DEDIS

Decentralized Distributed
Systems Laboratory

École Polytechnique Fédérale de Lausanne

Execution Layer Based Front-running Protection on Ethereum

by Shufan Wang

Master Thesis

Approved by the Examining Committee:

Prof. Dr. Bryan Ford
Thesis Advisor

Dr. Hubert Ritzdorf
External Expert

Haoqian Zhang
Thesis Supervisor

EPFL IC IINFCOM DEDIS
BC 210 (Bâtiment BC)
Station 14
CH-1015 Lausanne

January 5, 2023

To the dark forest of Web3 and decentralized security.

Acknowledgments

I would like to thank Haoqian for proposing the project and for his help in making the idea mature. I would also like to thank DEDIS, especially Prof. Bryan Ford for the inspiring feedback and the interesting lab meetings.

I would like to express my appreciation to the ChainSecurity team for their support and kindness during this hard period. I want to thank Dr. Hubert Ritzdorf for the time of supervision and thank Anton Permenev and Simon Perriard for the quick response to the questionable Go-Ethereum code snippet.

Great thanks to my family and my girlfriend who support me and look after me when I was sick over the year. Without you, I cannot continue the thesis with the illness.

Finally, I want to thank Prof. Mathias Payer for the thesis template and thank EPFL for the wonderful campus, utilities, and resources provided.

Lausanne, January 5, 2023 Shufan Wang

Abstract

Nowadays, front-running attacks have been prevalent in many blockchains such as Ethereum, where a miner can inspect and order the transactions in a block with one's preference to arbitrage. The profit of this attack, so-called MEV (Miner Extractable Value), has grown to a billion market.

Different approaches have been proposed to defend against front-running attacks. One promising approach is adding another consensus layer on the content of a block, which decentralizes the privilege of ordering transactions. However, this approach requires remarkable changes to the existing blockchains' consensus layer. Another ideal solution employs a two-phase commit-and-reveal scheme, where miners have to order transactions in a blind manner. This solution introduces remarkable latency as it requires multiple rounds of interactions between the user and the blockchain.

Recent research shows the possibility of one-round interaction solutions. F3B (Flash Freezing Flash Boys) is one of them, which is proposed as a low-latency, consensus-agnostic, commit-and-reveal scheme, where users only need to interact with the blockchain once thanks to the SMC (Secret Management Committee). In this project, we design and implement F3B-ETH, a front-running protected version of Ethereum client. In F3B-ETH, users can send encrypted transactions, which would be ordered in one block and decrypted for execution in a delayed block. At the ordering time, the transactions are encrypted, so the miners cannot know the content. At the execution time, the ordering of the transactions cannot be changed. Thus miners are unable to front-run.

We unveil the drawbacks of several similar commit-and-reveal solutions. We show F3B-ETH is a practical front-running protection design that achieves negligible latency compared to the current Ethereum and minimal extra cost.

Contents

Chapter 1

Introduction

Originating in 2018, DeFi (Decentralized Finance) has experienced explosive growth. Driven by blockchain and inspired by smart contracts, DeFi's open-source and non-custody property win itself wide adoptions. Nowadays, millions of transactions are happening on DeFi ecosystem every day. These transactions are signed and broadcasted by the users, ordered and executed by the miners, and finally included in the blockchain.

Transparent and fair as DeFi is, huge profits are extracted under the seemingly peaceful space. As miners have the priority to inspect and order the transactions in a way they want, they can front-run a transaction with a well-designed order to make a profit. So-called MEV (Miner Extractable Value) has been recognized as a controversial existence in DeFi. The cumulative extracted profit has reached 600 million dollars since 2020.

Several solutions have been studied to mitigate MEV. One promising approach is *Content by Consensus*: transactions and their order in a block are randomly shuffled by consensus. However, one more consensus layer would significantly worsen the efficiency of blockchains. Another direction is to blindly order based on a commit-and-reveal scheme, though requires over two rounds of user interactions.

Existing approaches are either non-trivial to implement or unfriendly to users. Fortunately, F3B (Flash Freezing Flash Boy) serves as a low-latency commit-and-reveal scheme that requires only one round of interaction. Depending on a secret management committee, the users are free to go after the commit, and the committee will be responsible for the reveal.

Theoretically, this solution is agnostic to the consensus layer. However, there is no integration of F3B on mainstream blockchains. To bring it to Ethereum, we implement F3B-ETH, a front-running resistant Ethereum client. We add a new encrypted transaction type, modify the transaction execution logic, and re-distribute the priority gas fee. We conclude that F3B is flexible to integrate with Ethereum with minor code modifications. The simulation test results show that F3B only adds a negligible latency and minimal cost.

Chapter 2

Background

In this section, We briefly introduce blockchains, smart contracts, and DeFi (Decentralized Finance), especially on Ethereum [21]. Then we present the front-running attacks, as well as some existing mitigations to the attacks.

2.1 Blockchains

Generally speaking, Blockchain is an immutable distributed ledger maintained by a group of participants (usually called nodes, validators, or miners) to record some data. Most of these blockchains are used to store and execute transactions. The earliest implementation dates back to Bitcoin [13], a blockchain that records the transactions of Bitcoins. Nowadays there are blockchains with various functionalities, such as Ethereum [21], Filecoin [11], and Ripple [17]. Despite the different protocols these blockchains have, they all target defending Sybil attacks [3] with different consensus algorithms such as PoW (Proof of Work) and PoS (Proof of Stake). Forging identities is unrealistic as PoW requires high computational power, and Ethereum PoS requires 36 ETH (around 40k USD at the price of writing) for each identity.

The state (data) of the blockchain will be updated by the nodes following the protocol. All the nodes shall have the same view towards the state of the blockchain, otherwise, we call it a fork.

Currently, Ethereum is the biggest blockchain that is permissionless, atomic, and supports smart contracts. Ether is the native cryptocurrency of Ethereum. Ethereum has shifted from PoW to PoS since September 2022. In this project, we focus on Ethereum as it is one of the most well-studied and mature ecosystems.

2.2 Smart Contracts

Smart contracts [15] are data stored in blockchains that represent executable programs. These programs model the contracts in reality. Smart contracts can be invoked by users or other smart contracts to modify the blockchain storage. There are some mechanisms to guarantee a deterministic execution of the invocation across all nodes, such as EVM(Ethereum Virtual Machine).

2.3 Transactions

In blockchains, a transaction is associated with several accounts, which may be an EOA (externally owned account) or a smart contract. Usually, some financial assets will be involved in a transaction and the blockchain storage will be modified. A typical example could be transferring Bitcoin or Ether between two accounts, the balance of these two accounts would be updated accordingly on chain.

The lifetime of a transaction starts when a user creates it and broadcasts it to miners of a blockchain. At this stage, we call that the transaction is pending and lying in the public mempool. Its lifetime ends when a miner picks it up and inserts it into a new block, representing the change of global state. As there is limited space for transactions in each block, not all the pending transactions can be put in the next block. It is obvious that the lifetime of a transaction largely depends on the miners. Therefore, users need to pay miners gas fees (for executing the transaction) and priority fees (for ordering the transaction as soon as possible). Though the execution of a transaction can be successful or reverted, the user always pays fees for the steps that the miner runs for this transaction.

A transaction is considered finalized when the block it resides in reaches finality. Finality means the state that contains the transaction becomes immutable. In blockchains with probabilistic finality, with some blocks of confirmation (i.e. several consecutive blocks), there is a negligible probability that the transaction can be reverted.

2.4 Decentralized Finance

DeFi represents the mainstream application of smart contracts in the past five years. It provides a self-custody financial instrument without relying on any centralized exchanges, banks and etc. AMM (Automated Market Maker) and lending protocol are two typical applications. AMM is an exchange protocol where users can exchange one token for another token. The exchange rate is automatically calculated based on a constant function and the current liquidity in the pool. The lending protocol is similar to what it is in reality, where users can deposit one token as collateral and borrow another token. In both applications, the users' funds are locked into the protocol and under the users' control. There are no intermediaries that control the funds, and everything follows the rules written in the smart contracts.

2.5 Front-running and Mitigations

Front-running derives from the fact that the contents of transactions are in plaintext, and the order of transactions is centralized on miners. When generating a new block, a miner has the privilege to inspect transactions contents in the public mempool, front-run a transaction, and order the transactions with its own preference. For example, if one makes a big swap (transaction TX) from asset A to asset B on an AMM, the price of asset A would fall after this transaction. Observing this, the miner can arbitrage by selling asset A before TX, and buying back asset A after TX.

Front-running is not necessarily regarded as an "attack" from the perspective of Flashbots [5]. Flashbots aim at mitigating the side effects of MEV by a permissionless and transparent architecture for everyone to extract the value. In flashbots auctions, users bid through a private channel to the validators for a preferred transaction bundle. The MEV opportunity goes to the user who wins the first-price sealed-bid auction. A recent study from Ben et al. [20] demonstrates the success of Flashbots that over 80% MEV extraction is achieved in this way and miners' profits doubled since the launch of Flashbots.

Despite mitigating side effects of front-running, different approaches have been proposed to eliminate the attack itself. Schwartz et al.[16] propose a novel direction that decentralizes the control of the block content by one more consensus layer, where the transactions in the mempool are randomly shuffled. Whereas, this approach requires remarkable changes in the base layer protocol, which is expensive to implement. Another direction is to hide the transaction contents from others. Besu hyperledger [7] solves front-running by private transactions which will only be seen by the parties involved. Different privacy groups exist in Besu where smart contracts can be created and executed. The state and transactions within this group are isolated from the outside and only visible inside. In this case, front-running attacks would not work as long as the attacker is not among the members involved.

Commit-and-reveal scheme based solutions [9, 12, 14] are also well-studied, where the miners blindly order the user-submitted encrypted transactions. Later, users reveal the plaintext transactions to miners for execution. Most commit-and-reveal designs involve more than two rounds of interactions between users and the underlying blockchain, which is inefficient and user-unfriendly. Submarine [19] is implemented in the application layer and agnostic to the underlying blockchain. However, the user's transaction for revealing may be temporarily excluded by the miner to deter the execution.

Some are implemented in the execution layer like Fairblock [12], Shutter Network [14], and

F3B [22]. Only one round of user interaction is required as all of these solutinos have an SMC (Secret Management Committee) that is responsible for revealing. A block key is used to encrypt all the transactions that target a certain block in Fairblock and Shutter Network. This greatly increases the throughput, however, introduces front-running again regarding block gas limit. Assuming 1000 encrypted transactions are targeted at block 1, where 200 are rejected for depleting the block gas limit. These excluded encrypted transactions' content would be necessarily leaked upon the decryption of those included ones because they are encrypted by the same block key.

F3B [22] sheds light on the possibility of a low-overhead commit-and-reveal scheme based solution in the execution layer. The SMC runs DKG (Distributed Key Generation) of a threshold encryption scheme periodically to generate a distributed secret key. Users will get the per-transaction-based key derived from the committee's distributed secret key to encrypt their transactions. Later, the miners will interact with the committee to decrypt and execute transactions in the previous block. As the key is per-transaction-based, transactions not included in the target block will not be decrypted and leaked. F3B's effectiveness has been shown by a prototype on Dela architecture. However, its compatibility and efficiency on a mature industrial blockchain remain unclear. Theoretically, F3B is agnostic to the consensus layer and can be applied to most blockchains.

Chapter 3

Design

3.1 F3B on Ethereum

The detailed F3B-ETH protocol is shown below. In order to distinguish two miners/validators who order and execute the transactions, we name them as 1)the sequencer who orders an encrypted transaction in block n and receives the gas tips. 2) the executor who executes the encrypted transaction in block $n + d$ where d is the block delay for the ordering block to reach finality. A system overview is provided in Fig. 3.1.

F3B Setup Phase: For each epoch, F3B committee follows TDH2 [18] cryptosystem with NIZK (Non-Interactive Zero Knowledge) proof to run a *DKG*, generating a public key pk_{smc} with shares of secret key sk_i. We assume the user can retrieve this public key in each epoch from F3B.

Life Time of an Encrypted Transaction: The interactions between users, Ethereum nodes, and F3B committee are shown below.

1. User A generates a random symmetric key k, and encrypts the transaction data with k: $c_{tx} = Enc_k[tx]$. Then A encrypts k with pk_{smc}: $c_k = Enc_{pk_{smc}}[k]$. A sends the encrypted transaction as (c_{tx}, c_k) to the Ethereum nodes.
2. The sequencer orders this encrypted transaction by committing it into a block.
3. After d blocks of delay, the executor retrieves the encrypted transactions from the previous block and interacts with F3B committee for decrypted shares with NIZK proof.
4. The executor verifies the correctness of decryption and executes the transactions.
5. Other Ethereum miners validate the block the executor proposed by verifying decryption and executing the decrypted transactions.

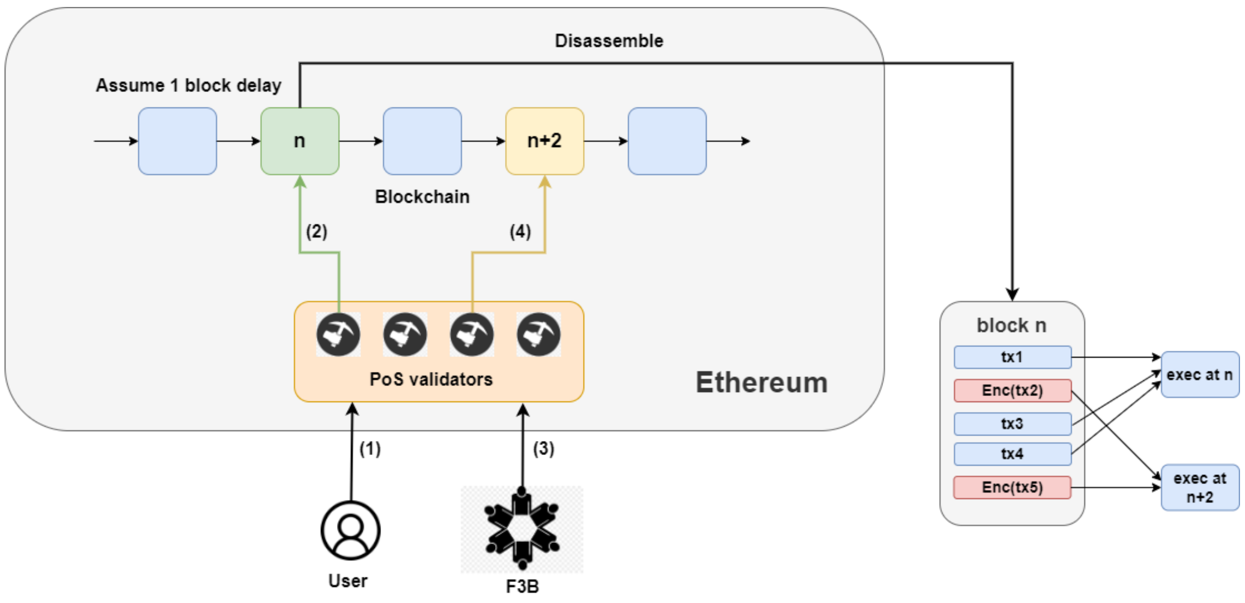

Figure 3.1: Overview of F3B-ETH. For simplicity we assume 1 block delay to reach finality. Both plaintext and encrypted transactions are enabled. (1) User sends an encrypted transaction. (2) Sequencer commits the encrypted transaction into block n. (3) Executor retrieves verifiable decryption from SMC. (4) Executor executes the decrypted transaction in block $n+2$.

3.2 F3B SMC Misbehaviors

The most obvious malicious scenario is that the SMC colludes to decrypt and front-run encrypted transactions. Another kind of misbehavior is blocking the threshold decryption by going offline. A malicious member may also provide a wrong decryption share to the executor. However, this member would be identified because of the NIZK proof. We suggest a mature staking and slashing protocol as Ethereum PoS to limit the possibility and profits of these attacks.

3.3 Executor Misbehaviors

A benign executor would follow the rules to execute a correctly decrypted transaction. However, a malicious executor may also hide the correct decrypted data and claim garbage data as a decryption result. In this case, the execution of this transaction would revert, and the executor can inspect the correct decrypted data and front-run it in this or a later block.

We implement F3B-ETH under the assumption that the SMC is always online so that all miners can verify the decrypted transaction by querying the secret management committee for the decrypted shares and proof. Whereas in a more practical situation, SMC would rotate and members may go offline. The data availability of shares and proof must be ensured for block validation and fraud-proof. We come up with two approaches for storing shares and proof.

1. **Data availability by Ethereum** The user attaches a hash of the symmetric key to the encrypted transaction. And the executor stores the decrypted symmetric key in the block header. In this case, a validator can check if the hash of the decrypted symmetric key matches the user-submitted hash. If so, the validator decrypts the data and verifies the state. Otherwise, the validator can fall back to interact with F3B SMC, obtain the correct decryption shares with proof and slash the executor who proposes the wrong key.
2. **Data availability by external infrastructure** The executor stores the verifiable shares with NIZK proof in third-party decentralized storage like Filecoin, which acts as an Oracle for decrypted shares and proof.

3.4 User Misbehaviors

Attacks can also be launched by a user. A user can use the wrong key to encrypt the transaction or simply send garbage transaction data. In both cases, the transaction can be ordered and would revert immediately upon execution in the delayed block. At a first glance, this behavior seems harmless, because sending a garbage transaction is allowed in Ethereum. Ethereum prevents this DoS (Denial of Service) attack by charging fees according to the length of transaction data as well as the finished execution steps.

After a deeper investigation into Ethereum, the viciousness of this delayed spamming is revealed. Before understanding the attack, let's look into Ethereum's execution layer first. A miner would follow the steps in Alg. 1 to fill a block. Apparently, for plaintext transactions, the miner will try to fill as many transactions as possible to maximize its tips. As the miner can execute the transaction at the same time, one would know how much block space each transaction will consume. If a plaintext transaction reverts immediately, the block space it consumes could be as small as zero.

However, things are totally different when it comes to F3B encrypted transactions. At the ordering phase, the sequencer is unable to execute the encrypted transaction, and thus reserves the block space equal to the transaction gas limit. Here comes the very problem that one can spam a whole block by sending garbage encrypted transactions. The sequencer being attacked would lose most of the tips, as it spends all its block limit on these non-executable transactions.

To mitigate this spamming attack, we design that in F3B-ETH, the unused gas would not be refunded to the users. Namely, a user shall carefully choose a transaction gas limit, as one would finally pay the whole limit. The sequencer hence would be incentivized to order encrypted transactions as it knows exactly how much it will get.

Algorithm 1 A miner fills transactions into a block on Ethereum

```
Input: An EVM. A public transaction mempool $\mathcal{D}$ where each transaction $d_i \in \mathcal{D}$ has a transaction gas limit $g_i$. Initial block gas limit $\mathcal{S}_0$. A coinbase $\mathcal{P}$ for collecting transaction tips. An empty block $\mathcal{B}$.
Output: A block contains transactions.
1:  while $\mathcal{D}! = \emptyset$ and $\mathcal{S}_i! = 0$ do
2:      Pick a transaction $d_i \in \mathcal{D}$ according to miner's preference.
3:      if $g_i > \mathcal{S}_i$ then
4:          Break: insufficient block gas space.
5:      else
6:          Purchase transaction gas limit $g_i$ from the sender's balance.
7:          Fill $d_i$ into the block by running $d_i$ on the EVM. Put this transaction $d_i$ into the block $\mathcal{B}$.
            Get the gas cost $c_i$ and miner tips $t_i$.
8:          Refund the unused gas back to the sender.
9:          Update the remaining block gas limit $\mathcal{S}_i = \mathcal{S}_{i-1} - c_i$ and add tips to coinbase's balance
            $Balance[\mathcal{P}] = Balance[\mathcal{P}] + t_i$.
10:     end if
11: end while
12: return $\mathcal{B}$
```

Chapter 4

Implementation

4.1 Delayed Execution

In Ethereum PoS, finality is reached when 2 epochs (32 slots, 384 seconds) in a row are justified [4]. Therefore, a 32 blocks delay is required for a committed transaction to be executed. Given the high volatility of cryptocurrency prices, trading on an AMM may always revert if the slippage tolerance is low. A long delay before execution aggravates this situation. Even though, it is believed that a slippage within 384 seconds incurs less loss compared to being front-run.

The implementation follows the design as described in Section. 3.1. The block validation is carried out under the assumption that all the SMC members are online to provide data availability for decrypted shares and proof. An analysis of two practical data availability solutions is discussed in Chapter. 5.

4.2 Encrypted Transaction Type

Given the remarkable delay, we recognize that some transactions require fast execution and could not necessarily be front-run. Thus, we decide to support the existing transaction types as well. Namely, in F3B-ETH, users can choose to send a plaintext transaction for faster execution with no front-running protection, or send an encrypted transaction for front-running protection. We follow EIP-2718 [23] to design a new encrypted transaction type and modify the execution logic accordingly.

Ideally, all the fields of an encrypted transaction should be encrypted. However, there are certain fields necessary for the ordering phase that should remain plaintext. For example, *gas base fee* and *gas priority fee* [4] are important for miners to select highly profitable transactions. A transaction is invalid if the *base fee* is lower than that defined in the block header. And a miner would not be incentivized to order an encrypted transaction if the tips paid are unknown. Besides, the *value* attached to a transaction should be clear, otherwise, it is impossible to check if the sender has enough balance to execute the transaction. Obviously, *chainID* should also be plaintext to resist replay attack. For simplicity, we only encrypt the message data of the transaction by a symmetric key. Following F3B, we add an extra field *Key* to the transaction, representing the symmetric key encrypted by pk_{smc} concatenated with a user-submitted hash. Most execution logic remains the same for encrypted transactions, except that a miner would retrieve the decrypted shares and proof first before executing it. After decryption, the executor would compare the decrypted key hash with the user-submitted key hash to identify potential user misbehavior.

4.3 Updated Transaction Receipt

One-to-one correspondence between a transaction and a receipt is strictly enforced in Ethereum, which raises the problem that we can not store both the ordering receipt and the execution receipt of one encrypted transaction. A miner cannot store a decrypted version of an encrypted transaction either, because the sender's signature of the decrypted transaction is missing. Hence, we implement it in a way that the former ordering receipt will be overwritten with the new execution receipt. As one can deduct the content of the ordering receipt from the execution receipt, an execution receipt suffices.

4.4 Interaction with Dela

We use a *dkgcli* of the F3B implementation on Dela blockchain to do the encryption and verifiable decryption. For every transaction, the miner would call verifiable decryption to *dkgcli*, which aggregates all the shares and proof and verifies the correctness.

Chapter 5

Evaluation

In this chapter, we present the time and communication overhead of F3B-ETH, compared with related works. All the experiments are conducted on a machine with Intel(R) Core(TM) i5-1035G1 CPU @ 1.00GHz single core and 8GB RAM. All the statistics analyzed in this section are based on the average gas price at *30 gwei*, average 160 transactions per block, block size at 75184 bytes, and ETH price at around $1200\$$ from YCHARTS [8] on Dec 19th, 2022. All the results are based on 10 trials.

5.1 Time Overhead

We measure the time overhead introduced by ordering and executing the encrypted transactions. In practice, ordering a transaction takes 0 in milliseconds. As shown in Fig. 5.1(a), we measure the execution time of encrypted transactions and plaintext transactions with different batch sizes. We set the SMC size as 3 across different batches. On average, an encrypted transaction takes 117.56 *ms* during the second phase, where 1.14% is used for execution and 98.86% are used for verifiable decryption with F3B. However, the decryption time can be regarded as a constant once the symmetric key length is set, and the execution time is dependent on the transaction itself. Compared to the application layer solution Submarine [19] that introduces 200% delay because of three transactions, our solution only introduces around 10% delay.

The time overhead with respect to SMC size is shown in Fig. 5.1(b) where the transaction batch size is 10. The communication overhead and decryption time overhead increases linearly with respect to the size of SMC. Basically, a 251.28ms delay would be induced per-transaction, which is higher than 79.65ms by identity-based encryption in Fairblock [12] on the same machine. However, as we mentioned before, using identity-based encryption to encrypt a whole block with one key would leak the contents of transactions that are not included in the target block.

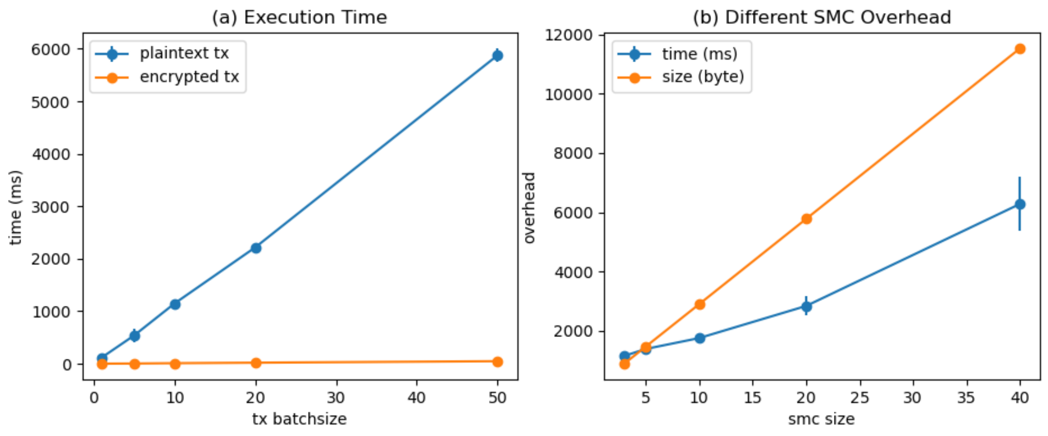

Figure 5.1: **Overhead of F3B-Ethereum**: **(a)** Time used for executing encrypted and plaintext transactions with different tx batch sizes. **(b)** Decryption time and shares size with different SMC sizes, where the batch size is set to 10 as a constant. For all experiments, we use the same transaction: sending 1 Ether from Account1 to Account2 with a greeting message attached. The x-axis of the figures is limited by the machine's performance.

Table 5.1: Storage cost for different extra fields stored in the block. The conversion between gas and USD is calculated based on Ether price at 1200\$. In the best case, the extra cost is negligible. In the worst case, 20\$ is used to store shares and proof.

Extra fields stored in block per tx						
	tx.key	tx.hash	shares&proof	decrypted key	best	worst
size(B)	32	32	35K	24	88	35.06K
fee (gas)	512	512	560K	384	1408	1051.8K
fee ($)	0.019	0.019	20.64	0.014	0.052	20.678

5.2 Storage Overhead

As shown in Fig. 5.1(b), the size of shares with proof increases linearly with respect to the number of members in SMC. According to the committee size of each epoch in Ethereum, we can assume there are 128 members in F3B SMC. Hence, each transaction would be associated with around 35.9KB for decrypted shares and proof if there is any misbehavior of the user or the executor.

The details of all the new fields possibly stored in a block are shown in Table. 5.1, where we assume the extra fields are priced the same as *msg.data* at 16 unit gas per byte. In a benign case, the extra cost for storing the 24-byte symmetric key is 0.052\$, which is trivial. In the worst case where there is misbehavior, an extra 21\$ per transaction is induced on Ethereum, which is acceptable for risky tradings. In general, the price for storage overhead is cheap.

Table 5.2: Comparison between several decentralized storage infrastructures.

Worst case storage overhead per tx				
	Ethereum	Filecoin	Crust	IPFS
cost($)	20.68	$8.62 \cdot 10^{-9}$	$4.70 \cdot 10^{-6}$	0
persistency	yes	yes	yes	no

The average block size on 19th Dec is 75184 bytes, so the decrypted keys would increase the block size by 6.81% in a benign case. However, once the executor or user misbehaves, other validators would interact with the SMC and store the shares and proof in the block header. An SMC with threshold 128 would induce $160 \cdot 35$KB shares and proof for 160 transactions, the extra storage would remarkably bloat a block by x100 times.

Third-party storage service is required if the shares are not stored in Ethereum due to high costs. IPFS [1] provides unlimited free storage, despite the fact that data persistency is not ensured. Filecoin [10] enables persistency storage by Proof-of-Storage where miners periodically submit proof of the committed storage. Crust [6] is another choice utilizing TEE (Trusted Execution Environment) to provide persistency. The comparison between several decentralized storage solutions is shown in Table. 5.2. In general, the cost of third-party storage services is negligible.

5.3 Communication Overhead

As shown in Fig. 5.1(b), the length of decryption shares and proof grows proportionally to the size of SMC. For a typical SMC with size 128, the size of shares and proof would be around 35 *KB* per transaction. Assuming N validators in F3B-ETH, the total communication overhead is $35 * N$ *KB* per transaction regardless of whether the shares and proof are stored in the Ethereum block or external infrastructures.

Chapter 6

Discussion

6.1 Limitations

As described in Section 4.1, there would be a significant delay (384 seconds) before an ordered encrypted transaction is executed. However, reaching finality is a strong guarantee. If a fork attack costs more than the profit of front-running, a shorter delay before finality may still be acceptable. It may be a good direction for future work that one can choose the delay of front-running protection so that efficiency and security could be both achieved.

Users are free from the reveal phase thanks to the SMC in F3B. Obviously, the SMC becomes an external security dependence of the underlying blockchain. If the committee refuses to decrypt the transactions, the blockchain will halt. If the committee colludes with the miners, the front-running attack is still feasible. Hence, a proper protocol and incentives should be designed to prevent the misbehavior of the committee. This protocol should take into consideration of committee's random selection, rotation, and slashing. We suggest the Ethereum PoS coordination layer as the base protocol of SMC, which is not in the scope of this project. The current F3B-ETH does not implement the storage of shares and proof into the block. Future work may consider this direction, which should also integrate the protocol of SMC and executor slashing.

There is a trade-off regarding the data availability of the decrypted shares and proof. If the shares and proof are stored in the block, the block would remarkably bloat x100 times and users have to pay this high gas fee. Besides, larger blocks increase the network delay associated with transaction data transmission and may induce security risks [2]. However, the data availability is hard to be guaranteed if the shares and proof are stored somewhere other than the block. Outsourcing data availability to decentralized storage infrastructures like Filecoin [10] would largely influence the security model of Ethereum. In summary, we need a cheap data availability solution without external security dependence. Rollups like Arbitrum and Optimism may be a better fit for industry-level front-running protection as they cut up to 95% cost and theoretically inherit Ethereum mainnet's security. A rollup bundles up the transactions on itself and finally packs the entire batch to Ethereum. Fraud proof or zero-knowledge proof is utilized to ensure the honesty of operators and the correctness of execution. Eventually, transactions on these rollups will be as safe as on the Ethereum mainnet, though there is still a long way for rollups to become mature by auditing their code and decentralizing their operators. Hence, a rollup implementation like Shutter Network could be a direction for future work.

Currently, the throughput of F3B is lower than the identity-based solutions such as Fairblock and Shutter Network. Using a block key would largely reduce the decryption time whereas posing the threat that excluded transactions' content may be leaked. The problem lies in the fact that both the included and the excluded transactions are encrypted with the same block key K_b. We observe that this problem can be solved if there is a distributed key-switch operation to re-encrypt the included transactions from the target block key K_b to another key K_b'. In this case, SMC would reconstruct K_b' to the executor upon execution. Hence, the executor can only decrypt those included encrypted transactions but not those excluded ones. We haven't found such an efficient encryption scheme that is distributed, identity-based, verifiable and supports re-encryption. The design of this encryption scheme and optimization of the throughput could be a promising direction.

Chapter 7

Conclusion

In conclusion, F3B-ETH is a practical front-running protection design that achieves negligible latency compared to the current Ethereum and is agnostic to the upper-level consensus layer. F3B-ETH eliminates the potential transaction leakage in Fairblocks [12] and Shutter network [14], and achieves 10% delay compared to 200% in Submarine [19]. The implementation itself proves the flexibility and composability of F3B on industrial blockchains.

Chapter 8

Installation Guidelines

8.1 Prerequisite

F3B-ETH is a modified version of the Go-Ethereum client, which is written in Golang. Please make sure to read the Go-Ethereum guidelines before using F3B-ETH. Preparation for F3B-ETH client and F3B SMC in Dela:

- Download the source code of F3B-ETH from F3B-ETH for running the Ethereum client.
- Download the F3B SMC from Dela for running the SMC.
- Build geth with: go install -v ./cmd/geth

8.2 Parameters

In the code base, we already defined several pre-funded user accounts and authority accounts for simulation. There are several default parameters that are changeable in the system in *core/types/transactions.go* as shown below. The *EncryptedBlockDelay* defines the block delay between the ordering block and the execution block. The *GBar* is a publicly known constant we directly take from *Dela dkg*. The *NodePath* specifies the path to the SMC node directory which would be used to launch the verifiable encryption. Make sure you adapt NodePath according to your machine.

```
const EncryptedBlockDelay uint64 = 2

const SymKeyLen = 24 //length of symmetric key to encrypt msg.data

const GBar string = "1d0194fdc2fa2ffcc041d3ff12045b73c86e4ff95ff662a5eee82abdf44a53c7"
```

const NodePath string = "D:/EPFL/master_thesis/dela/dkg/pedersen/dkgcli/tmp/node1/"

8.3 Single Geth Node Simulation

In this simulation, we run a single Geth node with PoA consensus for a quick example. There are no other validators for block validation.

- Run *Dela dkgcli* to start a committee with n members.
- Initialize the Geth node with: geth −−datadir .ethereum/ init clique.json
- Run one Geth node with: geth –nodiscover −−networkid 42 −−datadir .ethereum/ −−unlock 0x280F6B48E4d9aEe0Efdb04EeBe882023357f6434 −−mine
- Send 1 encrypted transaction to this node with: go run script/f3b-eth/main.go -encrypted -num 1
- Send 1 plaintext transaction to this node with: go run script/f3b-eth/main.go -num 1

8.4 Multiple Geth Nodes Simulation

In this simulation, we run two Geth nodes with PoA consensus, where two nodes would validate the blocks proposed by each other.

- Run *Dela dkgcli* to start a committee with n members.
- Initialize the Geth node1 with: geth −−datadir .ethereum1/ init ../multiclique.json
- Initialize the Geth node2 with: geth −−datadir .ethereum2/ init ../multiclique.json
- Run Geth node1 with: geth −−nodiscover −−networkid 43 −−datadir .ethereum1/ −−unlock 0x280F6B48E4d9aEe0Efdb04EeBe882023357f6434 −−mine −−ipcpath .1.ipc −−authrpc.port 8551 −−port 30303
- Run Geth node2 with: geth −−nodiscover −−networkid 43 −−datadir .ethereum2/ −−unlock 0xa9ca84343c8dB08d596400d35A7034027A5F4b31 −−mine −−ipcpath .2.ipc −−authrpc.port 8552 −−port 30304 −−miner.etherbase 0xa9ca84343c8dB08d596400d35A7034027A5F4b31 −−syncmode full
- Setup peers with: geth attach \\.\pipe\geth1.ipc, where we add the enode of another node by: admin.addPeer.

- Send 1 encrypted transaction to node1 with: go run script/f3b-enc/main.go -num 1 -id 1 -encrypted
- Send 1 plaintext transaction to node2 with: go run script/f3b-enc/main.go -num 1 -id 2
- Query the balance of the accounts in block 3 from node1 with: go run script/view-balance/main.go -id 1 -bn 3

8.5 Test

In the script above, we send a simple transaction, where a user sends 1 ether to the receiver, with an attached message *Merry Christmas*. One can run two validators and send encrypted or plaintext transactions to these validators. One can observe the logs of Geth nodes and query the balance of these accounts in different blocks to verify that:

- Both plaintext and encrypted transactions are correctly handled.
- Encrypted transactions are delayed and executed correctly.
- Blocks proposed by one node are correctly validated and accepted by the other node.
- The gas tips of encrypted transactions go to the sequencer but not the executor.
- There is no gas refund for encrypted transactions.

Bibliography

[1] Juan Benet. “IPFS - Content Addressed, Versioned, P2P File System”. In: (2014). URL: https://ipfs.io/ipfs/QmR7GSQM93Cx5eAg6a6yRzNde1FQv7uL6X1o4k7zrJa3LX/ipfs.draft3.pdf.

[2] Christian Decker and Roger Wattenhofer. “Information propagation in the bitcoin network”. In: *IEEE P2P 2013 Proceedings*. IEEE. 2013, pp. 1–10.

[3] John R Douceur. “The sybil attack”. In: *International workshop on peer-to-peer systems*. Springer. 2002, pp. 251–260.

[4] Ethereum.org. “Ethereum: Proof-of-Stake”. In: (2022). URL: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

[5] “Flashbots”. In: (2021). URL: https://docs.flashbots.net/.

[6] Decentralized Cloud Foundation. “Crust Network, Web3.0 Storage for the Metaverse”. In: (2022). URL: https://crust.network/.

[7] Hyperledger.org. “Hyperledger Besu Ethereum client”. In: (2022). URL: https://besu.hyperledger.org/.

[8] YChats Inc. “YCHARTS”. In: (2022). URL: https://ycharts.com/indicators/ethereum_average_block_size.

[9] Harry A Kalodner, Miles Carlsten, Paul M Ellenbogen, Joseph Bonneau, and Arvind Narayanan. “An Empirical Study of Namecoin and Lessons for Decentralized Namespace Design.” In: *WEIS*. Vol. 1. 1. 2015, pp. 1–23.

[10] Protocol Lab. “Filecoin: A Decentralized Storage Network”. In: (2017). URL: https://filecoin.io/filecoin.pdf.

[11] Protocol Labs. “Filecoin: A Decentralized Storage Network”. In: (2017). URL: https://filecoin.io/%20filecoin.pdf.

[12] Peyman Momeni. “Fairblock: Preventing blockchain front-running with minimal overheads”. MA thesis. University of Waterloo, 2022.

[13] Satoshi Nakamoto. “Bitcoin whitepaper”. In: *URL: https://bitcoin. org/bitcoin. pdf-(: 17.07. 2019)* (2008).

[14] Shutter Network. “Combating Front-running and Malicious MEV Using Threshold Cryptography”. In: (2021). URL: https://shutter.network/.

[15] Alexander Savelyev. “Contract law 2.0: ‘Smart’ contracts as the beginning of the end of classic contract law”. In: *Information & Communications Technology Law* 26.2 (2017), pp. 116–134.

[16] David Schwartz, Noah Youngs, Arthur Britto, et al. “Targeting Zero MEV - A Content Layer Solution”. In: (2021). URL: https://ethresear.ch/t/targeting-zero-mev-a-content-layer-solution/9224.

[17] David Schwartz, Noah Youngs, Arthur Britto, et al. “The ripple protocol consensus algorithm”. In: *Ripple Labs Inc White Paper* 5.8 (2014), p. 151.

[18] Victor Shoup and Rosario Gennaro. “Securing threshold cryptosystems against chosen ciphertext attack”. In: *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 1998, pp. 1–16.

[19] Submarine.org. “Submarine: Defeat Front-Running on Ethereum”. In: (2018). URL: https://libsubmarine.org/.

[20] Ben Weintraub, Christof Ferreira Torres, Cristina Nita-Rotaru, and Radu State. “A Flash(Bot) in the Pan: Measuring Maximal Extractable Value in Private Pools”. In: *Proceedings of the 22nd ACM Internet Measurement Conference*. IMC ’22. Nice, France: Association for Computing Machinery, 2022, pp. 458–471. ISBN: 9781450392594. DOI: 10.1145/3517745.3561448. URL: https://doi.org/10.1145/3517745.3561448.

[21] Gavin Wood et al. “Ethereum: A secure decentralised generalised transaction ledger”. In: *Ethereum project yellow paper* 151.2014 (2014), pp. 1–32.

[22] Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galinanes, and Bryan Ford. *F3B: A Low-Latency Commit-and-Reveal Architecture to Mitigate Blockchain Front-Running*. 2022. DOI: 10.48550/ARXIV.2205.08529. URL: https://arxiv.org/abs/2205.08529.

[23] Micah Zoltu. “EIP-2718: Typed Transaction Envelope”. In: (2020). URL: https://eips.ethereum.org/EIPS/eip-2718.