Rethinking Execution Layer Front-Running Protection with Threshold Encryption

F3B: A Per-Transaction Front-Running Protection Architecture with Low-Latency Overhead

ZIYAN QU
Rethinking Execution Layer Front-Running Protection with Threshold Encryption

F3B: A Per-Transaction Front-Running Protection Architecture with Low-Latency Overhead

ZIYAN QU

Master’s Programme, Communication Systems, 120 credits
Date: August 8, 2023

Supervisors: Bryan Ford, Douglas Wikström
Examiner: Johan Håstad
School of Engineering Sciences
Host organization: EPFL
Swedish title: Omprövning Utförande Lager Front-Running Skydd med Tröskel Kryptering
Swedish subtitle: F3B: En front-running skyddsarkitektur per transaktion med låg latens overhead
Abstract

Blockchain is a decentralized and immutable append-only ledger. Smart contracts, the self-executing programs on blockchain, help build the Decentralized Finance (DeFi) markets. Front-running is the practice of benefiting from advanced knowledge of pending transactions. It impairs the fairness of DeFi ecosystem, leading to huge losses of honest participants. In this thesis, We present Flash Freezing Flash Boys (F3B), a blockchain framework that mitigates front-running with threshold cryptography. In F3B, transactions are encrypted with symmetric keys, which are collaboratively kept by a decentralized secret-management committee (SMC). Once the transactions are committed and immutable, the keys are reconstructed to execute the transactions. F3B hides the content of pending transactions so that adversaries cannot acquire information about them, thus mitigating front-running. Previous work using threshold encryption mitigates front-running with per-block encryption, which would fail when a transaction is not included in the expected future block. F3B solves this issue by adopting per-transaction encryption, ensuring that any uncommitted transaction remains encrypted and private, even when a huge network delay occurs. F3B is an execution layer front-running solution, meaning that it is independent of the consensus algorithms and compatible with existing blockchain networks and smart contracts. F3B is evaluated on a simulated Ethereum network, and proved to be a practical low-latency solution. F3B presents a negligible (0.026%) latency overhead with 128 SMC members, compared with the unprotected blockchain.

Keywords

Blockchain, Front Running, Threshold Encryption
Sammanfattning


Nyckelord

Blockchain, Front Running, Tröskelkryptoering
iv | Sammanfattning
Acknowledgments

The degree project is my first research project in blockchain and the most important experience that leads me to my future research journey. I would like to first express my deepest gratitude to Haoqian, the proposer of F3B and my student supervisor. His ideas and comments are really inspiring, and only with his support and guidance can I complete this project. Many thanks to Bryan and all the labmates at DEDIS, I very much enjoy my time here and appreciate all the discussions we had during my presentations. Again, I’d like to thank Haoqian for authorizing me the permission to develop my thesis from our F3B paper, which started and iterated before I joined the project. It is my honor to work on the F3B project, improving it to a better shape.

I sincerely appreciate the help from Johan and Douglas, my examiner and supervisor at KTH. You provided much support during my project, and offered previous suggestions from the cryptography perspective. Especially, I want to thank Douglas for our meeting, which further determines my pursuit of a PhD degree in applied cryptography.

This period of time is challenging, and I’m grateful for my family and friends who supported me and encouraged me. I wouldn’t achieve this alone without you. I want to thank the most special person, Zhihan. You are the most precious treasure I found in Lausanne. You accompany me all the way through this bittersweet journey, supporting me to be stronger.

I appreciate everything I learned from KTH and EPFL. It is not only knowledge but also how to recognize myself and achieve personal goals. The two years I spent here made me who I am today. I appreciate all the kindness and help from the professors I met during my master’s study, especially Emil and Yang who supported me through my PhD applications.

Finishing my master’s study is not the end, but the beginning of another wonderful journey. Doing research is always full of challenges, and I’m looking forward to that.

Lausanne, Switzerland, August 2023
Ziyan Qu
Acknowledgments
# Contents

1 Introduction 1
   1.1 Background ............................................ 1
   1.2 Problem .............................................. 3
   1.3 F3B system overview ................................... 4
   1.4 Contribution .......................................... 6
   1.5 Research Methodology .................................. 6
   1.6 Delimitations .......................................... 7
   1.7 Structure of the thesis ................................ 7

2 Background 9
   2.1 Blockchain ........................................... 9
   2.2 Smart Contracts ...................................... 11
   2.3 Front-Running & Solutions ............................ 11
   2.4 Threshold Encryption .................................. 13
   2.5 Related Work .......................................... 14
   2.6 Strawman Protocols .................................... 16
      2.6.1 Strawman I: Sender Commit-and-Reveal .......... 16
      2.6.2 Strawman II: The Trusted Custodian ............. 17
      2.6.3 Strawman III: Threshold Encryption with per Block Encryption ......................... 17
   2.7 Summary .............................................. 18

3 Methods 19
   3.1 System Overview ...................................... 19
      3.1.1 System Goals .................................... 19
      3.1.2 Architecture Overview ............................ 20
      3.1.3 System and Network Model ......................... 20
      3.1.4 Threat Model ...................................... 21
   3.2 F3B Protocol .......................................... 21
List of Figures

1.1 F3B architecture. Senders first publish transactions to the consensus group. Once the transactions are immutable, the secret-management committee releases the decryption shares. The SMC and the consensus group can consist of the same set of nodes. For clarity in the thesis, we logically separate them into two different entities. ........................................ 4

3.1 F3B per-transaction protocol steps: (1) Send an encrypted transaction to the underlying blockchain, (2) Prepare shares by trustees while waiting for transaction commitment, (3) Reconstruct the key, (4) Execute the transaction. .............. 22

3.2 In Ethereum, once they are inserted in the blockchain, the transactions are executed and committed after receiving $m$ block confirmations. Whereas, in F3B, transactions are encrypted, and their executions are postponed after receiving $m$ block confirmations when the secret-management committee releases the encryption keys. Both scenarios have a similar commitment time. ................................. 25

4.1 The breakdown latency of each step in F3B by varying the number of secret-management committee trustees from 8 to 128 nodes. ................................................................. 38

4.2 A comparison of the sender commit-and-reveal approach latency with F3B against a baseline modeled in Ethereum. The string “F3B-$X$” represents $X$ trustees. ................................. 39

4.3 Overall performance of the two protocols ....................... 42

4.4 The latency cost of DKG setup and three resharing scenarios. 43

6.1 The traditional blind auction contract with commit-and-reveal. 52

6.2 The new proposed blind auction contract with F3B. ............ 53
List of Tables

4.1 Latency Overhead for Ethereum Blockchain . . . . . . . 40
4.2 Size of ciphertext for two protocols with different Secret-management Committee sizes. . . . . . . . . . . . . . . . . 44
List of Tables
<table>
<thead>
<tr>
<th>Acronym</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>DApp</td>
<td>Decentralized Application</td>
</tr>
<tr>
<td>DeFi</td>
<td>Decentralized Finance</td>
</tr>
<tr>
<td>DEX</td>
<td>Decentralized Exchange</td>
</tr>
<tr>
<td>DKG</td>
<td>Distributed Key Generation</td>
</tr>
<tr>
<td>MEV</td>
<td>Maximal Extractable Value</td>
</tr>
<tr>
<td>PVSS</td>
<td>Public Verifiable Secret Sharing</td>
</tr>
<tr>
<td>VSS</td>
<td>Verifiable Secret Sharing</td>
</tr>
</tbody>
</table>

xiv | List of acronyms and abbreviations
Chapter 1

Introduction

1.1 Background

In recent years, blockchain technology has emerged as a transformative force, revolutionizing various industries and enabling novel decentralized applications. Blockchain is a decentralized, distributed, and immutable ledger. Blockchain is used to securely record and verify transactions across the distributed computer network using cryptographic primitives. Initially introduced as the underlying technology for cryptocurrencies like Bitcoin [1], blockchain is now extended to various applications beyond finance.

One of the key components driving the adoption of blockchain is the concept of smart contracts, which are self-executing computer programs modeled after contracts or agreements. When certain conditions are met, the smart contracts automatically execute predefined operations. Smart contracts suit well for blockchain networks, which are decentralized and tamper-resistant. Since the launch of the Ethereum blockchain, smart contracts thrive on distributed ledgers for building all kinds of Decentralized Applications (DApps). Smart contracts enable the automation of trustless transactions without the need for intermediaries, and have the potential to revolutionize traditional business processes by eliminating the need for trust in transactions and reducing associated costs.

However, as blockchain gains popularity and its applications expand, new challenges and vulnerabilities arise. Front-running is the practice where actors gain advantages from their knowledge of pending transactions in decentralized systems [2, 3, 4]. This type of attack exploits the transparent nature of blockchain transactions, allowing malicious actors to manipulate the order and timing of transactions for personal gain. Front-running attacks not
only compromise the fairness of blockchain networks but also jeopardize user privacy and confidentiality. However, with the pseudo-anonymity, openness, and decentralization offered by blockchain, it is hard to regulate front-running in Decentralized Finance (DeFi) [5, 2, 6]. A front-running actor in blockchain can, for example, read the data of a pending transaction, create new transactions accordingly and place them before or after this target transaction [7, 6, 2].

In parallel, the concept of Maximal Extractable Value (MEV) has gained attention due to its potential for abuse within blockchain systems. MEV is closely related to front-running, which refers to the maximal additional value that miners can extract from the order and inclusion of transactions in blocks [8]. Intuitively, miners should entirely gain MEV, but as the transactions are visible in the whole blockchain network, anyone can conduct front-running with algorithms to search for profitable opportunities and automatically submit specialized transactions. Besides, miners can engage in manipulative behaviors and extracting disproportionate profits. MEV poses a significant risk to the integrity and security of blockchain networks, as it incentivizes miners to engage in malicious activities at the expense of the broader ecosystem.

Front-running brings negative impacts to the DeFi world, as it infringes profits of honest actors and endangers the fairness of the market [9]. It is estimated that front-running results in a monthly loss of $280 million for DeFi actors [10]. Except for financial loss, front-running also threatens the security of the underlying consensus layer for the blockchain by incentivizing unnecessary forks [11, 6].

Many efforts have been made to mitigate or prevent front-running, but these approaches turn out to have drawbacks or limitations. Flashbots introduces a private channel for a transaction to be directly sent to a validator [12]. The validator then adds it to the blockchain, so that malicious actors cannot do front-running. However, actors in the process still have full access to the transaction data, and can easily conduct front-running. Namecoin adopts the commit-and-reveal mechanism to mitigate front-running [13]. Users first send a commit and later reveal the transaction, which needs two rounds of communication with the underlying blockchain. Submarine hides the smart contracts addresses to further improve this design, increasing the rounds of communication to three [14, 13]. There are also works that aim at front-running protection against specific applications or consensus algorithms [15, 16, 17, 18, 19, 20, 21].

A better and more promising approach is to make use of threshold
encryption. Keeping transactions encrypted can prevent actors from reading the content before a specific time. A group of actors will be responsible for decrypting the transactions later following predefined rules, as presented in Fairblock [22] and Shutter [23, 24]. Their protocol is based on distributed key generation and identity-based encryption, and a user is required to choose a future block or epoch where the transaction is expected to be included and get the corresponding encryption key. In this scenario, potential risks exist if a transaction failed to be included in the specific block chosen by the user. That might happen with network congestions [25] or denial-of-service attacks [2]. In this case, the transaction will be revealed undesirably and might cause a huge loss. (see Section 2.6.3 for details).

1.2 Problem

To address these challenges, the Flash Freezing Flash Boys* (F3B) architecture is proposed recently, which is designed to mitigate front-running attacks with low latency overhead. Similar to Fairblock [22] and Shutter [23, 24], F3B leverages the power of threshold cryptography to provide front-running protection, but it is based on a per-transaction protocol rather than the per-block scheme. Instead of using a key that is associated with a future block, every transaction now has a unique symmetric key, which ensures that it will be revealed only after it is properly committed. F3B can be directly used upon existing consensus algorithms and smart contracts, and is evaluated to provide low latency overhead in Ethereum.

In this thesis, we work on the previous prototype of F3B to further improve the F3B architecture, by optimizing the existing protocol and designing a new protocol. Furthermore, this thesis provides a detailed discussion about security and system goals and evaluates the performance of the two protocols with respect to latency, throughput, storage, and others. This project also explores potential applications of the F3B and discusses various properties.

This thesis follows the same architecture as our research publication of the F3B system. While our current version of submission is under review and not public yet, some of the content in this thesis refers directly to the former version of the research paper [26]. The thesis author is the main contributor to this project in the new submission, and is authorized to directly use contents from the paper in this thesis.

*The name Flash Boys comes from a popular book revealing this aggressive market-exploiting strategy on Wall Street in 2014 [5].
Figure 1.1: F3B architecture. Senders first publish transactions to the consensus group. Once the transactions are immutable, the secret-management committee releases the decryption shares. The SMC and the consensus group can consist of the same set of nodes. For clarity in the thesis, we logically separate them into two different entities.

Hence, the research question of this project is, how can we better protect against front-running using the F3B architecture? Can we design different protocols that satisfy different scenarios? What performance can we achieve using F3B on common blockchain systems such as Ethereum? Can we prove F3B to be secure and reliable? Can we utilize the F3B to build other DApp that is simple and secure?

1.3 F3B system overview

As described in Figure 1.1, F3B introduces the secret-management committee (SMC). In general, the architecture of F3B consists of the following steps: (a) A client encrypts their transaction using a unique symmetric key. The key is collaboratively kept by the secret-management committee. The encrypted transaction is sent to the consensus group that operates the underlying blockchain. (b) The SMC reads the encrypted transaction from the underlying blockchain. (c) The SMC prepares the decryption shares for the consensus group. (d) The SMC releases the decryption shares to the consensus group once the underlying blockchain has committed the transaction. (e) The
The consensus group reconstructs the key. (f) The consensus group decrypts and executes the transaction. Once the SMC begins to release the decryption shares after block confirmations, malicious actors cannot launch a front-running attack because all the transactions in that block are already irreversibly ordered and immutable on the blockchain. Although adversaries can still conduct speculative front-running attacks based on side-channel information, we argue that these attacks have a greater possibility of failure, and cannot guarantee promising profit [7].

F3B needs to solve two key challenges to be practical: (a) the spamming of inexecutable junk transactions that are encrypted and written onto the underlying blockchain, and (b) the latency overhead and throughput limit introduced by providing front-running protection. To mitigate spamming, we propose a refundable storage fee for storing encrypted transactions in addition to the standard transaction fee (e.g., gas in Ethereum). The latency overhead is limited by only requiring one transaction compared with previous commit-and-reveal mechanisms. Besides, we allow the SMC to prepare decryption shares when waiting for block confirmations, and enable batch processing to achieve low latency and high throughput.

We propose two cryptographic threshold schemes that can plug into F3B: TDH2[27] and PVSS [28]. TDH2 enables clients to encrypt their transactions under the same public key of a secret-management committee which is only changeable by time-consuming DKG or resharing protocols. On the other hand, PVSS empowers clients to adopt a different secret-management committee for each transaction but at the cost of the additional preprocessing time for preparing the shares for each transaction.

We implemented a prototype of F3B referring to the post-Merge *Ethereum [29] as the underlying blockchain. The core part of F3B, which is the secret-management committee, is implemented on the DEDIS Ledger Architecture (Dela) [30], which is an architecture implemented by our DEDIS lab for blockchain research purposes. Our analysis shows that, with a secret-management committee size of 128 trustees, the latency overhead of F3B is 0.026% and 0.027% in Ethereum under the TDH2 and PVSS respectively; In comparison, Submarine, which also offers per-transaction front-running protection as F3B, presents a 200% latency overhead, because it requires three rounds of communication with the underlying blockchain [14, 31]. The detailed introduction of protocol designs and evaluation will be presented in later chapters.

*The Merge refers to the merge executed on September 15th, 2022, to complete Ethereum’s transition to proof-of-stake consensus.
In our prototype on Ethereum, we modified the execution layer by implementing a new transaction type for the encrypted transaction, and connecting with the Dela which serves as the SMC network for threshold encryption and decryption. Since F3B only requires modifications on the execution layer, it is compatible with various consensus algorithms embedded in Ethereum’s consensus layer, including Proof-of-Work (PoW), Proof-of-Authority (PoA), and the Proof-of-Stake (PoS). At the same time, the F3B can work with existing smart contracts without modifying the codes, and provide front-running protection by easily changing the transaction type.

1.4 Contribution

The main contribution of the degree project can be summarized below:

1. The redesign and improvement of the F3B architecture, mitigating front-running with per-transaction threshold encryption, enabling confidentiality for pending transactions.

2. The design of PVSS-based protocol and optimization of TDH2-based protocol, satisfying different demands and user scenarios.

3. The full implementation of F3B architecture with two different protocols, compatible with various consensus algorithms and existing smart contracts.

4. The systematic evaluation of F3B on post-merge Ethereum with regards to transaction latency, throughput, storage, and reconfiguration costs. Proving that the F3B achieves low-latency overhead.

1.5 Research Methodology

The degree project is carried out using a mixed methodology. The protocols of the F3B will be designed with existing cryptography primitives. We analyze the incentives, security, and system goals by qualitative analysis. The system is then implemented on the Dedis Ledger Architecture [30]. We test the latency, throughput, and other metrics of the system under different settings, and conduct quantitative analysis to evaluate the performance.
1.6 Delimitations

Due to the time and scope of this project, the main focus will be designing and implementing the framework. Experiments and evaluations on real blockchain networks and nodes will not be made. Instead, all results will be obtained in a simulated environment. Because of the limitation of computing resources, the number of nodes and batches is reasonably limited and sufficient to reach our conclusion. Due to the lack of codes and detailed design, our methods will not be compared with previous work using similar threshold encryption schemes. Considering the master’s thesis has a limited time, we leave detailed design and discussion of further applications of our architecture to future work and only provide prototypes in this project.

1.7 Structure of the thesis

The thesis consists of 6 chapters.

- Chapter 1 is the introduction, where a rough background is given, followed by the problem definition, system overview and others.

- Chapter 2 is the background. In this chapter, a detailed introduction to all necessary background knowledge is given, and related works are introduced. To help better understand the motivation and novelty, some strawman protocols are given based on the related works.

- Chapter 3 is the methods, which introduces the system and protocol design thoroughly. Some analysis and system goals are presented and discussed.

- Chapter 4 is the results and analysis, where further evaluation and discussion are given. Specifically, we present evaluation results about latency, throughput, reconfiguration, and ciphertext storage.

- Chapter 5 is the discussion. In this chapter, more discussion about deployment challenges and assumptions for optimization are given.

- Chapter 6 is the conclusions and future work, which presents our conclusion, limitations and some future work.
8 | Introduction
Chapter 2

Background

In this chapter, a detailed background of this thesis is presented, including blockchain and smart contracts, front running and MEV, various mitigations, and threshold encryptions.

2.1 Blockchain

A blockchain is a decentralized and immutable append-only ledger that securely records ordered transactions using cryptographic primitives [1]. It provides a transparent and tamper-resistant platform for trustless interactions, eliminating the need for intermediaries and enhancing data integrity.

Blockchain is operated and maintained through a decentralized network of nodes, each maintaining a copy of the entire blockchain, ensuring redundancy and resilience. Transactions in the blockchain go through several states before they are irreversibly ordered and recorded on the blockchain. Transactions, created by the senders, will first be broadcasted to the network and verified. Then transactions are grouped into blocks, which are subsequently added to the existing chain in sequential order. Before a block can be appended to the blockchain, it must be validated through a consensus mechanism, which determines the order of transactions within a block. The transactions in a block and their order should be consistent among all nodes in the blockchain, and such agreement is met through various consensus algorithms.

The well-known implementation of blockchain, Bitcoin [1], uses the Proof-of-Work(PoW) consensus algorithm. In PoW, miners(nodes in the blockchain) compete to solve computationally intensive puzzles, and the first miner who finds a solution is authorized to add a new block to the chain. Proof-of-Stake(PoS) algorithm, which is used by post-merge Ethereum [32], selects
validators proportional to their stake to be responsible for creating new blocks. In this way, the high computational cost of PoW is avoided. Normally, miners or validators get rewards from adding a block to the chain. Besides, for every transaction included in the block, the miner or validator receives a transaction fee (gas fee, priority fee, ...).

Except for consensus algorithms, blockchain networks can be categorized based on the level of access and control over participation, the two different categories are permissioned and permissionless blockchains. In permissioned blockchains, such as Quorum[33], Corda[34], and Ripple[35], the participation of nodes is restricted, and authorization is needed to join the blockchain. The permissioned blockchain is thus not publicly accessible and transparent, and always being partially decentralized or fully centralized. By only allowing trusted entities in the network, permissioned blockchains always achieve a higher level of privacy and scalability, as well as higher throughput. In permissionless blockchains such as Bitcoin and Ethereum, on the contrary, anyone can join the network. The permissionless blockchains are fully decentralized and always trustless, meaning that a node can validate transactions and run consensus algorithms without trusting other nodes. The openness of permissionless blockchains enables global participation and fosters censorship-resistant systems.

Regardless of the consensus algorithms and categories of the blockchains, pending transactions suffer from front-running attacks that manipulate the order of transactions. Furthermore, in some probabilistic consensus algorithms such as PoW or PoS, a transaction included in a validated block might still be reordered by inducing a fork of the underlying blockchain. To make sure that block data and transaction orders are immutable under such consensus algorithms, enough block confirmations are needed to confirm the inclusion and immutability of the block. Block confirmation refers to the process where nodes in the blockchain collectively agree on the validity of a block. The number of confirmations a block receives indicates the level of security and finality associated with it [1, 36, 37]. As more subsequent blocks are added on top of a particular block, the probability of a malicious actor reversing or altering that block decreases significantly. Therefore, a higher number of confirmations provides greater confidence in the immutability and validity of the block and the transactions it contains.
2.2 Smart Contracts

Smart contracts are self-executing programs modeled after agreements or contracts that are predefined. They have the potential to revolutionize traditional business processes by eliminating the need for intermediaries, facilitating trustless transactions, and enhancing transaction efficiency. Smart contracts are naturally suitable for decentralized fault-tolerant consensus algorithms like PoW, PoS, and PBFT-style algorithms [38, 1, 39], as they help ensure the execution and integrity of the smart contracts.

Once deployed and executed on the blockchain, smart contracts become immutable. This makes smart contracts resistant to tampering and fraud, and ensures that the agreements cannot be modified unilaterally. Various blockchain platforms provide environments for developing and executing smart contracts. Ethereum is one of the most well-known and commonly-used platforms for smart contracts. With its Ethereum Virtual Machine (EVM), Ethereum allows developers to create complex DApps using its Turing-complete programming language, Solidity. Other platforms such as EOS, NEO, and Cardano also support the development and execution of smart contracts.

Financial DApps offer DeFi services. For example, Decentralized Exchanges (DEXes) leverage smart contracts to enable peer-to-peer trading of digital assets without relying on a centralized authority. Other DApps build upon smart contracts to achieve various purposes, such as gaming, governance and decision-making, and supply chain management. However, while smart contracts are expressive, trustless and automatic, significant risks exist. The transparent nature of blockchain creates vulnerabilities that can be exploited by front-running.

2.3 Front-Running & Solutions

Front-running is the practice where actors gain an unfair advantage from their knowledge of pending transactions [2, 3, 4]. In the context of blockchain and DeFi, front-running is achieved by manipulating the order and timing of transactions. Commonly, only one miner will be granted the privilege to determine the order of transactions to be included in every block, and it is easy to have incentives that will influence the behavior of a miner and benefit from the change of transaction order, for example, profiting from price discrepancies or gaining other advantages over the market.
MEV refers to the additional value that a miner or validator can extract from the operations during block inclusion. For example, DEX arbitrage is a simple and most-known chance of MEV. If two DEXes have different prices for a token, someone can buy tokens from the lower-price DEX and sell them to the higher-price DEX in one single transaction. It is expected for the miner or validator to get MEV. However, due to the transparency of blockchain, everyone can read the pending transactions in the mempool, which gives front-running opportunities. Many searchers run well-designed algorithms to detect profitable MEV opportunities and automatically submit corresponding transactions [40]. For some other searchers, generalized frontrunners are preferred. When detecting a profitable transaction, the frontrunner will craft the same transaction with the address of this frontrunner, verify locally that modifying this transaction results in a profit to the frontrunner, and submit the modified transaction with a higher transaction fee or gas fee. The higher fee guarantees that the miner is incentivized to include them in the current block. At the same time, the miners can strategically include transactions and order them, and create their own transactions to make profits.

MEV has both positive and negative effects. As those searchers find the inefficiencies and unfairness, the economic rationality and robustness of DeFi improves. Still, they bring bad experiences to honest users and may result in network congestion. Similarly, front-running is not necessarily considered a kind of ”attack” that should be eliminated. There are different ways of addressing MEV extraction and front-running. Flashbots try to mitigate the negative effects of MEV by enabling everyone in the network to extract MEV [41]. It aims to establish a private channel between transaction senders and validators to negotiate the inclusion and order of transactions, preventing frontrunners from exploiting the knowledge and targeting pending transactions. Flashbots achieve over 80% of MEV extraction and the validators' profits doubled since the launch of flashbots [42].

Despite eliminating the side effects of MEV and preserving fairness, other approaches try to simply prevent front-running. Ideally, an immediate and consistent global ordering of transactions is desired to protect against front-running, but this is practically impossible because of the network clock synchronization [43] and consistency problems. Many research works on fair ordering try to solve this transaction order problem under certain assumptions [44, 45].

A more practical solution is to make transaction content not accessible before the order is determined. The commit-and-reveal mechanism is thus introduced to solve front-running. In such a mechanism, every sender first
submits a commit of the actual transaction, for example, a hash of the transaction, to the blockchain. Miners or validators determine the inclusion and order of transactions without knowing what is actually inside. After these commits are written on the chain, users send another plaintext transaction consistent with their commit, which is used to verify the correctness and execute it according to the commit order. While commit-and-reveal indeed solves front-running, it always requires at least two rounds of transactions, which is really inefficient. A better way that is being studied recently is to use threshold encryption to encrypt the transaction first, and introduce a party for the automatic "reveal" process later. These methods make it impossible to read the content of transactions before the order is fixed to prevent front-running. Detailed protocol designs are introduced in section 2.6.

### 2.4 Threshold Encryption

Threshold cryptosystems protect information by encrypting and distributing it among a group of participants in a fault-tolerant way. Threshold encryption aims to provide secure and distributed access control to sensitive data. In threshold cryptography, multiple participants collectively contribute to the key generation and decryption process.

Secret sharing is the foundation of threshold cryptography. It allows a secret, such as a cryptographic key, to be divided into multiple shares, and these shares individually reveal no information about the secret, but can be used to reconstruct the original secret when combined together. In a \((t, n)\) secret sharing scheme, an honest dealer shares a secret over a set of \(n\) participants, so that at least \(t\) participants must coordinate to reconstruct the secret. **Verifiable Secret Sharing (VSS)**\(^4\) is an improved secret sharing, which enables participants to verify that the shares are consistent even if the dealer is malicious. VSS is used to build many other protocols, including **Distributed Key Generation (DKG)** protocols, threshold signature schemes (TSS), and Multi-Party Computation (MPC) protocols \(^4\). These protocols are further used in various applications and distributed cryptosystems.

There are also other variants of secret sharing, such as **Public Verifiable Secret Sharing (PVSS)** \(^2\). PVSS provides an additional layer of transparency and verifiability by enabling participants to publicly verify the correctness of the shares they receive during the secret sharing process. This verification ensures that no participant has tampered with or manipulated their share, enhancing the overall security of the threshold encryption scheme.

**DKG** further improves secret sharing by establishing the shared secret
collectively. In secret sharing, a dealer is responsible for dividing the secret into shares, while in DKG, a shared private key is securely generated among multiple participants. Each participant contributes their randomness, and collectively generate the private key without any individual having full knowledge of the key. They establish a corresponding collective public key that can be used to encrypt messages or other secrets, and at least a threshold number of participants must collaborate to reconstruct the encrypted secrets. DKG ensures that no single participant can compromise the security of the generated key.

2.5 Related Work

Namecoin is an early example of employing a commit-and-reveal design for front-running protection [13]. It serves as a decentralized name service. The Namecoin protocol operates by having users initially broadcast a salted hash of their desired name. After committing to the hash, users broadcast the actual name. The first strawman protocol (Section 2.6.1) is based on Namecoin.

After Namecoin, Eskandari et al. [2] further explored and systematized front-running attacks within the blockchain ecosystem. Their work categorizes front-running attacks into three different types: displacement, insertion, and suppression. These findings shed light on the various strategies employed by attackers to exploit the vulnerabilities within blockchain systems.

Daian et al. [6] conducted an economic analysis of front-running attacks, highlighting the security risks they pose to the underlying consensus layer. Their research unveiled how front-running can incentivize unnecessary forks, driven by MEV extraction. This further jeopardizes the security of the blockchain network and emphasizes the need for robust measures to mitigate front-running.

Many previous works explore the idea of applying threshold cryptography on blockchain. Virtual ASICs, a virtualized version of PoW, implement a threshold broadcast encryption scheme in the blockchain layer [49]. Many other works try to discuss front-running mitigation with threshold encryption, but they are limited to specific consensus algorithms [50][18][16][21][20]. Fairblock [22] and Shutter [23, 24] are recently developed methods to prevent front-running using identity-based encryption. They encrypt transactions on a per-block basis and decrypt them later. This strategy guarantees that transactions will not be seen before the specific block key is reconstructed. However, if an encrypted transaction fails to be included in the block chosen by the sender, the transaction would be revealed while not being immutable.
The Strawman III design (Section 2.6.3) is based on their approach.

Calypso introduces on-chain secrets, which are achieved by threshold encryption controlled by the secret-management committee [51]. In Calypso, ciphertexts can be stored on the blockchain, and any read or write operations require a key collectively decrypted by the secret-management committee members following a predefined policy. Our F3B is inspired by Calypso, and we adopt a similar strategy for mitigating front-running. F3B further improves and extends the functionality of secret-management committee to automatically release the key shares and decrypt transaction content once committed automatically. F3B specifically uses per-transaction encryption to solve the problem that occurred in Shutter and Fairblock. Now all pending and uncommitted transactions will be protected from front-running, even when transactions are delayed due to congestion or DoS attacks.

Apart from threshold cryptography, other works try various approaches to mitigate front-running. Some studies focus on fair ordering [52, 53, 54], which tries to guarantee that the order of transactions is consistent globally and cannot be easily manipulated by a single point. However, they always require too many assumptions and can hardly prevent an adversary with a quick enough network [7]. Wendy proposes to combine commit-and-reveal with fair ordering [54] but the method lacks quantitative overhead analysis. Submarine extends a commit-and-reveal design to prevent leakage of the smart contract address, focusing on application-layer front-running protection. As discussed before, however, the commit-and-reveal mechanism introduces high latency overhead, and the Submarine requires three rounds of communication between the sender and the underlying blockchain [14, 31].

Some works explore the possibility of time-lock encryption, and introduce time-lock puzzles to blind transactions [55]. For instance, the injective protocol [56] adopts verifiable delay functions [57] and achieves a proof-of-elapsed-time. However, an open challenge in using time-lock puzzles in the blockchain is how to link the parameters to real-world time [7].

In the end, there are works that use trusted execution environment [58] to mitigate front-running, such as MEV-SGX [59], Tessera [60], Secret Network [61], and Fairy [17]. While this guarantees security and privacy, such approaches will suffer from a single point of failure or compromise due to the centralized components [62, 63].
2.6 Strawman Protocols

To explore the inherent challenges involved in constructing a framework like F3B, we will first examine a couple of previous approaches, referred to as "strawman" designs. These approaches were considered promising but ultimately proved to be inadequate, and are representative of state-of-the-art proposals such as LibSubmarine, Shutter, and others [14, 31, 22, 23]. Their methods are abstracted and simplified for expository purposes.

2.6.1 Strawman I: Sender Commit-and-Reveal

The first strawman design is called traditional sender commit-and-reveal protocol. It requires the sender to create two transactions: a *commit* and a *reveal* transaction. The commit transaction serves as a commitment, usually a hash, of the intended reveal transaction. The reveal transaction contains the typical fields and contents of a traditional transaction that is susceptible to front-running. The sender propagates the commit transaction and then *waits* until it is validated by the consensus group, written on the blockchain and being immutable, before releasing the reveal transaction. Once the reveal transaction is propagated, the consensus group verifies and executes the transactions in the same order as the commit transaction’s commitment on the blockchain. With the commitment established in the former transaction, the sender is unable to modify the contents of the reveal transaction.

This straightforward strawman protocol effectively mitigates front-running by determining the execution order based on the commit transaction, while keeping the contents of the reveal transaction hidden. However, this strawman protocol encounters several notable challenges: (a) The sender must remain constantly connected to the blockchain network to monitor when it is appropriate to release their reveal transaction. (b) The reveal transaction may experience delays due to congestion events, such as the cryptokitties storm [25], or an intentional denial-of-service (DoS) attack, like the Fomo3D incident [2]. (c) This approach is susceptible to output bias, as the consensus nodes or the sender themselves may choose not to reveal certain transactions after the commit transaction [7]. (d) this approach incurs significant latency overhead, exceeding 100%, since the sender must now send at least two separate transactions instead of the typical single plaintext transaction.
2.6.2 Strawman II: The Trusted Custodian

A straightforward approach to eliminating the sender’s involvement, subsequent to sending the commit transaction, is to utilize a trusted custodian. Once the consensus group has committed the transaction onto the underlying blockchain, the trusted custodian reveals the transaction’s contents.

Again, this strawman protocol effectively addresses front-running since the nodes are unable to access the transaction’s contents before ordering it. However, relying on a trusted custodian introduces a single point of failure. If the custodian experiences a crash, consensus nodes cannot decrypt and execute the transaction, and the whole blockchain network will be stuck. Additionally, the trusted custodian poses a single point of compromise, as they could engage in covert malicious activities by colluding with front-running actors for personal gain.

To overcome these challenges, we propose employing a decentralized custodian, which mitigates both the single point of failure and compromise concerns, thereby significantly increasing the difficulty of collusion.

2.6.3 Strawman III: Threshold Encryption with per Block Encryption

The next natural idea involves establishing a decentralized committee responsible for generating a public key for each block. This arrangement empowers users to encrypt their transactions for a future block. Subsequently, the committee releases the private key after the block’s commitment. Additionally, the committee can leverage identity-based encryption [64] to enable users to derive a future block key based on the block’s height.

This strawman protocol appears to address front-running concerns since transactions within a block remain encrypted until they are committed to their designated block. And it only requires one transaction, and prevents single point of failure or compromise. However, if an encrypted transaction fails to be included in the specified block, its contents will be revealed shortly after while remaining uncommitted. Consequently, it becomes vulnerable to front-running. This could also lead to severe financial loss just because the transaction is revealed earlier or later than expected. Within the history of blockchain networks, such situations happened due to various reasons. For example, the cryptokitties storm [25] leads to huge congestion simply because of too many players. Deliberate well-funded DoS attacks, such as the Fomo3D attack that flooded the Ethereum network with transactions for
three minutes [2], can also result in the inclusion failure mentioned above. Moreover, this protocol could incentivize a consensus node to deliberately produce an empty block, aiming to expose the pending transactions for that block. Therefore, to ensure that a transaction is never revealed before it is committed on the blockchain, we require a per-transaction level of confidentiality rather than per-block confidentiality. That is how we are motivated to propose the F3B

2.7 Summary

In this section, we provided a thorough background of this thesis. Starting from the introduction of blockchain and smart contracts, the definitions and solutions of front-running are presented. Then we introduce threshold encryption, followed by a systematic related work. After that, we pick three types of protocols to serve as the strawman protocols, in order to explain the iteration of solutions and how our F3B architecture is inspired.
Chapter 3

Methods

In this chapter, we present the methods used in the project. Specifically, we introduce in detail the architecture, system design, and protocols of F3B.

Some of the system goals and designs are directly referred from previous version of the project [26], and are authorized to use.

3.1 System Overview

This section provides an overview of F3B’s system goals, architecture, and models.

3.1.1 System Goals

Our system goals, inspired by the strawman protocols, are as follows:

- **Front-Running Protection**: Prevents entities from conducting front-running.
- **Decentralization**: Mitigates a single point of failure or compromise.
- **Confidentiality**: Ensures that a transaction is only revealed after it has been committed by the underlying consensus layer.
- **Compatibility**: Remains agnostic to the underlying consensus algorithm and smart contract implementation.
- **Low-Latency**: Presents low-latency transaction overhead.
3.1.2 Architecture Overview

F3B, as depicted in Figure 1.1, addresses front-running by collaborating with a secret-management committee responsible for managing the storage and release of on-chain secrets, which is the symmetric key used for encrypting transactions. Instead of propagating their transactions in cleartext, the sender can now encrypt their transactions and store the corresponding secret keys with the secret-management committee. Once the transaction is committed, the secret-management committee releases the secret keys, allowing consensus nodes of the underlying blockchain to verify and execute transactions. Overall, F3B achieves state machine replication on the underlying blockchain through two steps: transaction ordering and transaction execution. By ensuring that the majority of trustees within the secret-management committee are secure and honest and by revealing the key to the consensus group at the appropriate time, each consensus node can consistently maintain the same blockchain state.

It is worth noting that F3B encrypts the entire transaction* instead of only the data field. Because other information, such as the smart contract address, can also provide enough information to launch a probabilistic front-running like the Fomo3D attack [2], or a speculative front-running attack [7].

3.1.3 System and Network Model

The architecture of F3B comprises three main components:

- **Senders**: Users in the blockchain that publish (encrypted) transactions.
- **Secret-management committee (SMC)**: The party engaging in threshold encryption and decryption. It manages and releases secrets.
- **Consensus Group**: The group that will commit, validate and execute transactions. It maintains the underlying blockchain.

While the secret-management committee can consist of the subset of nodes in the consensus group of the blockchain or come from another separate network, in this paper, we conceptually separate them into two distinct entities for clarity.

For the F3B based on the PVSS scheme, the client can choose a different SMC for each transaction. In contrast, for the F3B based on the THD2 scheme, an SMC has a fixed membership during one epoch. However, when transiting from one epoch to the next, the SMC can modify its membership and provide

---

*Section 5.4 further discusses how to hide the sender’s address.*
backward secrecy without influencing the ongoing encryption by running a resharing protocol [65].

For the underlying network, we assume that all honest blockchain nodes and trustees of the SMC are well connected, and their communication channels are synchronous. This means that if an honest node or trustee broadcasts a message, all honest nodes and trustees receive the message within a known maximum delay [66].

3.1.4 Threat Model

In our threat model, we assume that the adversary is computationally bounded and that the cryptographic primitives used in the system are secure. Specifically, we assume that the Diffie-Hellman problem and its decisional variant are hard. We further assume that all messages are digitally signed, and both the consensus nodes and the SMC only process correctly signed messages.

The secret management committee consists of $n$ trustees, where $f$ can fail or behave maliciously. We require $n \geq 2f + 1$ and set the secret-recovery threshold to $t = f + 1$. Additionally, we assume the security of the underlying blockchain. For example, in a PBFT-style or PoS blockchain, at most $f'$ out of $3f' + 1$ validators can fail or misbehave, or in a PoW blockchain, the adversary controls less than 50% of the computational power.

We assume that attackers do not launch speculative front-running attacks [7]. However, we discuss mitigation strategies for reducing side-channel leakage in Section 5.4.

3.2 F3B Protocol

In this section, we present the detailed protocol of F3B. We first introduce the necessary preliminaries, and then present the outline and steps of both protocols. Some analysis and comparison of the protocols are given, followed by a more comprehensive protocol description offered in Section 3.3. Later, Chapter 5 introduces some optimizations.

3.2.1 Preliminaries

This subsection introduces the preliminary concepts and components that are fundamental to understanding the F3B protocol. We cover the baseline model for the underlying blockchain and the cryptographic primitives utilized in F3B.
**Figure 3.1:** F3B per-transaction protocol steps: (1) Send an encrypted transaction to the underlying blockchain, (2) Prepare shares by trustees while waiting for transaction commitment, (3) Reconstruct the key, (4) Execute the transaction.

**Blockchain Model** We adopt a blockchain model where transactions are committed into blocks using a consensus protocol. Each block is linked to a previous block, forming a chain. The time of every underlying block is denoted as $L_b$ seconds. In PoW and PoS-based blockchains, a transaction is considered committed only after a certain number of additional blocks, known as block confirmations, have been added to the chain. We define that a transaction is committed after $m$ block confirmations. Hence, the baseline transaction latency is $mL_b$ seconds.

**Shamir’s Secret Sharing** Shamir’s secret sharing is a scheme that enables a dealer to distribute a secret among several trustees. In a secret sharing with an access structure of $(t, n)$, a secret $s$ is distributed among $n$ trustees, such that any group of $t \leq n$ or more trustees can reconstruct the secret $s$, while no group of less than $t$ trustees learns any information about $s$. In the simplest secret sharing, the dealer is always assumed to be honest. Verifiable secret sharing (VSS) enhances this scheme by allowing the trustees to verify the validity of the shares they receive [47]. Public verifiable secret sharing (PVSS) further extends VSS by enabling a third party to verify all the shares [28].

**Distributed Key Generation (DKG)** Distributed key generation is a process in which multiple parties collaboratively generate a private-public key pair.
(sk, pk) without relying on a single trusted dealer. Assuming the same access structure \((t, n)\) as above, each trustee \(i\) obtains a share \(sk_i\) of the secret key \(sk\), and collectively obtains a public key \(pk\) [67]. This public key \(pk\) can now be used by any client to encrypt a secret, and at least \(t\) trustees must cooperate to retrieve the encrypted secret [27].

### 3.2.2 Protocol Outline

We present the outline of F3B protocols with two different threshold cryptographic schemes. Figure 3.1 presents the protocol outline.

#### 3.2.2.1 Protocol based on TDH2

**Setup:** Before an epoch, the secret-management committee runs a DKG protocol to generate a private key share \(sk_{smc}^i\) for each trustee and a collective public key \(pk_{smc}\) written onto the underlying blockchain. To offer chosen-ciphertext attack protection and to verify the correctness of secret shares, we utilize the TDH2 cryptosystem [27] containing NIZK proofs.

**Per-Transaction Protocol:**

1. **Write Transaction:** When creating a transaction, a sender needs to use a symmetric key \(k\) to encrypt the cleartext transaction, denoted as \(c_{tx} = enc_k(tx)\). To distribute the symmetric key among the SMC trustees, \(k\) is encrypted with \(pk_{smc}\), which is obtained from the underlying blockchain. Along with necessary proofs, the ciphertext \(c_k\) is constructed. In the end, the ciphertext and encrypted transaction are packed together as a whole, forming the \((c_{tx}, c_k)\) which is sent to the consensus group and written onto the blockchain.

2. **Shares Preparation by Trustees:** Once written, each secret-management committee trustee reads \(c_k\) from the sender’s transaction and prepares their decrypted share of \(k\).

3. **Key Reconstruction:** When the sender’s transaction \((c_{tx}, c_k)\) is committed onto the underlying blockchain (after \(m\) block confirmations), each secret-management committee trustee releases their share to the consensus group. The consensus group verifies the decrypted shares and uses them to reconstruct \(k\) by Lagrange interpolation of shares when there are at least \(t\) valid shares.
4. **Decryption and Execution:** With the symmetric key, the consensus group decrypts the transaction $tx = \text{dec}_k(c_{tx})$ using $k$, and execute $tx$ accordingly.

**Resharing Protocol:** To modify a SMC’s membership and to offer backward secrecy over epochs, an SMC can periodically run a verifiable resharing protocol [65] to replace certain trustees or redistribute the trustees’ private keys. Unlike DKG, resharing keeps the epoch’s public key, thus preventing undesirable interruptions of encryption services.

### 3.2.2.2 Protocol based on PVSS

#### Per-Transaction Protocol:

0. **Share Preparation By Sender:** For every transaction, the sender runs the PVSS protocol [28] to generate an encrypted key share $share_i$ for each trustee, as well as a corresponding NIZK proof and public polynomial commitment. The proof and commitment can be used to verify the correctness of key share and protect against chosen-ciphertext attacks. The sender obtains the symmetric key $k$ from the PVSS protocol.

1. **Write Transaction:** A sender first creates the ciphertext $c_k$ with the key shares, NIZK proofs, and commitments generated during share preparation. Next, the sender creates their transaction and symmetrically encrypts it by using the symmetric key $k$, denoted as $c_{tx} = \text{enc}_k(tx)$. Finally, the sender sends $(c_{tx}, c_k)$ to the consensus group who writes the pair onto the blockchain.

2. **Shares Preparation by Trustees:** Same as (2) in 3.2.2.1.

3. **Key Reconstruction:** Same as (3) in 3.2.2.1.

4. **Decryption and Execution:** Same as (4) in 3.2.2.1.

### 3.2.3 Overhead Analysis

We analyze both protocols’ overheads. Write Transaction (step 1) is identical to sending a transaction to the underlying blockchain. We assume trustees can finish Shares Preparation by Trustees (step 2) within the confirmation time.
Figure 3.2: In Ethereum, once they are inserted in the blockchain, the transactions are executed and committed after receiving \( m \) block confirmations. Whereas, in F3B, transactions are encrypted, and their executions are postponed after receiving \( m \) block confirmations when the secret-management committee releases the encryption keys. Both scenarios have a similar commitment time.

of the \( \text{tx}^* \). Hence, the time for steps 1 and 2 is equivalent to submitting a transaction to the underlying blockchain, and waiting until it is committed, which takes \( mL_0 \) time based on the baseline model (Section 3.2.1). As in the PVSS protocol, the sender can finish Share Preparation By Sender (step 0) before having the \( \text{tx} \); thus step 0 does not contribute to the transaction latency. Comparing our protocol with the baseline, Key Reconstruction (step 3) and Decryption and Execution (step 4) are additional steps that lead to extra delay, and we denote the time of those steps to be \( L_r \).

\*As we presented in Chapter 4, confirmation time in Ethereum is much longer than the share preparation time by trustees.
Figure 3.2 illustrates the conceptual difference in commitment and execution time between F3B and the baseline blockchain model. In F3B, the secret-management committee introduces an execution delay of \( m \) blocks by releasing the secret keys with a delay of \( m \) blocks. However, in both F3B and the baseline blockchain model, the recipient of a transaction should not accept the transaction until it is committed, in order to prevent attacks such as double-spending. Therefore, from a commercial perspective, F3B is similar to the baseline model as it exhibits the commitment time of a transaction, which is essential to the recipient\(^*\).

### 3.2.4 Comparison of Two Protocols

When applying THD2 and PVSS to F3B, each scheme has some advantages and disadvantages. Some of the differences are discussed further in Chapter 4 and Chapter 5.

- **Preprocessing:** In THD2, the secret-management committee needs to do DKG per epoch, whereas in PVSS, the sender needs to prepare shares per transaction. This difference also illustrates that in THD2, the trustees in SMC must know each other and exchange messages to do the DKG setup, while in PVSS the trustees can just register themselves and never communicate with each other.

- **Membership:** In THD2, the secret-management committee’s membership is fixed per epoch without running the resharing protocol, whereas in PVSS, the sender can choose different trustees from the secret-management committee for each transaction, providing the best flexibility.

- **Ciphertext:** THD2 has a constant ciphertext length, whereas PVSS’s ciphertext grows linearly with the size of the secret-management committee.

In conclusion, no one protocol can completely replace another. System designers need to choose one or both protocols based on their needs and constraints to mitigate front-running.

\(^*\)In F3B, transaction finalization is slower due to the key reconstruction and delayed execution after transaction commitment. However, the overhead is negligible compared to commitment time, as discussed in Chapter 4.
3.3 Full Protocol

We provide detailed protocols for F3B based on two different threshold cryptographic schemes in this subsection.

3.3.1 Protocol based on TDH2

We define a cyclic group $G$ of prime order $q$ with generators $g$ and $\bar{g}$ that are known to all parties, and we define the following two hash functions: $H_1 : G^5 \times \{0, 1\}^l \rightarrow G$ and $H_2 : G^3 \rightarrow \mathbb{Z}_q$.

**Step 0: DKG Setup.** Before initiating an epoch, the secret-management committee runs a DKG protocol to generate a shared public key $pk_{smc} = g^{sk_{smc}}$, and shares of the private key for each trustee are denoted as $sk_i$. The corresponding private key $sk_{smc}$ can be reconstructed only by combining $t$ private key shares. All trustees also know the verification keys $h_i = g^{sk_i}$. We assume that $pk_{smc}$ and $h_i$ are written into the blockchain as metadata, e.g., in the first block denoting the beginning of this epoch. We adopt the synchronous DKG protocol proposed by Gennaro et al. [67].

**Step 1: Write Transaction.** For the write transaction step, we use the encryption protocol presented by the TDH2 cryptosystem [27].

The sender and the consensus group execute the following protocol to write the $tx_w$ on the underlying blockchain. The sender then starts the protocol by performing the following steps to create the transaction $tx_w$:

1. Obtain the secret-management committee threshold collective public key $pk_{smc}$ from the underlying blockchain.

2. Generate a symmetric key $k$ and encrypt the transaction $tx$ using authenticated symmetric encryption as $e_{tx} = \text{enc}_k(tx)$.

3. Embed $k$ as a point $k' \in \mathbb{G}$, and choose $r, s \in \mathbb{Z}_q$ at random.

4. Compute:

$$c = pk_{smc}^r k', u = g^r, w = g^s, \bar{u} = \bar{g}^r, \bar{w} = \bar{g}^s,$$

$$e = H_1 (c, u, \bar{u}, w, \bar{w}, L), f = s + re,$$
where $L$ is the label of the underlying blockchain\(^*\).

5. Finally, form the ciphertext $c_k = (c, L, u, \bar{u}, e, f)$ and construct the write transaction as $tx_w = [c_{tx}, c_k]_{\text{sig}_A}$ signed with the sender’s private key $sk_A$.

6. Send $tx_w$ to the consensus group.

Upon receiving the $tx_w$, the consensus group writes it onto the blockchain.

**Step 2: Shares Preparation by Trustees.** Each trustee $i$ performs the following steps to prepare its decryption share for the consensus group.

1. Extract $L$ from $c_k$ and verify that $L$ is consistent with the underlying blockchain’s metadata.

2. Verify the correctness of the ciphertext $c_k$ using the NIZK proof by checking:

   $$e = H_1(c, u, \bar{u}, w, \bar{w}, L),$$

   where $w = g^f / u^e$ and $\bar{w} = \bar{g}^f / \bar{u}^e$.

3. If the $tx_w$ is valid, choose $s_i \in \mathbb{Z}_q$ at random and compute:

   $$u_i = u^{sk_i}, \quad \hat{u}_i = u^{s_i}, \quad \hat{h}_i = g^{s_i}, \quad e_i = H_2(u_i, \hat{u}_i, \hat{h}_i), \quad f_i = s_i + sk_i e_i.$$

4. Create and sign the share: $\text{share}_i = [u_i, e_i, f_i]_{\text{sig}_A}$.

   In (2), the NIZK proof ensures that $\log_g u = \log_{\bar{g}} \bar{u}$, guaranteeing that whoever generated the $tx_w$ knows the random value $r$. If the value of $r$ is known, then the transaction can be decrypted; as it is impossible to generate $tx_w$ without knowing the plaintext transaction, this property prevents replay attacks mentioned in Section 3.5.2.

**Step 3: Key Reconstruction.** Once the transaction has received $m$ block confirmations, each trustee sends their decryption share to the consensus group.

Upon receiving the shares, each node in the consensus group executes the following:

\(^*\)This can be the hash of the genesis block.
1. Each node in the consensus group verifies the share by checking:

\[ e_i = H_2 \left( u_i, \hat{u}_i, \hat{h}_i \right), \]

where \( \hat{u}_i = \frac{u_i}{u_i^{\alpha_i}} \) and \( \hat{h}_i = \frac{g^{\alpha_i}}{h_i^{\alpha_i}} \).

2. After receiving \( t \) valid shares, the set of decryption shares is of the form:

\[ \{(i, u_i) : i \in S\}, \]

where \( S \subset \{1, \ldots, n\} \) has a cardinality of \( t \). Each node then executes the recovery algorithm that does the Lagrange interpolation of the shares:

\[ pk_{smc}^r = \prod_{i \in S} u_i^{\lambda_i}, \]

where \( \lambda_i \) is the \( i^{th} \) Lagrange element.

3. Recover the encoded encryption key:

\[ k' = c(pk_{smc}^r)^{-1} = (pk_{smc}^r k')(pk_{smc}^r)^{-1}. \]

4. Retrieve \( k \) from \( k' \).
   
   In (1), the NIZK proof ensures that \( (u, h_i, u_i) \) is a Diffie-Hellman triple, i.e., that \( u_i = u^{sk_i} \), guaranteeing the correctness of the share.

**Step 4: Decryption and Execution.**

1. Decrypt the transaction \( tx = \text{dec}_k(c_{tx}) \).

2. Execute the transaction following the consensus group’s defined rules.

### 3.3.2 Protocol based on PVSS

Let \( G \) be a cyclic group of prime order \( q \) with two distinct generators \( g \) and \( h \) where the decisional Diffie-Hellman assumption holds. The secret-management committee has a set of trustees \( N = \{1, \ldots, n\} \), where each trustee is identified by a unique index \( i \), and has a private key \( sk_i \) and a corresponding public key \( pk_i = g^{sk_i} \). The underlying blockchain stores all the trustees’ public keys; thus, they are accessible to everyone. We follow the PVSS scheme presented by Berry Schoenmakers [28]. The protocol runs as follows:
Step 0: Share Preparation by Sender. The sender starts the protocol to prepare key shares and the symmetric key:

1. Deriving the generator $h$ from the label of the underlying blockchain $L$ by computing $h = H(L)$ using Elligator maps [68]. This method will protect against replay attacks discussed in Section 3.5.2.

2. Pick a random secret sharing polynomial $s(x) = \sum_{j=0}^{t-1} a_j x^j$ of degree at most $t - 1$. $s = g^{s(0)}$ is the secret to be shared.

3. Compute the encrypted shares $\hat{s}_i = pk_i^s(i)$ of secret $s$ for every secret-management trustee $i$ that the sender wishes to include, create the corresponding NIZK proof $\pi_{\hat{s}_i}$, and the polynomial commitments $b_j = h^{a_j}$, for $0 \leq j \leq t - 1$.

4. Use $k = H(s)$ as the symmetric key.

The NIZK proof $\pi_{\hat{s}_i}$ will be used to verify that the corresponding encrypted share $\hat{s}_i$ is consistent, i.e., a proof of knowledge of the unique $s_i$ that satisfies:

$$X_i = h^{s(i)}, \hat{s}_i = pk_i^{s(i)}$$

where $X_i = \prod_{j=0}^{t-1} b_j^i$. $\pi_{\hat{s}_i}$ shows that $\log_h X_i = \log_{pk_i} \hat{s}_i$, and to generate it the sender picks randomly $w_i \in \mathbb{Z}_q$ and computes $a_{1i} = h^{w_i}, a_{2i} = pk_i^{w_i}$. Using Fiat-Shamir’s technique, the sender then computes the challenge $c_i$ and response $r_i$ as follows:

$$c_i = H(X_i, \hat{s}_i, a_{1i}, a_{2i}), r_i = w_i - s(i)c_i$$

Each proof $\pi_{\hat{s}_i}$ consists of $c_i$ and $r_i$.

Step 1: Write Transaction. Once the sender has the tx, they can write it to the underlying blockchain by the following steps:

1. Form the ciphertext $c_k = (\hat{s}, \pi_{\hat{s}}, i, b_j)$, encrypt the transaction $tx$ using authenticated symmetric encryption as $c_{tx} = enc_k(tx)$.

2. Construct the write transaction as $tx_w = [c_{tx}, c_k]_{sig_A}$ signed with the sender’s private key $sk_A$.

3. Send $tx_w$ to the consensus group.

Upon receiving the $tx_w$, the consensus group commits it onto the blockchain following its defined consensus rules.
Step 2: Shares Preparation by Trustees. Each trustee $i$ performs the following steps to prepare its decryption share.

1. Find the corresponding $\hat{s}_i, \pi_{\hat{s}_i}, b_j$ using the index $i$.

2. Verify the correctness of the encrypted share $\hat{s}_i$ using the NIZK proof. Compute $X_i = \prod_{j=0}^{t-1} b_j^i$ from the polynomial commitments $b_j$, $0 \leq j < t$. And compute $a'_{1i} = h^{r_i} X_i^{\kappa}$, $a'_{2i} = pk_i^{c_i} \hat{s}_i^{c_i}$. Check that $H(X_i, \hat{s}_i, a'_{1i}, a'_{2i})$ matches $c_i$.

3. If the encrypted share $\hat{s}_i$ is valid, decrypt the share by computing $s_i = (\hat{s}_i)^{\kappa_{i}}$. Create a new NIZK proof $\pi_{s_i}$ to verify the share is correctly decrypted. This proof shows that $\log_g pk_i = \log_g s_i \hat{s}_i$.

4. Create and sign the share: $\text{share}_i = [s_i, \pi_{s_i}]_{\text{sig}_i}$.

Step 3: Key Reconstruction. Once the transaction has received $m$ block confirmations, each trustee sends their decryption share to the consensus group.

Upon receiving the shares, each node in the consensus group executes the following:

1. Each node in the consensus group verifies the correctness of the decrypted share $\hat{s}_i$ using the NIZK proof $\pi_{s_i}$.

2. After receiving $t$ valid shares, each node then executes the Lagrange interpolation to recover $s$ from the shares:

$$s = \prod_{i=1}^{t} s_i^{\lambda_i},$$

where $\lambda_i$ is the $i^{th}$ Lagrange element.

3. Recover the encryption key $k = H(s)$.

Step 4: Decryption and Execution. Same as step 4 in 3.3.1.

3.4 Achieving the System Goals

In this section, we discuss how F3B achieves the system goals outlined in Section 3.1.1.
**Front-Running Protection:** prevents entities from conducting front-running.

F3B offers front-running protection by ensuring that the content of a pending transaction is only revealed when the transaction is committed. According to the definition of front-running, attackers can benefit from advanced knowledge of the pending transactions. While F3B sacrifices some transparency, the only one who knows the content of a pending transaction now is the sender, who is financially incentivized to keep the transaction encrypted. Only after the transaction is committed and decrypted, can everyone accesses the transaction content, which is too late for most front-running. So now attackers almost know nothing about pending transactions and cannot launch front-running attacks. However, speculative front-running attacks using side channels (e.g., metadata such as sender’s address and transaction size) of the transaction are still the concern, and this is discussed in Section 3.1.4 and Section 5.4. A more comprehensive security analysis is presented in Section 3.5.

**Decentralization:** mitigates a single point of failure or compromise.

By using DKG and secret sharing schemes, F3B ensures that the SMC can tolerate up to $t - 1$ malicious trustees and up to $n - t$ offline trustees. The decentralized approach mitigates the risk of a single point of failure or compromise, and the detailed properties are introduced in the DKG [67], THD2 [27], and PVSS [28] protocols.

**Confidentiality:** ensures that a transaction is only revealed after it has been committed by the underlying consensus layer.

F3B maintains confidentiality by encrypting each transaction with a newly generated symmetric key. The symmetric key is (a) encrypted under the secret-management committee’s public key in TDH2-based protocol, (b) embedded into the encrypted shares in PVSS-based protocol. In both protocols, $f + 1$ trustees are required to retrieve the symmetric key. According to the threat model, up to $f$ trustees can behave maliciously; this ensures that the transaction’s content remains confidential until it is committed. A more detailed security analysis is also presented in Section 3.5.

**Compatibility:** remains agnostic to the underlying consensus algorithm and smart contract implementation.

This allows F3B to be agnostic to the underlying consensus algorithm and enables existing smart contracts to benefit from front-running protection automatically.

F3B only requires modifications at the execution layer, so that encrypted transactions are allowed and processed properly. How the consensus algorithm
works does not affect the mechanism of F3B, thus making it independent of the consensus layer and compatible with various algorithms. Besides, existing smart contracts do not need to be revised and re-deployed to benefit from F3B, and can gain front-running protection automatically when the new encrypted transactions are used in the execution layer.

**Low-Latency:** presents low-latency transaction overhead.

F3B exhibits low-latency overhead as it requires clients to write only one transaction onto the underlying blockchain. At the same time, the extra latency introduced by the encryption and decryption process are very negligible according to our evaluation. Compared with other work that need multiple transactions, we greatly reduce the latency overhead and protocol complexity, and increase the security guarantees. The evaluation of F3B’s latency overhead is shown in Chapter 4.

### 3.5 Security Analysis

In this section, we present a more detailed security analysis of F3B’s protocol.

#### 3.5.1 Front-Running Protection

In the context of front-running protection, we reason that an attacker cannot launch front-running attacks with confidence in a financial profit, even when at most \( f \) malicious trustees collude and collaborate. While we acknowledge that there are many different ways of front-running, including speculative front-running attacks, we now only consider common front-running that makes use of the knowledge about pending transactions. Under the F3B architecture, the attacker’s inability to access the content of a transaction before it is committed to the underlying blockchain prevents them from knowing any content of the transaction. Thus it is impossible for any scripts or bots to search the network and find opportunities to make a profit from front-running.

Additionally, the secure symmetric encryption used in F3B ensures that the attacker cannot decrypt the transaction based on its ciphertext. From our threat model, the collusion limit of at most \( f \) trustees reinforces the security of F3B, as \( f + 1 \) trustees are required to recover or gain information about the symmetric key. The properties of the underlying cryptographic schemes, such as TDH2 [27], DKG [67], and PVSS [28], further prevent the attacker from obtaining the private key or reconstructing the symmetric key.
3.5.2 Replay Attack

Regarding the replay attack, two scenarios are discussed.

In the first case, where an adversary copies a pending encrypted transaction and submits it as their own transaction to reveal the transaction’s contents. If this transaction can be revealed before the victim’s transaction is committed, the attacker would easily launch a front-running. We explain why the attacker cannot benefit from such a strategy. If the adversary directly copies the ciphertext $c_k$, the encrypted transaction $c_{tx}$ from $tx_w$, and creates a new transaction $tx'_w$ digitally signed with the attacker’s signature. In this case, the adversary’s $tx'_w$ should be committed and decrypted no earlier than the victim’s transaction $tx_w$, because they have the same transaction fee and this $tx'_w$ is submitted to the underlying blockchain later. The attacker has no incentive to pay much more transaction fee for the $tx'_w$ to be included earlier, because the content of the $tx_w$ is unknown and cannot guarantee any profits from revealing this transaction earlier with extra cost.

In the second case, the adversary instead copies the ciphertext and the encrypted transaction to craft a new transaction, and sends the transaction to a blockchain with fewer block confirmations. We argue that the adversary is unable to form a valid transaction with the altered content in our two protocols.

Consider two blockchains $b_1$ and $b_2$, and the required number of block confirmations for them are $m_1$ and $m_2$, while $m_1 > m_2$. Assuming the adversary changes the label $L$ to $L'$ for the blockchain $b_2$ instead of blockchain $b_1$, the secret-management committee might successfully decrypt the transaction. However, we prove that it is hard to form a valid new transaction with $L'$ by the adversary.

For the TDH2 protocol, the adversary would need to generate $e' = H_1(c, u, \bar{u}, w, \bar{w}, L')$ and $f = s + re'$, without knowing the random parameter $r$ and $s$. Suppose the adversary generates $u = g^s$, $\bar{u} = \bar{g}^{s'}$ with $r \neq r'$ and $w = g^s$, $\bar{w} = \bar{g}^{s'}$ with $s \neq s'$. For $tx'_w$ to be valid, we must have $g^f = wu^e$ and $\bar{g}^f = \bar{w}\bar{u}^e$, this implies that $(s - s') + e(r - r') = 0$. As $r \neq r'$, the adversary has only a negligible chance of having $tx'_w$ pass verification.

For the PVSS protocol, the adversary must replace the original generator $h$ with $h'$ derived from $H(L')$. Hence, the adversary has to do the proofs without secrets. The security of PVSS guarantees that they only have a negligible probability of succeeding. Note that the base point has to be random to ensure the security. Using Elligator maps [68] guarantees that the generator $h$ is random.
3.6 Incentive

In order to ensure the proper operation and honest participation of actors in the F3B protocol, appropriate incentives need to be in place. In this section, we address the important incentives that prevent spamming of transactions and discourage dishonest collaboration among trustees to reveal transactions in advance.

3.6.1 Spamming Protection

Since transactions are no longer in cleartext while pending, a malicious sender might flood the blockchain with non-executable transactions that have inadequate transaction fees or meaningless content. To prevent adversaries from spamming the blockchain with such transactions and delaying the commitment of honest transactions, F3B introduces a storage deposit mechanism. In addition to the traditional transaction fee (gas fee in Ethereum), a sender is required to deposit a storage fee, which is based on the transaction’s size. This deposit will be partially or fully refunded to the sender after the transaction is successfully executed. The misbehaving sender will instead be penalized and not get a refund. This mechanism discourages spamming attacks and ensures that the blockchain is not easily congested with non-executable transactions.

3.6.2 Operational Incentive

Similar to the miners and validators in blockchains, an incentive structure is necessary for the proper behavior of the secret-management committees. While execution fees can also be used as rewards for the SMC trustees, they are not sufficient to prevent collusion among trustees for more financial gain. For example, trustees may collaboratively collude with some consensus group nodes by providing decryption shares to them before the transactions are committed. Since detecting out-of-band collusion is very hard, we need to limit the expected collusion profit and reward any reporting of malicious behaviors. F3B proposes an incentive structure where each trustee in an SMC is required to lock a collateral amount $c$. In exchange for this collateral, trustees are rewarded proportionally to the staked amount, earning $ac$ for the validation and decryption services they provide. Since efficient collusion requires at least $t$ trustees, the collateral acts as a deterrent against collusion, as the profits that malicious trustees could gain from front-running, would be no more than the...
amount of collateral that they would all lose, which is \((1 + a)ct\) according to the slashing protocol introduced later in Section 3.6.3. The epoch length is also associated with the collateral, because the higher a collateral is required, the longer an epoch could be. This ensures that trustees have a strong incentive to act honestly and discourages collusion for personal gain.

By incorporating these incentive mechanisms, F3B ensures the integrity and efficiency of the protocol while deterring malicious behavior and incentivizing honest participation.

### 3.6.3 Slashing Protocol

As discussed in the above subsection, a slashing protocol is needed to reward any reporting of misbehavior, whether it is of a trustee or the whole secret-management committee. This protocol should come with strict proof, so that the individuals that can really prove collusion or other misbehavior can get the reward, and at the same time false accusations will be prevented.

As discussed in Section 3.6.2, each trustee is required to lock an amount as collateral. This stake of cryptocurrency will be stored in a smart contract, which is designed to handle any disputes between the reporter and susceptible misbehaving trustees.

To initiate a dispute, a reporter invokes the smart contract with the decryption shares for a currently not revealed transaction and their own stake. If the smart contract validates that the decryption shares are indeed correct, and that this dispute is initiated before the transaction is revealed by the secret-management committee, then the suspects’ stake is forfeited and transferred to the reporter.

In the TDH2-based protocol, to prove a decryption share that is correct, the reporter submits \([u_i, e_i, f_i]\) such that \(e_i = H_2(u_i, \hat{u}_i, \hat{h}_i)\) where \(\hat{u}_i = \frac{u_i}{u_i^s}\) and \(\hat{h}_i = \frac{h_i}{h_i^s}\).

In the protocol based on PVSS, the plaintiff submits \([s_i, \pi_{s_i}]\), where \(\pi_{s_i}\) is the NIZK proof that shows \(\log_g p_{k_i} = \log_s \hat{s}_i\). Even if the sender knows \(s_i\), it is impossible to maliciously slash a trustee without the \(\pi_{s_i}\), which only the corresponding trustee knows.

To deploy the slashing protocol, the smart contract should have access to the ciphertexts of any pending transaction, to get parameters such as \(u\) or \(\hat{s}_i\) for verification. This is guaranteed because the smart contract should also run in the current blockchain and see all the encrypted transactions.
Chapter 4

Results and Analysis

The F3B is proposed and analyzed based on post-merge Ethereum [29] as the underlying blockchain. The implementation of F3B is written in Golang [69] and built upon the DEDIS Ledger Architecture (Dela) [30]. Due to the limitation and scope of the project, the evaluation are done on a simulated network running on our server, and for the sake of convenience, all the tests are done on Dela without actual communication with the Ethereum blockchain network.

To show what the performance would be like in Ethereum, we adhere to some regulations. To align with Ethereum’s security assumptions, the length of each epoch is set to 6.4 minutes in our evaluation. During an epoch, the trustees forming the secret-management committee are randomly selected from the validators. For the cryptographic primitives used in F3B, we instantiate them using the Edward25519 elliptic curve with 128-bit security. This implementation of cryptographic primitives is supported by Kyber package [70], an advanced cryptographic library also developed by the DEDIS lab. Experiments and evaluations of F3B were conducted on a server with 32GB of memory and 40 CPU cores running at 2.1GHz. The network communication delay was simulated to be a fixed 100ms. Furthermore, in Chapter 5, there is a more detailed discussion on the integration of F3B with Ethereum.

4.1 Latency

In Figure 4.1, the breakdown latency of each step in F3B is presented. We show the time needed for both TDH2 and PVSS protocols after a transaction commitment: (a) shares preparation by trustees, and (b) key reconstruction,
Figure 4.1: The breakdown latency of each step in F3B by varying the number of secret-management committee trustees from 8 to 128 nodes.  

and (c) decryption and execution. In addition, we show the time needed for PVSS shares generation by the sender in the purple of Figure 4.1. The results are obtained when changing the number of trustees in the SMC from 8 to 128.

From Figure 4.1, it is clear that the time needed in every step for both protocols is quite similar, and is growing as the number of trustees increases. Sometimes the PVSS spends more time than TDH2 during shares preparation by trustees, which is mainly because of the communication cost. As discussed later, the ciphertext size of PVSS is much bigger than TDH2, which introduces some extra delay when receiving the content and decrypting shares.

As discussed in Section 3.2.3, only (b) and (c) represent the overhead at the per-transaction level, which will count toward the end-to-end latency. Notably, we add a constant network communication delay of 100ms to the end-to-end latency, stimulating the real-world network.

According to Section 3.2.2, the overall transaction latency with F3B is $mL_b + L_r$, and the block time is 12 seconds in post-Merge Ethereum, which means $L_b = 12$ [71]. And, by official standards, 64 block confirmations are required, which is two epochs, for a block to be "finalized" and immutable, so $m = 64$ [32].
Figure 4.2: A comparison of the sender commit-and-reveal approach latency with F3B against a baseline modeled in Ethereum. The string “F3B-X” represents X trustees.

Figure 4.2 shows the end-to-end latency comparison between the baseline protocol (Section 3.2.1), a traditional commit-and-reveal protocol discussed in Strawman I (Section 2.6.1), and F3B’s two protocols. Again, we test the results when changing the size of the secret-management committee follow the same manner as latency, and the trustee number is stated as "F3B-X".

In the new PoS consensus of Ethereum, committing any transaction requires a time of $mL_b = 768$ seconds. The baseline protocol, which has no front-running protection, directly submits and executes one plaintext transaction, and waits for it to be finally committed. It has a total latency of 768 seconds according to our definition.

In comparison, the traditional commit-and-reveal protocol introduced in Strawman I offers front-running protection by requiring two transactions that cannot be overlapped. The commit transaction comes first, which can be a hash, followed by the reveal transaction which is plaintext, each requiring 768 in Ethereum, and they add up to present a total latency of 1536 seconds.

The Submarine, which is an improved approach of traditional commit-and-reveal, hides the smart contract address at the cost of requiring three
transactions, resulting in a 200% latency overhead compared with the baseline [14, 31].

In contrast, F3B adopts the threshold encryption similar to Strawman III (Section 2.6.3). Considering the architecture also as a commit-and-reveal process, where the reveal phase is automatically done by the SMC and consensus group. The sender is not involved in the shares preparation by trustees and key reconstruction steps, so no extra transactions or communication is required for the sender. This distinguishes F3B from other application-based commit-and-reveal approaches, as F3B only needs to send one transaction to the underlying blockchain.

Looking back to Figure 4.2, it illustrates that F3B has a low-latency overhead. When running with a SMC size of 128 trustees, the end-to-end latency of TDH2 and PVSS protocols are 197ms and 205ms, equivalent to 0.026% and 0.027% of the commitment time of Ethereum, respectively.

We acknowledge that some Ethereum users may accept a lower confirmation number to accept a transaction, even though Ethereum officially requires 64 blocks [32]. Without loss of generality, we outline different confirmation numbers with F3B’s latency overhead in Table 4.1. In general, all the latency overhead is within 0.1%, which is really negligible compared with the time spent waiting for block confirmations. Thus, we can claim that the F3B achieves low-latency front-running protection. This highlights the efficiency of F3B, making it an attractive solution for achieving front-running protection in Ethereum and other blockchain systems.

<table>
<thead>
<tr>
<th>Confirmations</th>
<th>Latency Overhead varying SMC sizes</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TDH2</td>
</tr>
<tr>
<td></td>
<td>64</td>
</tr>
<tr>
<td>8</td>
<td>0.164%</td>
</tr>
<tr>
<td>16</td>
<td>0.082%</td>
</tr>
<tr>
<td>32</td>
<td>0.041%</td>
</tr>
<tr>
<td><strong>64</strong></td>
<td><strong>0.020%</strong></td>
</tr>
<tr>
<td>128</td>
<td>0.010%</td>
</tr>
</tbody>
</table>

Table 4.1: Latency Overhead for Ethereum Blockchain
4.2 Throughput

We use the end-to-end latency defined above to calculate the throughput of F3B. If we decrypt and reconstruct transactions one by one, the overall throughput of F3B would be very limited due to the sequential execution. In blockchain such as Ethereum, transactions are always batched for execution to improve efficiency. So it is very natural for the SMC and consensus group in F3B to batch process the key shares and encrypted transactions. In practice, every trustee will validate and decrypt multiple key shares for different transactions, and upon receiving the batched decryption shares, the consensus nodes will also do batch reconstruction. Figure 4.3 presents the F3B’s throughput results and corresponding overall latency varying the batchsize of transactions. We now fix the SMC size to 128 trustees, as this is the greatest value we have tested, and can reflect on real-world network requirements. In this experiment, we assume that the underlying blockchain is not the bottleneck of throughput.

When increasing the batching size from 1 to 2048, the throughput is increased from 5 txns/sec to 359 txns/sec with the TDH2 protocol, and from 4 txns/sec to 348 txns/sec with the PVSS protocol. Intuitively, increasing throughput by batch process also introduces higher latency: With a batching size of 2048, the key reconstruction step of TDH2 now takes 5.71 seconds to process, and the same step of PVSS takes 5.88 seconds; this latency is equivalent to a 0.74% and 0.77% latency overhead over Ethereum, still very negligible.

Our results show that F3B provides more than sufficient throughput to support Ethereum (15 txns/sec [72]), and almost all the permissionless blockchains and some permissioned blockchains [33, 34]. For some permissioned blockchains such as Ripple [35], the throughput can be as high as over 1000 txns/sec. However, we argue that in a permissioned blockchain that is always not decentralized, trustless, and open to the public, a basic trust is established between the nodes in the network, and it is not that important to provide front-running protection.

4.3 Reconfiguration in TDH2

In the TDH2 protocol, trustees need to exchange messages to generate a collective public key. This happens during the DKG setup and resharings, which is the process for reconfiguring a secret-management committee. The
Varying batching size with 128 trustees

Figure 4.3: Overall performance of the two protocols

cost of DKG setup and resharing in terms of latency is illustrated in Figure 4.4.

The setup phase of TDH2, which is the DKG between trustees, is a one-time operation per epoch. In our experiments with a SMC size of 128 trustees, DKG setup spends approximately 144 seconds. This accounts for about 37.5% of Ethereum’s epoch time, which is 384 seconds according to previous explanation. A detailed discussion about how the TDH2 protocol makes transition between two epochs is presented in Section 5.1.

To provide dynamic membership and backward secrecy, a secret-management committee can run a verifiable resharing protocol [65]. The resharing protocol happens within an epoch, and allows for changes in the membership of secret-management committee while preserving the collective public key. In this case, updates of SMC members do not interrupt transaction encryption and decryption, and ensure the backward secrecy. In Figure 4.4 three different scenarios of resharing are evaluated and presented: (a) resharing among the same committee, (b) replacing one trustee, and (c) replacing a quarter of trustees. All of them exhibit latency of the same magnitude, indicating that the resharing cost is not significantly related to the number of trustees changed.

These results demonstrate that F3B’s resharing process is efficient and can
Figure 4.4: The latency cost of DKG setup and three resharing scenarios.

be seamlessly integrated within an epoch, allowing for dynamic committee changes and maintaining the system’s security guarantees while incurring minimal latency overhead.

### 4.4 Ciphertext Storage

In the TDH2 protocol, as the symmetric key is encrypted with the shared public key, the size of $c_k$ is independent of the number of trustees. We also optimize the original TDH2 protocol to remove the label $L$ from the ciphertext but only insert $L$ in the computation and verification steps of each party (consensus group, secret-management committee, sender) for protection against replay attacks (Section 3.5.2). Ultimately, we achieve 80 bytes per transaction of the storage cost for the ciphertext presented in Table 4.2.

In the PVSS protocol, however, the ciphertext contains encrypted shares, NIZK proofs, and polynomial commitments. The size of the ciphertext is thus positively related to the size of the secret-management committee, indicating the size of $c_k$ approximately grows linearly with the number of trustees, as demonstrated in Table 4.2.
<table>
<thead>
<tr>
<th>Number of trustees</th>
<th>TDH2 Protocol</th>
<th>PVSS protocol</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>80</td>
<td>792</td>
</tr>
<tr>
<td>16</td>
<td>80</td>
<td>1568</td>
</tr>
<tr>
<td>32</td>
<td>80</td>
<td>3120</td>
</tr>
<tr>
<td>64</td>
<td>80</td>
<td>6224</td>
</tr>
<tr>
<td>128</td>
<td>80</td>
<td>12432</td>
</tr>
</tbody>
</table>

Table 4.2: Size of ciphertext for two protocols with different Secret-management Committee sizes.
Chapter 5

Discussion

In this chapter, we discuss some challenges we meet during system design and deployment. We leave a detailed analysis for future work.

5.1 Transition of Epoch

In the transition between epochs, the TDH2 will rerun the DKG setup and generates a new collective public key. Users may face challenges in choosing the correct public key for encrypting their symmetric key for transactions in this protocol. Remember that in previous work using threshold encryption such as Shutter [23, 24] and Fairblock [22], the encrypted transaction might be revealed undesirably when the chosen key does not correspond to the actually included block. However, F3B does not have this issue even during epoch transitions, because it is ensured that uncommitted transactions are never revealed, regardless of the chosen key. If a transaction symmetric key is encrypted by the public key of the last epoch, the decryption process will just fail because all trustees now hold the new key. Although users may need to retry with the correct epoch key, the infrequency of epoch transitions compared with block creation further reduces the impact on users. The key property of F3B is per-transaction encryption, making every transaction securely encrypted.

To further address this challenge of epoch transition, the expiring epoch committee can provide a grace period during which both the old and new epoch keys are valid. This allows users more time to transition to the new epoch key, reducing the risk of choosing an incorrect key.
5.2 Ethereum Gas Fees

Regarding Ethereum gas fees, incorporating F3B on Ethereum requires considering gas limits and fees. Ethereum utilizes gas fees to cover the cost of executing transactions and imposes a maximum gas limit per block. When integrating F3B, two conditions must be met: (1) the gas limit of each transaction must be specified in cleartext, and (2) the sum of gas limits for all transactions within a block must not exceed the block gas limit. This opens up the possibility of a spamming attack where an adversary submits transactions with high gas limits, leaving little room for other transactions.

Given the fact that in F3B, transactions are now encrypted when pending, the actual gas fee cannot be determined precisely by the miners or validators because of the uncertainty of global states. To mitigate this issue, one possible approach could be to burn the remaining unused gas. However, this may be too strict in practice as the senders must pay more than they actually need. Alternatively, partial refunds could be considered, where a percentage of the remaining gas is refunded to the sender.

Addressing the gas fee challenge requires careful consideration, and the problem of transaction fee mechanism is not only about F3B and the underlying blockchain, but also closely connected to the economy and user rationality. This is a particular hard question and we leave any further discussion to future work.

5.3 Verifiable Key Propagation

In our proposed protocol, each consensus node is responsible for fetching the secret shares and performing Lagrange interpolation to reconstruct the symmetric key. However, it is possible to optimize the key propagation process by having one or a few consensus nodes reconstruct the symmetric key $k$, and propagate it to all other consensus nodes with a succinct proof. This approach would require additional storage overhead but would allow for faster validation and reaching consensus.

Instead of constructing their encrypted transaction as $(c_k, c_{tx})$, the sender additionally adds a hash of the symmetric key $h_k = H(k)$ as the third entry, creating the following signed write transaction: $tx_w = [c_k, c_{tx}, h_k]_{sig_{k, A}}$.

During key reconstruction, consensus nodes would verify if the hash of the received reconstructed key $k$ matches the hash published on the ledger $h_k$. If the hashes are consistent, a consensus node can skip the key reconstruction and
propagate the key $k$ to other nodes, and directly proceed with decrypting the transaction. However, if the hashes are inconsistent, the consensus node must still reconstruct the key from decryption shares. At the same time, the node can publish the shares to the underlying blockchain and initiate the slashing protocol against the node that provided an incorrect $h_k$.

5.4 Metadata Leakage

In F3B, adversaries cannot observe the contents of encrypted transactions until they are committed, which prevented most front-running. However, they can still exploit side channels such as transaction metadata for speculative attacks. For example, the sender’s address is leaked when paying the storage fee (Section 3.6.1) for publishing an encrypted transaction on the underlying blockchain. This leakage can be used to launch second-order front-running attacks by predicting the sender’s behavior based on their history.

To mitigate this type of attack, senders can use different addresses to pay for the storage fee, thereby preventing the linkage of their actions. Additionally, the underlying blockchain can offer anonymous payment options, such as utilizing privacy-preserving protocols like Zerocash [73] or employing mixing services [74].

Another source of side-channel leakage is the size of the encrypted transaction or the time it takes for the transaction to propagate. One possible solution for mitigating such metadata leakage is using Padded Uniform Random Blobs (PURBs) [75], which are the formats aiming to reduce metadata leakage.

Addressing metadata leakage requires careful consideration of various techniques and protocols to minimize the information exposed to potential attackers.

5.5 Key Storage and Node Catchup

In our protocol, a new node that wants to join the consensus group cannot execute historical transactions to catch up unless it obtains all decryption keys. One approach is for the secret-management committee or consensus group to store these keys independently from the blockchain. However, this would require maintaining an additional immutable ledger to store the keys.

To optimize the key storage process, an alternative solution is to store the keys as metadata on the underlying blockchain, which is already maintained
by the consensus nodes. The blockchain rules can require the storage of valid keys when producing blocks. This approach eliminates the need for an additional storage medium and leverages the existing infrastructure. However, this optimization introduces a timing issue regarding when the blockchain should require the consensus group to store the keys in a block. In our protocol, the transaction is committed at block height \( n \) and revealed at block height \( n + m \). Consequently, the earliest block to write the key would be at block height \( n + m + 1 \). The latest block height to write the key provides more flexibility, and a balance must be struck between the delay tolerance for all consensus nodes to retrieve the key and the time that consensus nodes must retain the key.

Assuming the key reconstruction step takes up to \( \delta \) block times, the key should be written in or before block \( n + m + \delta \). This ensures that all consensus nodes have sufficient time to retrieve the key and that it is retained for the necessary duration.

It is important to note that this setup works well for blockchains with a fixed block time. However, for blockchains where block time is probabilistic, additional considerations must be taken into account. In such cases, the key might not have been replicated to all consensus nodes by block height \( n + m + \delta \). In such circumstances, an artificial delay could be induced for new blocks to ensure that the key is available to all nodes.

### 5.6 Off-chain shares in PVSS

As presented in section 4.4, the cost of writing the \( c_k \) of PVSS-based protocol is high. However, this is the most straightforward way to ensure that the encrypted shares \( \hat{s}_i \) can be verified by the secret-management committee and the decrypted shares \( s_i \) are verifiable for all the consensus nodes. Recall that the NIZK proof generated by trustees shows that \( \log_g pk_i = \log_{s_i} \hat{s}_i \). To verify the correctness of \( s_i \), \( \hat{s}_i \) must be accessible to all the consensus nodes. Thus it is impossible for the sender to send \( \hat{s}_i \) to the secret-management committee via a private channel.

We attempt to propose a solution that allows the sender to directly propagate the ciphertext in the blockchain. Instead of writing data onto the transaction, the sender signs the ciphertext with the private key and broadcasts it to the blockchain network. The hash of \( \pi_{s_i} \) and \( b_j \) is included in the transaction now, for verifying that they are consistent. Using \( \pi_{s_i} \) and \( b_j \), the encrypted shares \( \hat{s}_i \) can be verified as well. The solution reduced transaction size but brings new concerns about the lifecycle and availability
of the ciphertext. The data must be accessible during the shares preparation by trustees and key reconstruction, and needs to be safely deleted by consensus nodes after successfully executing the transaction.

An alternative way would be to store the ciphertext constructed by the sender off-chain. Decentralized storage systems with redundancy such as IPFS [76] and Swarm [77] could provide data availability, and the hash pointer of data can be stored on the blockchain for integrity [78]. However, this will require additional security assumptions for the adopted blockchain.

We leave detailed analysis and design to future work.
Chapter 6

Conclusions and Future work

6.1 Conclusion

In this thesis, we have introduced F3B, a novel blockchain architecture that effectively addresses front-running attacks through the use of threshold encryption with a per-transaction scheme. Our evaluation of F3B has demonstrated its compatibility with different consensus algorithms and existing smart contract implementations. We have shown that F3B achieves the desired throughput while maintaining low-latency overhead, making it a suitable solution for integration with Ethereum and other permissionless blockchains.

By deploying F3B, modifications to the execution layer of a blockchain would be required. However, the benefits of F3B are substantial. The deployed blockchain would automatically provide standard front-running protection for all applications without requiring modifications to the smart contracts themselves. This means that developers and users can benefit from front-running protection without the need for additional implementation efforts.

F3B’s design ensures compatibility, confidentiality, and decentralization. It protects against front-running attacks, provides low-latency transaction processing, and achieves high throughput for permissionless blockchains. Furthermore, F3B’s security properties have been rigorously analyzed within the provided threat model.

Overall, F3B presents a practical and efficient solution for mitigating front-running attacks in blockchain applications. Its deployment can enhance the security and privacy of transactions with low-latency overhead.
6.2 Limitations

Even if the F3B architecture is well designed and evaluated, some limitations still exist and remain to be better addressed. By using threshold encryption, our solution sacrifices some transparency of the blockchain network, and might lead to strategy changes in many DeFi applications. While introducing a low-latency in our evaluation, the scalability of F3B is not well tested. It remains to be seen whether F3B can behave smoothly in a blockchain network that requires thousands of SMC trustees or more. Besides, the throughput bottleneck limits the performance of the blockchain. Even though this might be easily solved by group decryption and reconstruction, this is another topic that deserves more analysis. Currently, the PVSS protocol introduces a huge ciphertext to be stored on the blockchain, which is costly. This is expected to be solved by using other protocols that could provide the same level of access structure flexibility. While we make detailed security analysis and discussion, some problems presented in the previous chapters only have prompted solutions but not real implementations, which we leave for future work.

![Figure 6.1: The traditional blind auction contract with commit-and-reveal.](image)

6.3 Future work

For future work, there are some directions to explore.
As discussed earlier, the scalability of F3B is not guaranteed, as the throughput, PVSS ciphertext size, and latency will keep increasing as the number of trustees increases. To solve this problem, we propose several future directions. One of them is to investigate sharding [79] and implement it into the F3B architecture to improve performance. There are also other advanced cryptographic primitives that suit the scenario, such as dynamic threshold public-key encryption [80]. Such protocol could offer access structure flexibility while having small ciphertext size, and could be integrated into F3B. Other issues discussed in Chapter 5 will also be further analyzed in the future.

![Diagram of the new proposed blind auction contract with F3B.](image)

**Figure 6.2**: The new proposed blind auction contract with F3B.

Specifically, we want to make use of the encryption mechanism of F3B and previous work such as Shutter and Fairblock, to create smart contracts that are easy to use. We take an auction smart contract for example: In a traditional blind auction contract, shown in Figure 6.1, the commit-and-reveal mechanism is used to guarantee that participants cannot see each others’ bids before the auction ends. Also to make sure that the winner are incentivized to behave honestly and pay for the auction, a mandatory deposit is required for the auction. These preliminaries make the whole process complicated. With threshold encryption such as protocols in F3B, a much simpler contract shown in Figure 6.2 can be built. In such case, one single transaction and no deposit is needed for completing the whole blind auction process, which significantly reduces the complexity. Even though in the current F3B, the auction time cannot exceed the fixed block confirmation time, integrating an identity-based encryption scheme can easily solve such concerns. We prototype the above two example smart contracts for comparison and leave a detailed design and
analysis for future work. This intuition could be extended to other smart contracts such as voting, gaming, and random number generating.
References


