.

Implication and Realistic Context

This attack leverages the asynchronous interaction between encrypted and normal transactions. Moreover, this pattern generalizes to any architecture where some contracts process encrypted transactions and others accept and execute immediately in plaintext forms. In particular, any contract that modifies user balances or positions in real-time without involving the encryption layer or delay by ordering, becomes a potential vector for this conditional strategy.

Such attacks are especially viable in ecosystems where high-volume trades or predictable trading patterns are expected, such as when automated vaults or DAOs execute regular swaps. Attackers may monitor the encrypted mempool for signs of large transactions and act only when lucrative opportunities appear-thus minimizing cost and maximizing selective engagement.

Mitigation

The core vulnerability enabling the Encrypted Front-Running Attack lies in the inconsistent handling of transaction types. While the core Shutter-enabled contracts enforce encrypted transaction submission and batch-based ordering, the deposit contract deviates from this model by accepting and executing plaintext transactions immediately. This discrepancy allows attackers to conditionally react after decryption but before execution, thereby undermining the intended privacy and fairness guarantees of the system.

To address this issue, we propose the following mitigation strategies:

- **Enforce Encrypted Submissions Across All Contracts:** A straightforward and robust mitigation is to enforce that all user-facing contracts, including the deposit contract, only accept transactions that are submitted through the encryption layer. By requiring all interactions-deposits, swaps, transfers-to be encrypted and processed via the same keypers-controlled batching and ordering pipeline, we eliminate the timing gap that attackers exploit. This ensures uniform treatment of all transactions as well as consistent and fair ordering enforced by the protocol.

- **Enforce Execution Ordering Across Encrypted and Plaintext Transactions:** If enforcing encryption on all contracts is impractical-e.g.,contracts that are deployed in Ethereum mainchain and validators there do not handle encrypted transactions-an alternative is to introduce explicit execution ordering rules within the protocol. In this approach, plaintext transactions (like deposits) are still allowed, but their execution is deferred until all encrypted transactions in the same block have been processed. The protocol could implement a priority queue, where encrypted transactions are guaranteed to be executed first.

This ensures that an attacker cannot submit a deposit transaction and have it take effect before their encrypted front-running transaction executes. These solutions need further investigations to check it's feasibility and effectiveness in Shutter like protocols.

Chapter 4

Implementation: Front-Running Encrypted Transactions

4.1 Implementation Framework

To evaluate the feasibility and impact of our proposed attack 3.2.4 on the Shutter protocol, we implemented a simplified local prototype of the attack scenario. The following setup was used to simulate the behavior of key protocol participants and the encrypted transaction lifecycle.

Smart Contracts

We developed and deployed three custom smart contracts to represent the core components of the attack environment:

- A DEX contract to facilitate token swaps and liquidation.
- A token contract to manage ERC-20 compatible assets as well as minting functionality.
- A Coordinator contract that contains deposit function as well as handle encrypted and decrypted transactions.

Entities

The system involves three key roles:

- **Victim:** A user submitting encrypted large amount swap transaction to the DEX.
- **Attacker:** An adversary who strategically submits encrypted swap transaction and conditionally funds them after observing profitable opportunity.
- **Keyper:** A simplified decryption authority responsible for decrypting encrypted transactions and posting them to the mempool.

Note: While the real Shutter protocol employs a threshold network of keyper nodes to ensure robustness and decentralization, our prototype uses a single keyper node to simplify the simulation and execution flow.

Blockchain Environment

The attack was tested on a local development blockchain using:

- Hardhat[1] for smart contract development, deployment, and testing.
- Ganache-CLI[2] to simulate a lightweight and controllable local blockchain network.

Encryption Scheme

We used the Elliptic Curve Integrated Encryption Scheme (ECIES) for our encryption scheme which is available in `eccrypto`[3] library. This scheme is based on: Elliptic Curve Diffie-Hellman (ECDH) for key agreement and AES-256 in CBC mode for symmetric encryption of transaction payloads.

Implementation Details

- Smart contracts were written in Solidity and compiled with version `0.8.28`.
- Local blockchain was running on `http://127.0.0.1:8545`.
- Client-side and off-chain scripts, including keyper logic and attacker behavior, were implemented in JavaScript.
- The complete codebase is available on GitHub [21]

4.2 Smart Contracts

4.2.1 Token Contract

The `Token.sol` contract is a standard ERC-20 token implementation, primarily used to simulate the asset traded on the DEX and transferred through the deposit contract. It is built using OpenZeppelin's audited ERC20[4] library, ensuring compatibility with widely adopted token standards.

Key Features

- **Initial Supply:** Upon deployment, the contract mints 1 million tokens (with 18 decimals) to the deployer address. This provides the deployer with sufficient liquidity to conduct swaps and simulate attacks.
- **Custom Minting Functionality:** In addition to the standard ERC-20 functions, the contract includes a custom `mint()` function that allows arbitrary minting of new tokens to any specified address. Although in production, unrestricted minitng should not be allowed, but for the purposes of controlled experimentation in a local testnet, this design choice provides flexibility for simulating various scenarios.
- **Use in Attack Scenario:** This token is used as the swapping asset on the DEX, as well as the deposit target in the attack setup.

[1] https://hardhat.org/
[2] https://github.com/trufflesuite/ganache
[3] https://github.com/bitchan/eccrypto
[4] https://docs.openzeppelin.com/contracts/4.x/erc20

4.2.2 DEX Contract

The `Dex.sol` contract implements a simple constant-product automated market maker (AMM) supporting two tokens (`tokenA` and `tokenB`).

Key Features

- **Token Pairing:** Supports swapping between two predefined ERC-20 tokens.
- **Liquidity Pool:** Maintains reserves (`reserveA`, `reserveB`) for both tokens, which are updated after each swap or liquidity addition.
- **Swap Function:** Allows users to swap one token for another, applying a 0.3% fee. The output amount is calculated using the constant product formula used in UniswapV2[5].
- **Liquidity Addition:** Users can contribute liquidity to the pool by depositing both tokens proportionally.

4.2.3 Coordinator Contract

The `Coordinator.sol` contract plays a central role in coordinating encrypted transaction submission, decryption, and token deposit. It facilitates user interactions with the keyper node and includes the deposit functionality that was later exploited in the attack.

Key Features

- **Public Mempool:** The contract maintains two separate arrays: one for encrypted transactions and another for decrypted transactions. Users submit encrypted payloads through `submitEncryptedTx()`, which are later decrypted by the keyper and submitted via `submitDecryptedTx()`. These decrypted transactions include the target contract address and the decoded payload, which can then be executed in a controlled order.
- **Deposit Function:** The contract contains a `deposit()` function that allows users to deposit ETH and receive tokens in return (at a fixed rate of 1 ETH = 100 tokens). This function does not rely on encryption and executes immediately, which was later exploited in our described front-running attack.

4.3 Attack Implementation Workflow

To demonstrate the attack in practice, we developed a series of JavaScript scripts simulating each role in the system. These scripts collectively replicate the full attack sequence, without repeating the already discussed attack logic. The execution order reflects the real-time interaction among entities:

1. `deployer.js`: Deploys the necessary smart contracts, `Token.sol`, `Dex.sol`, and `Coordinator.sol`, and performs initial configuration such as token minting and liquidity provisioning.

[5] https://blog.uniswap.org/uniswap-v2

2. `keygeneration.js`: Executed by the keyper, this script generates the required ECDH keys used for encrypting and decrypting transactions.

3. `attacker.js`: The attacker submits a front-running encrypted swap transaction to the coordinator without sufficient token balance at the time of submission.

4. `victim.js`: Simulates the victim's large encrypted swap transaction, which creates a profitable opportunity for the attacker and submit to the coordinator.

5. `keyper.js`: The keyper decrypts the batch of encrypted transactions and submits the decrypted payloads to the coordinator.

6. `deposit.js`: If a profitable opportunity is detected post-decryption, the attacker uses this script to immediately deposit ETH and mint the required tokens, bypassing the encrypted pipeline.

7. `execute.js`: Finally, this script triggers the execution of the previously submitted decrypted transactions, finalizing the attack.

This sequential execution mimics a real-world scenario on a local blockchain using Hardhat and Ganache, demonstrating how the vulnerability can be exploited in practice.

4.4 Results

To evaluate the severity and impact of the proposed attack, we vary both the swap amounts and the pool liquidity. Specifically, we consider two pools with different pool balances and both follows AMM multiplicative rule to compute fees.

For each liquidity pool, we simulate multiple different swap amounts for both the attacker and the victim with real time datas. These variations help us analyze how the effectiveness and profitability of the attack change with respect to transaction size and liquidity depth. By comparing outcomes with and without the attack, we aim to understand how such manipulation affects both parties under different scenarios.

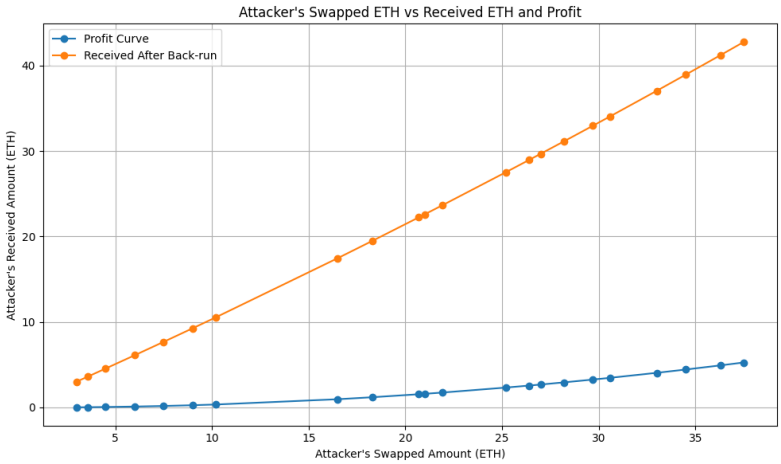
- **Pool 1: ETH/USDC** We consider a Uniswap ETH/USDC pool[6] with reserves of approximately 1,700 ETH and 5,000,000 USDC, as observed on July 11, 2025. Using these real-world reserve values, we simulated our sandwiching attack across various swap amounts. The outcomes of these simulations are illustrated in the graphs below.

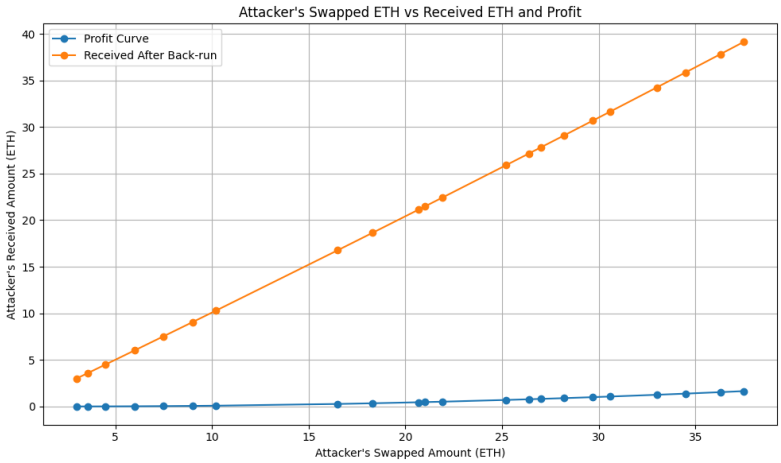
- **Pool 2: ETH/USDT** We alos analyzed a Uniswap ETH/USDT pool[7] with reserves of approximately 5,000 ETH and 151,000,000 USDT, based on data observed on July 11, 2025. Leveraging these real-world reserve values, we conducted simulations of the sandwich attack across a range of swap amounts. The results of these simulations are presented in the graphs below.

[6] https://app.uniswap.org/explore/pools/base/0x88A43bbDF9D098eEC7bCEda4e2494615dfD9bB9C
[7] https://app.uniswap.org/explore/pools/ethereum/0x0d4a11d5EEaaC28EC3F61d100daF4d40471f1852

For each liquidity pool, we have included two sets of graphs. The first set(described above) illustrates the relationship between the victim's swapped amount and the amount received, both in the presence and absence of a sandwich attack. This clearly highlights the loss incurred by the victim due to front-running.

The second set of graphs(described below) focuses on the attacker's perspective. First graph is for pool 1 and second is for pool 2. One curve represents the profit gained from the sandwich attack with respect to the attacker's initial swap amount. The other shows the amount of ETH the attacker receives back after executing the back-running transaction. These visualizations provide insight into both the victim's loss and the attacker's gain under different trading conditions.





From our experiment, we can observe: If the swapping amount increased, the severity of the attack will also increase. Also, as the liquidity pool size increases, the relative loss incurred by the victim decreases. This indicates that larger pools are more resilient to slippage and front-running attacks, as individual swaps have a smaller impact on the token price.

Chapter 5

Conclusion and Future Work

5.1 Summary of Contributions

In this thesis, we conducted an in-depth analysis of the Shutter protocol, with a particular focus on its use of threshold encryption to obscure transaction contents during the mempool phase. While this design offers a promising direction for mitigating front-running and other mempool-level attacks, our analysis reveals several critical vulnerabilities and potential attack vectors.

We examined issues related to transaction ordering, execution of encrypted transactions, all of which can be exploited to compromise fairness and security guarantees. To demonstrate the practicality of these threats, we implemented one such attack in a local blockchain environment, showing how an adversary can front-run encrypted transactions through strategic use of deposit contracts that accepts and immediately executes deposit transactions. This attack can be applied to all of those contracts that show same functionalities irrespective of the platforms.

In addition to identifying weaknesses in the current Shutter deployment, we reviewed the recently proposed encrypted Ethereum mainchain design. Our study highlights further design limitations and security gaps, particularly regarding block builders and block proposers behavior.

Finally, we proposed some mitigation strategies, including enhanced slashing logic with fault tolerance, fair-ordering of encrypted transactions, and contract-level modifications that enforce encrypted execution pathways. Together, these insights contribute to a more robust understanding of secure transaction sequencing and help guide future improvements in encrypted mempool designs.

5.2 Future Research Directions

Building on our findings, several directions for future research and improvement of the Shutter protocol are suggested:

- We propose the design and integration of a fair ordering protocol executed by keyper nodes to determine the final ordering of encrypted transactions as a solution to proposer-controlled ordering. While one protocol, Hedera [2] uses a median-based approach, future research could explore alternative metrics or consensus mechanisms that provide stronger guarantees of fairness and efficiency.

- To mitigate front-running vulnerabilities arising from the asynchronous execution of normal (unencrypted) and encrypted transactions, we recommend

modifying smart contract behavior. Contracts should either:

  - Accept only encrypted transactions, or
  - Enforce a joint execution schedule where unencrypted transactions are executed in sync with or after the decrypted batch, preserving fair ordering.

- Users using encrypted mempool need to trust Layer 2 validators for key management and decryption, which reduces the overall security of the blockchain. To address this, we need to design a mechanism that better integrates encrypted mempool in layer 1 by sampling Layer 2 validators from the Layer 1 validator set and enforcing a slashing mechanism on those validators.

These directions aim to strengthen the guarantees of privacy and fairness promised by encrypted mempool systems, and provide a stepping stone for secure deployment in adversarial environments.

Bibliography

[1] Aave Protocol. Aave: Open source liquidity protocol. https://aave.com/, 2024. Accessed: 2025-05-15.

[2] Leemon Baird, A. Luykx, and P. Madsen. Hedera technical insights: Fair timestamping and fair ordering of transactions. https://hedera.com/blog/fair-timestamping-and-fair-ordering-of-transactions, 2020.

[3] Massimo Bartoletti and Roberto Zunino. A theoretical basis for blockchain extractable value. *CoRR*, abs/2302.02154, 2023.

[4] Bolt protocol. Bolt documentation, proposer commitments to accelerate ethereum. https://docs.boltprotocol.xyz/, 2025. Accessed: 2025-06-01.

[5] Dan Boneh and Matthew Franklin. Identity-based encryption from the Weil pairing. In *Annual International Cryptology Conference*, pages 213–229. Springer, 2001.

[6] Vanesa Daza Bruno Mazorra, Michael Reynolds. Price of MEV: Towards a Game Theoretical Approach to MEV. In *DeFi'22: Proceedings of the 2022 ACM CCS Workshop on Decentralized Finance and Security*, pages 15 – 22. ACM, 2022.

[7] Miles Carlsten, Harry Kalodner, S. Matthew Weinberg, and Arvind Narayanan. On the instability of Bitcoin without the block reward. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, pages 154–167. ACM, 2016.

[8] Arka Rai Choudhuri, Sanjam Garg, Jose Piet, and Ravi Policharla. Mempool privacy via batched threshold encryption: Attacks and defenses. *USENIX Security Symposium*, 2024.

[9] Arka Rai Choudhuri, Sanjam Garg, Chen-Da Wang, and Ravi Policharla. Practical mempool privacy via one-time setup batched threshold encryption. *USENIX Security Symposium*, 2024.

[10] Victor Costan and Srinivas Devadas. Intel SGX Explained. *ePrint Archive*, 2016.

[11] Philip Daian, Steven Goldfeder, Tyler Kell, Yiwen Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash Boys 2.0: Frontrunning in Decentralized Exchanges, Miner Extractable Value, and Consensus Instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[12] DeFiLlama. Defillama: Chains overview. https://defillama.com/chains, 2024. Accessed: 2025-05-21.

[13] Stefan Dziembowski, Sebastian Faust, and Jannik Luhn. Shutter network: Private transactions from threshold cryptography. Cryptology ePrint Archive, Paper 2024/1981, 2024.

[14] Ethereum Foundation. Proposer-Builder Separation (PBS), 2024. Available at https://ethereum.org/en/roadmap/pbs/.

[15] Paul Feldman. A practical scheme for non-interactive verifiable secret sharing. In 28th Annual Symposium on Foundations of Computer Science, Los Angeles, California, 1987.

[16] Flashbots. Flashbots transparency dashboard, 2024. Available at https://transparency.flashbots.net/.

[17] Flashbots. MEV-Boost: A PBS Implementation, 2024. Accessed: 2025-05-15.

[18] Forbes. The Evolution Of Ethereum MEV. https://www.forbes.com/sites/amorsexton/2024/08/15/the-evolution-of-ethereum-mev/, 2024.

[19] Ethereum Foundation. Maximum extractable value. https://ethereum.org/en/developers/docs/mev/, 2025.

[20] Frederik Luhrs, Luis Bezzenberger, Francesco Mosterts, Sebastian Faust, Andreas Erwig. The Road Towards a Distributed Encrypted Mempool on Ethereum. *ethresearch archieve*, 2025.

[21] Front-Running Encrypted Transactions - Github Link. https://github.com/PrabalDas04/Encrypted-Front-Running-Attack/tree/main.

[22] Lioba Heimbach, Lucianna Kiffer, Christof Ferreira Torres, and Roger Wattenhofer. Ethereum's proposer-builder separation: Promises and realities. *arXiv preprint arXiv:2305.19037*, 2023.

[23] Aniket Kate, Yizhou Huang, and Ian Goldberg. Distributed key generation in the wild. Cryptology ePrint Archive, Paper 2012/377, a preliminary version of this paper appeared at IEEE ICDCS '09., 2012.

[24] Mahimna Kelkar, Michael Reiter, Steven Goldfeder, and Ari Juels. Order-Fairness for Byzantine Consensus. *Advances in Cryptology – CRYPTO*, 2020.

[25] Bohan Zhang Peyman Momeni, Sergey Gorbunov. Fairblock: Preventing blockchain front-running with minimal overheads. *EAI SecureComm 2022 - 18th EAI International Conference on Security and Privacy in Communication Networks*, 2022.

[26] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest? *arXiv preprint arXiv:2101.05511*, 2021.

[27] Adi Shamir. How to share a secret. Communications of the ACM, 1979.

[28] Uniswap Labs. Uniswap protocol overview. https://uniswap.org/, 2024.

[29] Wikipedia contributors. Herfindahl–Hirschman Index - Wikipedia, the free encyclopedia, 2024. Accessed: 2025-05-21.

[30] Wikipedia contributors. Trusted Execution Environment - Wikipedia, the free encyclopedia, 2024. Accessed: 2025-05-21.