FairBlock: Preventing Blockchain Front-running with Minimal Overheads

by

Peyman Momeni

A thesis
presented to the University of Waterloo
in fulfillment of the
thesis requirement for the degree of
Master of Mathematics
in
Computer Science

Waterloo, Ontario, Canada, 2022

Author's Declaration

This thesis consists of material all of which I authored or co-authored: see Statement of Contributions included in the thesis. This is a true copy of the thesis, including any required final revisions, as accepted by my examiners.

I understand that my thesis may be made electronically available to the public.

Statement of Contributions

The smart-contract based instantiation discussed in section 4.3.1 is accepted as a paper in 18th EAI International Conference on Security and Privacy in Communication Networks. The accepted paper was written under supervision of Prof. Sergey Gorbunov, and Bohan Zhang has contributed in the implementation of smart contracts [74]. The implementation in the paper has been used as the basis of more sophisticated implementations in this thesis. Peyman Momeni was the sole author of the rest of this thesis and implementations.

Abstract

While blockchain systems are quickly gaining popularity, front-running remains a major obstacle to fair exchange. Front-running is a family of strategies in which a malicious party manipulates the order of transactions such that a transaction tx_2 which is broadcasted in time t_2 executes before the transaction of victim tx_1 which is broadcasted earlier in time t_1 ($t_1 < t_2$). In this thesis, we show how to apply Identity-Based Encryption (IBE) to prevent front-running with minimal bandwidth overheads. In our approach, to decrypt a block of N transactions, the number of messages sent across the network only grows linearly with the size of decrypting committees, S. That is, to decrypt a set of N transactions sequenced at a specific block, a committee only needs to exchange S decryption shares (independent of N). In comparison, previous solutions are based on threshold decryption schemes, where each transaction in a block must be decrypted separately by the committee, resulting in bandwidth overhead of $N \times S$. Along the way, we present a model for fair block processing, explore technical challenges, and build prototype implementations. We show that on a sample of 1000 messages with 1000 validators our work saves 42.53 MB of bandwidth which is 99.6% less compared with the standard threshold decryption paradigm.

Acknowledgements

I would like to thank Sergey Gorbunov, my supervisor. He has been a constant source of inspiration to me. This thesis would not have been possible without his technical vision, guidance, and patience. More importantly, I have learned from him to be more determined and brave to take steps toward my goals.

I am also grateful to my thesis readers: Alfred Menezes and Mohammad Hajiabadi for offering their time and invaluable perspectives on this thesis.

Dedication

This is dedicated to the one I love, my lovely family, and loved ones in PS752.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Maximal (or Miner) Extractable Value (MEV) is one of the central problems that prevent fairness [66, 110] and trust in decentralized exchanges and other decentralized applications (dApps) [37, 46, 112, 87, 9]. MEV allows a block proposer to influence the order of transactions to extract some "value" for themselves before they are executed by the application. By rearranging the order, the block proposer may inject extra transactions to extract profit. For example, if the block proposer sees a transaction Tx_{org} that tries to buy an asset from a decentralized exchange, it may include another transaction Tx_f (or a sequence of transactions) in the block that first buys the asset and then sells it to the sender in Tx_{org} for a higher fee.

MEV is informally defined as the revenue other than transaction fees and block rewards which can be extracted by reordering, censoring, and adding transactions in blocks [37, 46, 112, 87, 9]. MEV is present on any blockchain infrastructure that includes a party that is responsible for transaction ordering such as miners in Ethereum [105], validators in Cosmos [34], or sequencers in Layer 2 solutions such as roll-ups [78, 73]. Most of the extracted MEV happens in the form of a front-running attack whereby a party other than the block proposer itself closely observes the submitted transactions to the public mempool and exploits this information to detect profitable opportunities such as arbitrages, liquidations, and mispriced non-fungible tokens (NFT). After detecting them, the adversary makes sure that their profitable transaction will be executed in a high order by offering a high transaction fee to the block proposer or any party that is responsible for ordering. They do so by submitting it either in the public mempool or a private backchannel. Lower-

bound estimates show that sophisticated bots and their affiliated miners are making up to 5M USD in 24 hours with the total amount of over 607M USD million from 2020 to date just in the Ethereum network [73, 85, 105]. These attacks lead to serious problems such as high gas fees, network congestion, and even consensus instability [37].

Threshold decryption schemes are one of the most promising and well-known methods to prevent front-running [28, 49]. The idea was proposed in 1994 [86], and recently explored by blockchain projects such as Sikka, F3, and Anoma [49, 92, 2, 109]. In this approach, every transaction sent to the blockchain is first encrypted by the user using a global public key. A committee of decryptors (e.g. validators or set of users) holds shares of the corresponding private key. After a block of encrypted transactions is finalized and sequenced by the consensus layer, they collectively decrypt each transaction in a block to see its cleartext values. Subsequently, the transactions must be executed in the order in which they were finalized prior to the decryption. It is easy to see that this mechanism solves many forms of front-running attacks: the validators must finalize a block of encrypted transactions and fix their order, they cannot see the information in them, and hence it is much harder for them to influence the outcome. While this approach may be used to solve the problem, it introduces significant bandwidth overheads on the network. To be more specific, due to the high cost of distributed key generation process, decryption should happen without revealing private key shares. Consequently, for each encrypted transaction in a block, every committee member must propagate a separate decryption share, and a designated individual can aggregate decryption shares to reveal the transaction. For a N-transactions block and S-members committee, this results in decryption complexity of $N \times S$ broadcast messages. As an example, for $N = 1000$ transactions of size 64 bytes, $S = 1000$ of validators with a two-thirds honest majority, this adds an extra 42.7 MB of traffic on the network. This increases the bandwidth required to process transactions non-linearly resulting in significant scalability constraints. We refer the reader to Section 2.2 for limitations of other front-running prevention mechanisms.

1.2 Our contributions

In this thesis, we construct a front-running protection protocol with minimal bandwidth overheads – linear in the number of users or validators called keepers. Our construction, called FairBlock, is based on well-studied cryptographic assumptions. In particular, the scheme is based on identity-based encryption where one can exploit the linearity and secret sharing of the IBE private keys [16, 90, 31]. In FairBlock, a committee composed of keepers that run a distributed key generation (DKG) [82, 53] protocol to generate a shared master

key msk associated with a system-wide master public key mpk for an IBE scheme. Next, we associate each block identifier h with an IBE "identity". Consequently, clients can commit to their transactions by encrypting their information with mpk and identity for a future block h (or a range of blocks). Validators run the consensus and sequence all encrypted transactions in a block. Finally, to decrypt the block with minimal overheads, each keeper k (a) computes a share b_h^k of the private key b_h (named block key) for the IBE identity corresponding to block h, and (b) broadcasts it over the blockchain. After sufficiently many keepers propagated their shares b_h^k, anyone can perform the key extraction process to obtain the private key b_h that allows decryption of all transactions encrypted under identity h with no further communication. In FairBlock, another set consists of users or validators named "relayers" (which can overlap with keepers) is responsible for key extraction and decryption. The original sender of the transaction can also reveal the plaintext transaction without block key extraction and decryption to avoid paying fees.

FairBlock is a general solution that can be applied to all blockchain networks and dApps. The scheme is practical and can be applied in real systems as IBE constructions that support the linearity properties that we leverage are efficient. FairBlock does not have basic commit-reveal challenges, which can facilitate denial of service attacks, whereby a client commits to a transaction and reveal it later only if subsequent transactions make it profitable [46, 28].

Compared to the solutions [67, 30, 44, 100] that leverage time-lock puzzles [14], we do not introduce significant delays or high computational complexity in decryption. Moreover, our work does not rely on secure enclaves [107] to realize a private pool [94]. Unlike the standard threshold decryption approach, FairBlock bandwidth overhead is minimal as the number of messages in this system grows linearly with the number of keepers.

1.3 Overview of thesis

The remainder of the thesis is organized as follows. In Section 2, we take a tour of the world of decentralized applications and familiarize the reader with the front-running problem, and limitations of the proposed countermeasures in related works. In Section 3, we review the cryptographic preliminaries of our work. In Section 4, we propose several designs to mitigate blockchain front-running in different settings and explore practical technical considerations in detail. Section 5 describes our prototype implementation and evaluation. We indicate future research directions and challenges in Section 6, before concluding in Section 7.

Chapter 2

Background and Related Work

2.1 Front-running and MEV

2.1.1 Problem definition

In this section, we formally define Miner Extractable Value (MEV) and front-running attacks [78, 37], and discuss some of their negative consequences. MEV was first coined by Daian et al. [37] and informally defined as the value which can be extracted by a miner by manipulating the order of transactions in a block. However, the term miner is specific to proof-of-work (PoW) blockchains and the term should be broadened to sequencers to address other blockchain architectures such as proof-of-stake (PoS) blockchains, Layer two solutions, and cross-chain protocols. Moreover, while the sequencer has the power to reorder or censor transactions, other players such as traders and bots are making most of the profits from MEV opportunities [73] by cooperating with sequencers. To avoid this confusion and broadening the scope of MEV, the definition has evolved to Maximal Extractable Value in the blockchain community [73, 78]. Later works [64, 78] have attempted to formalize and extend the MEV definition to cover more general settings including cross-domain space and a diverse set of players. Despite the popularity of MEV in the blockchain community, the term has not been used consistently in recent works [37, 78, 64, 6] and no precise definition has been proposed that can accurately describe and cover all the different forms of activities that are known as MEV. Although the definition of MEV will continue to evolve, to the extent of our knowledge, Obadia et al. [78] have formalized the most precise definition of MEV to date and we briefly familiarize the reader with this definition.

The authors of [78] define extractable value for a player P in a domain i as the difference

in its balance b_i after execution of a sequence of actions $a_1, ..., a_n$ on an initial state s to reach the new state s':

$$ev_i(P, s, a) = b_i(s', P) - b_i(s, P), \tag{2.1}$$

where domain is defined as follows.

Definition 1 *A domain is a self-contained system with a globally shared state. This state is mutated by various players through actions (often referred to as transactions), that execute in a shared execution environment's semantics.*

Based on the definition of extractable value and domain, MEV is defined and formulated as follows:

Definition 2 *Maximal Extractable Value (MEV) is the maximal value extractable between one or more blocks, given any arbitrary re-ordering, insertion or censorship of pending or existing transactions.*

$$mev_i^j(P, s) = \max_{a \in A_j}(ev_i(P, s, a)), \tag{2.2}$$

where balance changes occur in domain i, actions are taken in domain j, and A_j is the action space in domain j. Further, we refer the reader to [78] for the extended format of this definition for the cross-domain maximal extractable value of several domains.

Having defined what is meant by MEV, we will now move on to define blockchain front-running as the most notorious strategy for making profit in the MEV space. In this thesis, we define blockchain front-running as follows:

Definition 3 *Blockchain front-running is a family of strategies in which a malicious party directly or indirectly manipulates the order of transactions in a blockchain architecture such that a transaction tx_2 which is broadcasted in time t_2 executes before the transaction of victim tx_1 which is broadcasted in time t_1 where $t_1 < t_2$.*

In practice, front-runners may be the parties who are responsible for sequencing transactions themselves including miners, validators, roll-up providers, or relayers. Alternatively, front-runners may indirectly influence the order of transaction by offering high tips (gas price) to block proposers, performing attacks in the network layer such as DDOS attacks, or utilizing high-speed networks similar to high-frequency traders in traditional financial markets. Typically, front-runners such as sophisticated bots actively listen to pending

transactions in the public mempool or in the peer-to-peer network to exploit the revealed (but not executed) information of transactions to make profits by broadcasting a transaction and front-running the victim's transaction to capture the opportunity. This form of front-running attacks significantly increases the cost of transaction fees for normal users, unfairly steal many profitable opportunities, and makes the user experience much more complex and slow by failing the victim's transaction. Front-running and MEV-related transactions can also result in significant network congestion. For instance, Bank for International Settlements [3] has reported that up to one out of thirty transactions in Ethereum blocks from 2020 to 2022 were included for MEV extraction purposes. Moreover, several works in the literature [78, 46, 37] have also discussed the potential threat of front-running attacks to the consensus mechanism of blockchain networks due to the high profitability of these opportunities which incentivize some players such as miners to sabotage the whole network.

2.1.2 Front-running strategies

In this section, we discuss two families of the most common front-running strategies with the goal of familiarizing the reader with the MEV space and nature of the front-running attacks.

Sandwiching attack

Sandwich attacks are the most notorious form of front-running attacks. Predatory parties observe profitable pending transactions in the public mempool or exploit their privileged access to plaintext orders in centralized exchanges or relayer services. At its core, they manipulate the transaction ordering in a block and ensure that their front-running transaction tx_1 executes before the victim's transaction tx_{org} and their back-running transaction tx_2 executes immediately after the victim's transaction. The profitability of this strategy is based on the assumption that demand for assets results in a higher price. In simple terms, when the attacker observes a pending buy order, it can buy the same asset before the original trade, and immediately sell after execution of the original trade to enjoy price increases thanks to a) its back-running transaction and b) the victim's transaction. For a concrete example, assume the scenario that Alice broadcasts tx_{org} to trade 100 USDC for DAI with a standard 0.3% transaction fee and 1% slippage tolerance in a decentralized exchange (DEX) that has 1000 DAI and 1000 USDC reserve. Following the standard automatic market maker (AMM) model [1] in DEXes, Alice is expecting to receive 90.66 DAI in return. However, Bob observes this trade in the mempool and front-runs Alice by

submitting t_1 to trade 5.23 USDC for 5.19 DAI which increases the price of DAI to the maximum limit that Alice can tolerate due to 1% slippage. Consequently, Alice's trade t_{org} returns 1% less DAI (89.75 DAI) and even further increases DAI price. Finally, Bob pockets 1.05 USDC (ignoring gas fees) in profit by submitting t_2 and trading its 5.19 DAI for 6.28 USDC. To realize this strategy Bob should manipulate the ordering by offering gas prices (price for computing each unit of computation) to block proposers such that t_1 and t_2 sandwich t_{org}. Block proposers normally sort transactions with respect to gas price; and for a successful attack, Bob has the challenge to strategically offer a gas price that overbids competitors and still be profitable which makes this strategy complex for Bob. However, Flashbots [37] allows front-runners to sandwich users with much less risk as they can offer a bundle of transactions containing t_1, t_2, t_{org}, and a bid directly to the block proposer without submitting it to the mempool. Then the block proposer chooses the most profitable bundles and executes them in their profitable order. Consequently, Bob can almost guarantee his profit by only paying for the bid and fees only if the block proposer executes t_1, t_2, t_{org} in the specified order.

Generalized front-running

Blockchain networks such as Ethereum [105] and Avalanche [4] are modelled as a distributed state machine and their global state changes from block to block with respect to a pre-defined set of rules. This means that any party can observe a pending transaction tx_{org} and simulate its resulting state change. Consequently, generalized front-runners can simulate all pending transactions and determine the profitability of them by checking the balances of the transactions' senders. In case of a net increase in the original sender's balance, the generalized front-runner copies the same transaction fields and signs it with its private key. Next, it simulates the copied transaction locally to check that the transaction is indeed profitable e.g. not a trap smart contract. Finally, the generalized front-runner submits transaction tx_1 to front-run tx_{org} and capture the profit. This strategy enables parties that have access to the mempool to extract profits by mimicking a pending transaction (even blindly) and outbidding competitors and the original sender. While the generalized front-runner may be able to simulate all pending transactions in order to find the most profitable ones, due to the high number of pending transactions and cost of simulating, the front-runner can also filter specific target addresses and markets which is expected to have more profitable opportunities including NFT markets, DEX and CEX liquidity pools, yield aggregators, or well-known traders.

2.2 Related works

Several academic works and projects have attempted to either limit or prevent front-running. For instance, Flashbots [37] has mitigated front-running and bidding war consequences such as high gas fees and network congestion with a private channel for front-runners to make bids directly to miners through relayers. However, relayers and white-listed miners in this approach have full access to the transaction content in clear which makes it prone to front-running and censorship. LibSubmarine [20] conceals the transaction among other similar transactions by locking the amount of the transaction to a generated address that is indistinguishable from an address that has not been used on Ethereum previously. However, the security of this solution is not based on strong cryptographic assumptions, and also the contents of the transactions are still in plaintext and prone to front-running.

DEXes and AMMs [1, 95], as the main target of front-running [73, 9] have tried to limit front-running consequences such as transactions failure and gas waste using slippage. This approach has interestingly led to near-guaranteed sandwich attacks by taking a deal and selling it again to the buyer with a higher price to the maximum extent that slippage allows. CowSwap [35] protects DEX users from sandwich attacks by matching simultaneous users off-chain, whenever a user is buying an asset and another is selling the same asset. Currently, this approach is limited as it cannot prevent general front-running on transactions in the public mempool. Moreover, a trader can still front-run orders in CoWSwap by making them fail, doing the same trade resulting in higher prices, and selling it back with the assumption that the failed transactions will be resubmitted with a high probability.

Recent projects including Secret Network [76] and Fairy [94] leverage secure enclaves namely Intel SGX [107] to build private mempools at the cost of potential latency, storage limits, and security risks due to several successful recent attacks on secure enclaves [99, 98]. The basic commit-reveal approach relies on clients to reveal their transactions after the finalization of the commitment phase which leads to connectivity issues, and denial-of-service attacks (selective revealing based on the market output). As a way to address basic commit-reveal issues, time-lock encryption [44, 30, 67, 84] relies on the secure implementation of verifiable delay functions (VDF) [14] and time-lock puzzles at the expense of long delays between transaction inclusion and execution e.g. 3 or 7 minutes delay in VeedDo implementation by Starkware [100].

Shutter Network [91] leverages threshold decryption and distributed key generation as their tools to prevent front-running by generating a private key for each epoch but additional research is needed to validate their cryptographic protocols. Projects such as Ferveo [49], Sikka [92], Helix [2], and F3 [109] employ threshold decryption with high

communication overhead as decryption of every single message requires all members of the decryption committee to send their partial decryption shares.

Finally, recent works [11, 29, 55] have proposed multi-party computation [111] to mitigate front-running attacks. We discuss their current drawbacks as a promising approach for future work in Section 6.

Chapter 3

Cryptographic Preliminaries

3.1 Pairing-based cryptography

Let G_1 and G_2 be cyclic additive groups of prime order q, and G_T a cyclic multiplicative group of the same order. Assume that $g_1 \in G_1$ and $g_2 \in G_2$ are generators of their respective groups. A bilinear pairing is a map $\hat{e} : G_1 \times G_2 \rightarrow G_T$ with the following properties for all $a, b \in Z_q^*$:

- **Bilinearity:** $\hat{e}(g_1^a, g_2^b) = \hat{e}(g_1, g_2)^{ab}$.
- **Non-degeneracy:** $\hat{e}(g_1, g_2) \neq 1$.
- **Computability:** There is an efficient algorithm to compute $\hat{e}(g_1, g_2)$.

The security of the identity-based encryption in this work is based on groups in which any probabilistic polynomial-time solution to the decisional Bilinear Diffie-Hellman problem is only negligibly better than a wild guess or the computational problem is assumed to be hard [83]. Other identity-based schemes can be used to instantiate the construction, assuming they describe properties we outline in Section 3.2.

- *Computational Bilinear Diffie-Hellman (CBDH):*
  Given $g \in G_1$, and also g^a, g^b, g^c with unknown $a, b, c \in Z_q$, find $\hat{e}(g, g)^{abc}$.
- *Decisional Bilinear Diffie-Hellman (DBDH):* Given $g \in G_1$, and also g^a, g^b, g^c with unknown a, b, c and a challenge $x \in G_T$, decide whether $x = \hat{e}(g, g)^{abc}$.

3.2 Identity-based encryption

In classic public-key encryption, a secure communication channel can be built using a public-private key pair. Senders should encrypt their messages with the receiver's public key. Subsequently, the receiver can decrypt the message with the corresponding secret key. However, assuring that the used public key belongs to the receiver party is crucial to the security of the system, as an adversary can potentially make the sender party encrypt with another public key. In practice, certifying authorities (CA) generate certificates for public keys which is a digital signature on the receiver's public key and its identity. As a result, the sender party can simply verify the authenticity of the receiver's public key with the public key of the CA. In reality, establishing and managing a Public Key Infrastructure (PKI) is very complex and expensive [71, 83, 16].

In 1984, Shamir [90] addressed this problem by proposing the concept of identity-based cryptography. In identity-based cryptography, identifier information can be used to generate public keys for encryption or signature verification. More specifically, a trusted third-party (TTP) also known as private key generator (PKG) can extract private keys for any user Alice with its own private key and her identifier information which should be securely sent to her by request. Another user Bob can easily encrypt messages or verify signatures by only using Alice's identifier information and the trusted third-party public key [71, 83, 16].

One interesting aspect of identity-based encryption is that Bob can include certain conditions while encrypting the message using Alice's identifier information. For example, different kinds of policies can be enforced by including dates, scores, and any other conditions along with Alice's ID while encrypting. For instance, Bob can encrypt a message for Alice that can be expired after a certain date since Alice cannot receive its private key from the trusted third party after that date. This property is especially very useful to revoke private keys in situations where the private key is compromised or the owner leaves the system. Many other real-world applications can be built based on this property. As described later in this work, we have exploited this advantage to encrypt messages (transactions) that can be decrypted only after a certain deadline.

The open problem of constructing an identity-based encryption (IBE) scheme was only solved in 2001 by Boneh and Franklin [16] and Cocks [31] independently. Later theoretical works and implementations have been built on top of Boneh and Franklin's scheme because of the inefficient length of ciphertexts in the Cock's scheme [83]. An identity-based encryption (IBE) [16, 31, 90] allows establishing a global master key in the system that can be used to derive identity-specific public keys (and associated private keys). For instance, it enables a sender, Alice, to encrypt a message for receiver Bob using his identifier

information such as email address, phone number, or IP address. The receiver Bob, having obtained a private key associated with his identity information from the trusted third-party (TTP), can decrypt the ciphertext [71].

An IBE scheme consists of a tuple of algorithms: $Setup$, $Extract$, $Encrypt$, and $Decrypt$ satisfying the following semantics:

- $Setup(1^\lambda)$**:** On input corresponding to the security parameter λ, the setup algorithm outputs a master key msk and its associated master public key mpk which is publicly known.

- $Encrypt(mpk, ID, m)$**:** On input of the master public key mpk, an identity ID and a message m, the encryption algorithm outputs a ciphertext C.

- $Extract(ID, msk)$**:** On input of the master key msk and identity ID, the extraction algorithm returns a private key d_{ID} for user with identity ID.

- $Decrypt(d_{ID}, C)$**:** On input of the private key d_{ID} and ciphertext C, the decryption algorithm recovers the plaintext message m.

For correctness of the IBE scheme, we require that for all pairs of (mpk, msk) generated by $Setup$ and all identities ID:

$$\Pr[Decrypt(Extract(ID, msk), Encrypt(mpk, ID, m)) = m] = 1 \tag{3.1}$$

We build FairBlock using an IBE that is semantically secure under the BDH assumption in a random-oracle model [16]. In particular, we use the Boneh-Franklin IBE [16]. Our construction will use an IBE in a non-black box way and exploit two common properties. Other IBE schemes [69, 15] may also be used assuming they satisfy these two properties:

1. Support efficient distributed key generation (DKG) protocols.
2. Support linear homomorphic operations over the private keys for identities. That is, given a share of a master key, one should be able to compute a share of the corresponding private key for any identity ID, such that given a collection of shares, anyone can extract the private key for ID.

The central TTP in the described IBE algorithms is a single point of failure and contradictory to the distributed nature of blockchains. As suggested in [16], Shamir's secret sharing (SSS) [89] technique can replace the TTP by distributing the shares of msk among a group of keepers with an honest majority. In Section 4.4, we further show how to employ a distributed key generation [82] in our work to eliminate the trusted dealer in SSS to achieve complete decentralization.

3.3 Hash function

In our protocol design, we rely on an efficiently computable cryptographic hash function that map arbitrarily strings to binary strings of fixed length with the collision-resistance property. Furthermore, we have assumed that the hash function offers sufficient security guarantees for this application to be modelled as a random oracle as often considered in the literature.

A family H of hash functions, parameterized by a security parameter k, is defined as collision resistant if it is hard to find two different strings x_1 and x_2 such that $H(x_1) = H(x_2)$.

3.4 Commitment scheme

In order to ensure that transactions cannot be modified, censored, or added after decryption by relayers, our protocol should verify that decrypted transactions are indeed the ones that have been encrypted. To realize this, we have leveraged a basic non-interactive hash-based commitment [19, 75, 60] with computational binding and hiding properties in the random oracle model based on a collision-resistant hash function H. The hiding property is vital for our commitment scheme, as an adversary should not acquire any information about the transaction. We also need binding, so a relayer cannot submit a different transaction with the correct commitment to censor the original transaction. The simple and efficient hash-based cryptographic commitment scheme in this work can be replaced with more advanced commitment schemes [81, 60, 59] with stronger security guarantees.

Chapter 4

Protocol Design

In this chapter, we present three architectures to show how we can integrate FairBlock in different scenarios in real-world distributed applications. In Section 4.3.1, we aim to build a system that can be integrated into legacy blockchains with minimal modifications using smart contracts. In Section 4.3.2, we design a system as a service using an oracle network to prevent front-running attacks which is more cost-beneficial, faster, and secure. In Section 4.3.3, we show how to apply FairBlock in the consensus layer of the underlying blockchain to eliminate composability and user experience challenges with previous approaches. In Section 4.4, we explore distributed key generation in more detail which is the first step of all designs in this work. Following proving correctness and security in Section 4.5, we introduce some of the practical considerations and challenges in our designs in Section 4.6. Later in Section 6, we further propose applying the same FairBlock technique in emerging blockchain infrastructures; particularly in roll-ups and cross-chain ecosystem.

4.1 Design summary

FairBlock is a technique that allows eliminating bandwidth overheads for threshold-based commit-and-reveal schemes. In summary, FairBlock is composed of four simple steps:

1. Users encrypt their transaction.
2. Validators sequence encrypted transactions.
3. A threshold committee decrypts transactions in a block.

4. Transactions execution follows the fixed sequence.

Since the sequencing is performed over encrypted transactions, no party can learn any information about them, and inject or reorder them efficiently to perform front-running attacks. Using identity-based encryption, given a master key shared across all validators, any user can derive a round-specific public key and use it to encrypt their transactions for that round. Validators simply need to propagate their shares of the private key for each block in the clear, and subsequently, anyone can re-derive the secret key for the round and decrypt all transactions with no further communication.

4.2 Model

4.2.1 Players

In this protocol, we define three types of players:

- **Users:** Parties who wish to communicate with a target smart contract without being front-runned. Users submit a transaction containing their encrypted message e.g. trading information to our system.
- **Keepers:** Parties that are responsible for generating a distributed secret key and submitting their shares for each block key. Keepers set can be composed of any parties in the network including users, consensus validators, decentralized oracle networks (DON) [28], or decentralized autonomous organizations (DAO) [38].
- **Relayers:** Parties that are responsible for aggregating block key shares, computing block keys, and decrypting committed transactions. Relayers set can be composed of users, keepers, consensus validators, decentralized oracle networks (DON) [28], or decentralized autonomous organizations (DAO) [38]. In practice, keepers can also play the relayers' role; however, we have defined an independent set to highlight the fact that they can be a very large group competing to decrypt transactions. Also, even just a single honest party e.g. the next block proposer would suffice.

4.2.2 Setup

In our protocol, a set of n keepers $P = \{P_1, P_2, ..., P_n\}$ generate a shared master key msk and a system-wide public key mpk. Users pick a desired block identifier h as the ID of the

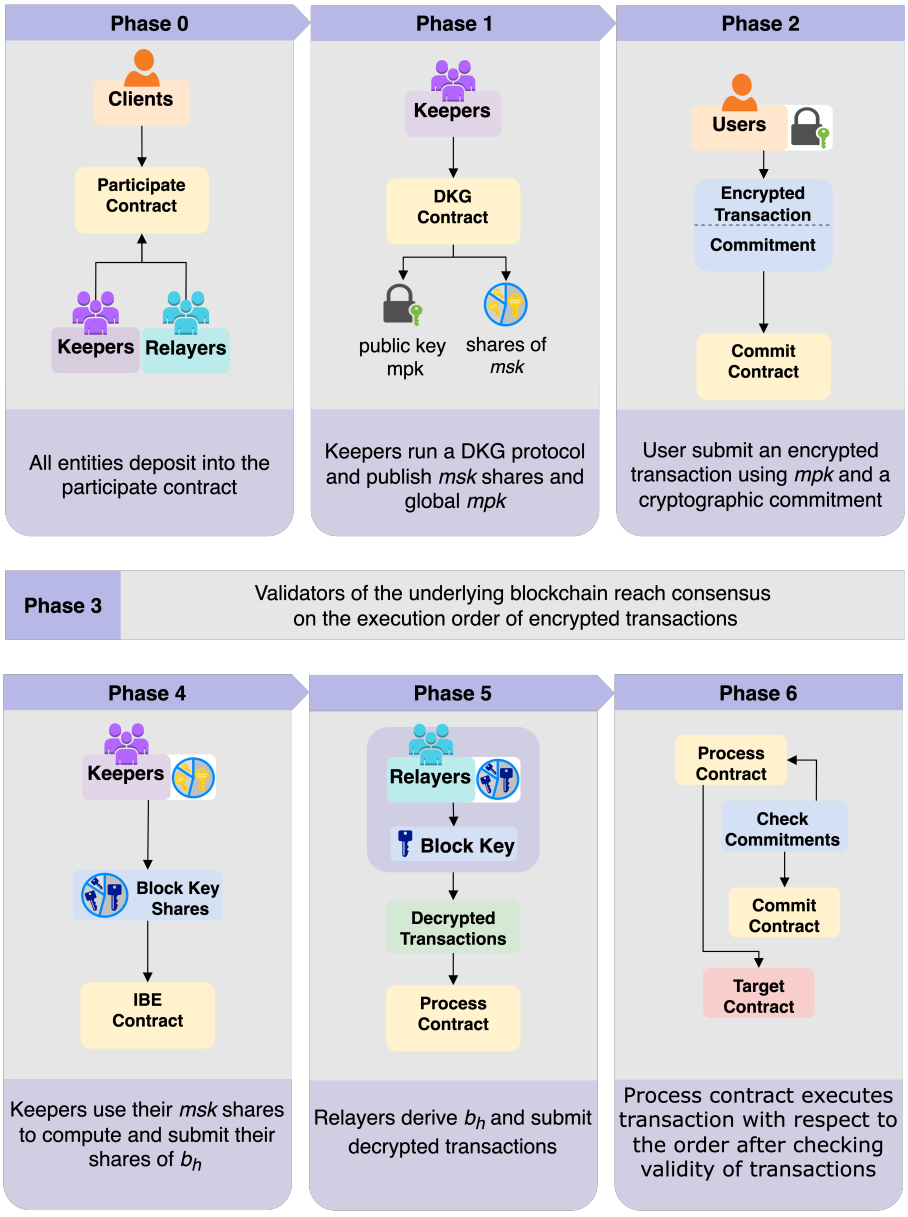

Figure 4.1: Architecture of FairBlock using smart contracts

block (or range of blocks) in which their encrypted transaction should be executed without being front-runned.

Assume that the associated groups of a symmetric bilinear pairing, $\mathbb{G}_1$ and $\mathbb{G}_T$ have order q, that is, the pairing is $\hat{e}$: $\mathbb{G}_1 \times \mathbb{G}_1 \to \mathbb{G}_T$. Two cryptographic hash functions H_1 and H_2 are also used. H_1 maps block identifier $h \in \{0,1\}^*$ to $\mathbb{G}_1$, and H_2 maps $\mathbb{G}_T$ to transaction information t_x of bitlength l_1. Additionally, a collision-resistant hash function H_c is used for the cryptographic commitment. Also assume that a generator $g \in \mathbb{G}_1$ is available to all entities.

4.2.3 Threat model

We assume that the adversary is computationally bounded and our cryptographic schemes including IBE, DKG, and Commitments are secure. In this work, we work with an honest majority assumption on the keepers. That is, an adversary controls at most t keepers, whereas a collaboration of $t+1$ keepers is required to extract the block key and also the presence of at least one honest relayer is necessary to perform decryption. Assuming that keepers are running Pedersen's DKG [82] protocol, the adversary must control at most $t \leq \frac{n-1}{2}$ keepers. In the case of consensus-level implementation, the underlying BFT-style [21] consensus algorithm may enforce a two-thirds honest majority assumption. In this case, the adversary must control at most $t \leq \frac{n-1}{3}$ keepers' shares as consensus validators also play keepers' roles. A party controlled by the adversary may deviate arbitrarily from the specified protocol. We consider an adaptive adversary, in the sense that it can decide which parties to corrupt at any point during the protocol execution.

4.2.4 Correctness

Our construction should also satisfy correctness. We define this property as follows: Given a sequence of encrypted transactions submitted by the users, every player should be able to learn the cleartext transactions and their correct execution order after the block key reconstruction phase.

4.2.5 Security model

We now describe a security model that captures the notion of fairness. In essence, it states that no adversary that controls less than the corruption threshold of parties can influence the order or censor transactions in the system.

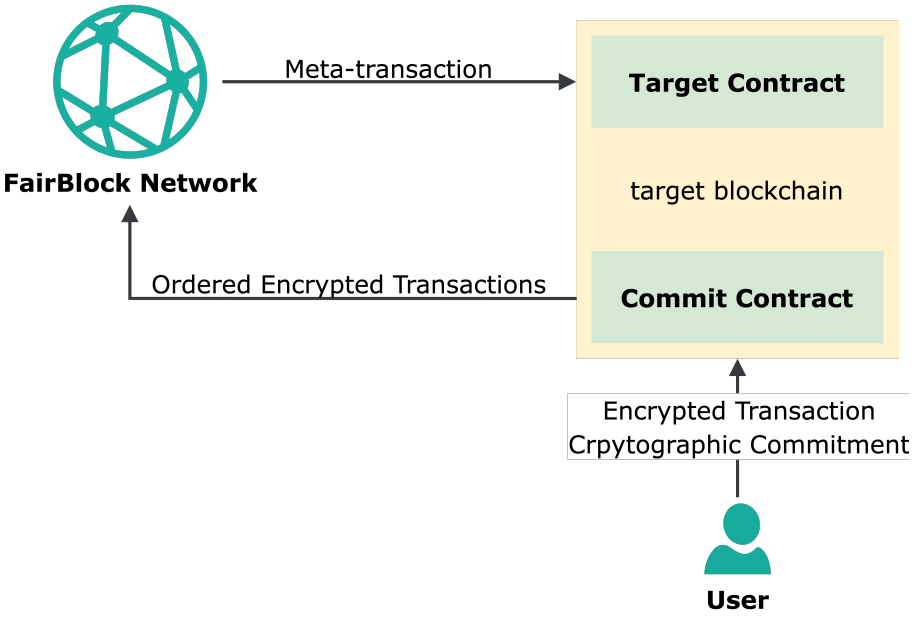

Figure 4.2: Simplified architecture of oracle-based instantiation

We follow the formal notion of fairness in recent works [66, 28, 110] and aim to provide fairness by satisfying both order-fairness and secure causality preservation [45, 26]. Order-fairness requires that if a large fraction of nodes γ receive T_1 before T_2, then T_1 should not be executed after T_2. We refer the reader to [66, 28, 110] for further formal discussion and technical detail of order-fairness. Also, the security conditions of secure causality-preservation [45, 26] require formally that no information about a transaction becomes known before the finalization of its order in the block. Until that time, the system must not reveal any information to an adversary in a cryptographically strong sense. In Section 4.5.2, we show how FairBlock satisfies both secure causality-preservation and order-fairness.

4.3 FairBlock

4.3.1 Smart-contract based instantiation

In this section, we show how to apply FairBlock to any dApp by adding special-purpose smart contracts to the system. In this instantiation, the target application will need to slightly modify its logic to leverage these contracts. Figure 4.1 illustrates the high-

level workflow of the smart-contract based design. To implement FairBlock using smart contracts as the communication layer, we introduce five smart contracts:

1. $\mathsf{Participate}(\mathsf{dep}, \mathsf{val_m})$: This contract keeps track of keeper and relayers sets. It may also lock security deposits *dep* and the value of an encrypted transaction *val* so it can be transferred to the target contract.

2. $\mathsf{DKG}(\mathsf{m_{DKG}})$: During running the distributed key generation protocol, keepers submit their broadcast messages m_{DKG} and read others' from this contract. At the end of the protocol, it may also store the system-wide public key *mpk* and other public system parameters, so users can read it and encrypt transactions.

3. $\mathsf{Commit}(\mathsf{enc}(\mathsf{t_x}), \mathsf{H_c}(\mathsf{t_x}))$: This contract stores received encrypted transactions $enc(t_x)$ and cryptographic commitments $H(t_x)$. The main purpose of this contract is to preserve the order of received transactions.

4. $\mathsf{IBE}(\mathsf{b_h^k})$: This contract receives block key shares b_h^k from each keeper k, so the relayer can aggregate them to construct the block key.

5. $\mathsf{Process}(\mathsf{t_x}, \mathsf{x})$: This contract receives decrypted transactions t_x alongside the commitment's x, validates them, and executes them.

What follows is a description of FairBlock's design using smart contracts in seven phases:

Phase 0: Enrollment

Keepers and relayers enroll in participating in the protocol by sending a security deposit fee to Participate. Users also lock the value of their transaction in this contract so the value could be automatically transferred to the target contract in the last phase.

Phase 1: Distributed key generation

Keepers generate a shared public key *mpk*, and shares w_k of the associated master key *msk* split across all of them using a DKG protocol [82] by broadcasting messages in DKG. In Section 4.4, we discuss the DKG protocol in more detail.

Phase 2: Encryption and commitment

Users encrypt their message m (which includes their transaction information) using public key mpk and block identifier h. To encrypt transaction information $t_x \in \{0,1\}^{l_1}$, a user computes $Q_h = H_1(h)$ followed by selecting a random integer $r \in \mathbb{Z}_q^*$, and a random string $x \in \{0,1\}^{l_2}$. Afterward, it sets $m = t_x \| x$ and $R = g^r$ [16]. Having them, it computes the ciphertext value U as:

$$U = m \oplus H_2(\hat{e}(Q_h, mpk)^r). \tag{4.1}$$

Finally, it submits an encrypted message $C = (R, U)$ alongside a commitment $H_c(m)$ to Commit.

Phase 3: Transaction sequencing

Consensus validators of the underlying blockchain receive a list of encrypted transactions which are submitted to Commit, and run a consensus algorithm to agree on their unique order. Consequently, the fixed execution order will be publicly accessible and anyone can read it from Commit at no cost.

Phase 4: Broadcasting block key shares

Right after finalization of block $h-1$, keepers compute their block key shares b_h^k [16] and propagate it by submitting it to IBE.

Keeper k computes its share for block h as $b_h^k = H_1(h)^{w_k}$.

Phase 5: Decryption

Relayers can compute the block key b_h after reading at least $t+1$ valid shares from IBE. Next, they use b_h to decrypt each of the encrypted messages for the block h. Relayers are incentivized to decrypt correctly by rewards for each correct decryption. Each Relayer can extract block key b_h after the following steps:

1. *Share verification:* Relayer verifies each of the received shares b_h^k from keeper k by checking the following condition [16]:

$$\hat{e}(\prod_{i=0}^{t} V_i^{k^i}, H_1(h)) \stackrel{?}{=} \hat{e}(g, b_h^k), \tag{4.2}$$

where V_is are public verification values for keepers $i \in [0, t]$ in the DKG protocol defined as $V_i = \prod_{k \in T} A_{ki}$.

2. *Block key extraction:* After verifying the shares, the block key for block h is extracted as follows:

$$b_h = \prod_{k=1}^{t+1} (b_h^k)^{L_k}, \tag{4.3}$$

where L_ks are Lagrange coefficients for point 0 defined as $L_k = \prod_{\substack{r=0 \\ r \neq k}}^{t} \frac{r}{r-k}$. The derived key b_h is indeed the IBE key for identity h that would have been extracted by the TTP. See Section 4.5.1 for correctness discussion.

3. *Decryption:* Let $C = (R, U)$ be the ciphertext for block identifier h. A Relayer decrypts C using the private key b_h as:

$$m = U \oplus H_2(\hat{e}(b_h, R)). \tag{4.4}$$

Finally, the relayer submits the batch of the decrypted messages to Process. In the event that relayers are unable to include the decrypted transactions in the desired block (or the desired block range) h, an application-specific policy determines if the submitted encrypted transactions should fail or be included in a later block.

Phase 6: Execution

Given a list of decrypted messages $m_1, ..., m_n$, Process extracts x and t_x for each of the messages. Next, it checks the validity of each transaction t_x e.g. user's balance in Participate which should be more than the transaction value. Finally, it verifies that a) none of the committed transactions is censored, b) all of them have been decrypted correctly, and c) their received order follows the specified ordering policy. This verification can be simply done by recomputing the cryptographic commitment for each of decrypted messages, and then reading previously submitted (in phase 2) cryptographic commitments $H_c(m)$ from Commit for each of them. The verification can be done in a single step by computing the hash of all commitments as $H_c(m_1, ..., m_n)$ and comparing them. Finally, it executes the batch by calling the target contract for each of the decrypted transactions.

Figure 4.1 illustrates the high-level workflow of the described smart-contract based design.

The proposed smart-contract based design has the advantage of preventing front-running for dApps in legacy blockchains solely on-chain and independent of any other network for keepers and relayers. To leverage this design target applications should require a simple condition in their smart contract to receive all incoming transactions through Process so other users cannot front-run the batch of decrypted transactions by sending direct transactions to the target contract. Although this implementation is realistic and can be used for many applications such as auctions, it may get impractical for some use cases such as DEXes with hundreds of sensitive transactions. Storing and broadcasting DKG shares and Block key shares on target blockchains such as Ethereum [105] (which hosts the majority of current dApps) can be very expensive and slow. Given the above trade-offs. Next, we describe an alternative instantiation based on decentralized oracle networks.

4.3.2 Oracle-based instantiation

In oracle-based instantiation, a distributed network of validators plays the roles of keepers and relayers. Figure 4.2 illustrates the high-level workflow of this instantiation. In summary, users will submit their commitments to their future transactions to a smart contract. After the finalization of the previous block of the desired height for decryption on the target blockchain, validators send their block key shares as a transaction in the oracle network. After broadcasting sufficient shares, any validator can compute the block key, read the committed transactions from the commit contract with the fixed order, decrypt them, and submit the batch of transactions to the target contract. The distributed oracle network which is normally implemented as a blockchain with a BFT-style consensus such as Tendermint [21] offers much faster and more economical transactions with instant finality for broadcasting messages in the DKG protocol, and block key extraction. This implementation also offers the additional feature of sending transactions to smart contracts hosted on any blockchains. Our construction of oracle-based instantiation is based on the following building blocks:

1. $\mathsf{Commit}(\mathsf{enc}(\mathsf{t_x}), \mathsf{H_c}(\mathsf{t_x}))$: This contract stores received encrypted transactions $enc(t_x)$ and cryptographic commitments to them $H_c(t_x)$. The main purpose of this contract is to preserve the order of received transactions and it can be hosted on any smart contract blockchain. In this work, we assume that this smart contract is hosted in the same blockchain as the target application for a better user experience. In case of high gas fees or extensive delays, Commit can also be hosted in an economical and fast blockchain with instant finality.

2. FairBlock Network: Decentralized oracle networks enhance the capabilities of a target blockchain by providing data from an off-chain source and bringing it on-chain. FairBlock network is a distributed oracle network implemented as a Byzantine Fault Tolerant system such as Tendermint [21]. Validators in this network play the role of keepers and relayers, by gathering commitments from a source other than the target blockchain, performing computations which are not possible on-chain and providing it to the target blockchain. For this design, we use the terms validators, keepers, and relayers interchangeably.

What follows is a brief description of FairBlock's design using FairBlock Network:

Phase 0: Enrollment

Validators enroll in FairBlock Network by staking a certain amount of token. Their stake can be slashed in case of dishonest cooperation. A decentralized autonomous organization (DAO) can hire the set of validators.

Phase 1: Distributed key generation

Keepers generate a shared public key mpk, and shares w_k of the associated master key msk split across all of them using a DKG protocol [82] by broadcasting messages in FairBlock Network. In Section 4.4, we discuss the DKG protocol in more detail.

Phase 2: Encryption and commitment

Users encrypt their message m (which includes their transaction information) using public key mpk and block identifier h. To encrypt transaction information $t_x \in \{0,1\}^{l_1}$, a user computes $Q_h = H_1(h)$ followed by selecting a random integer $r \in \mathbb{Z}_q^*$, and a random string $x \in \{0,1\}^{l_2}$. Afterward, it sets $m = t_x \| x$ and $R = g^r$ [16]. Having them, it computes the ciphertext value U as:

$$U = m \oplus H_2(\hat{e}(Q_h, mpk)^r). \tag{4.5}$$

Finally, it submits an encrypted message $C = (R, U)$ alongside a commitment $H_c(m)$ to Commit.

Phase 3: Transaction sequencing

Consensus validators of the underlying blockchain receive a list of encrypted transactions which are submitted to Commit, and run a consensus algorithm to agree on their unique order. Consequently, the fixed execution order will be publicly accessible and anyone can read it from Commit at no cost.

Phase 4: Broadcasting block key shares

Right after the finalization of block $h-1$, keepers compute their block key shares b_h^k [16] and broadcast them in FairBlock Network to be written on the public ledger of FairBlock Network. Keeper k computes its share for block h as $b_h^k = H_1(h)^{w_k}$.

Note that the consensus mechanism in FairBlock Network is significantly faster than the target blockchain; so there is enough time for the next steps before the finalization of the desired block in the target blockchain.

Phase 5: Decryption.

One of the validators (as a relayer) computes the block key b_h after reading at least $t+1$ valid shares from the public ledger. Next, it uses b_h to decrypt each of the encrypted messages for the block h. Relayer can extract block key b_h after the following steps:

1. *Share verification:* Relayer verifies each of the received shares b_h^k from keeper k by checking the following condition [16]:

$$\hat{e}(\prod_{i=0}^{t} V_i^{k^i}, H_1(h)) \stackrel{?}{=} \hat{e}(g, b_h^k), \tag{4.6}$$

   where V_is are public verification values for keepers $i \in [0,\ t]$ in the DKG protocol defined as $V_i = \prod_{k \in T} A_{ki}$.

2. *Block key extraction:* After verifying the shares, the block key for block h is extracted as follows:

$$b_h = \prod_{k=1}^{t+1} (b_h^k)^{L_k}, \tag{4.7}$$

where L_ks are Lagrange coefficients for point 0 defined as $L_k = \prod_{\substack{r=0 \\ r \neq k}}^{t} \frac{r}{r-k}$. The derived key b_h is indeed the IBE key for identity h that would have been extracted by the TTP. See Section 4.4 for correctness discussion.

3. *Decryption:* Let $C = (R, U)$ be the ciphertext for block identifier h. A Relayer decrypts C using the private key b_h as:

$$m = U \oplus H_2(\hat{e}(b_h, R)). \tag{4.8}$$

For each decryption batch, there is a designated relayer chosen in a round-robin fashion with respect to the proportion of their stake.

Phase 6: Execution

The same relayer in the decryption phase does the following:

1. Check validity constraints of decrypted transactions e.g. user's balance.
2. Batch the valid plaintext transactions with respect to the received order in Commit and target contract's policy.
3. Make a single transaction which includes all the ordered plaintext transactions in the data field.
4. Validators can send the transaction to the target contract using a multi-signature wallet of the target blockchain. Validators can verify decrypted transactions (using cryptographic commitments), batching process (with respect to Commit and target contract's policy). If the meta-transaction is valid, the majority of validators will sign it and it will be sent to the target contract. However, in the case of long delays or manipulation of the meta-transaction, the responsible relayer will be slashed and the next relayer in line will be responsible for decryption and forming the correct meta-transaction.

In the event that FairBlock Network cannot decrypt transactions for the desired block (or the desired block range) h, an application-specific policy determines if the submitted encrypted transactions should fail or be included in a later block.

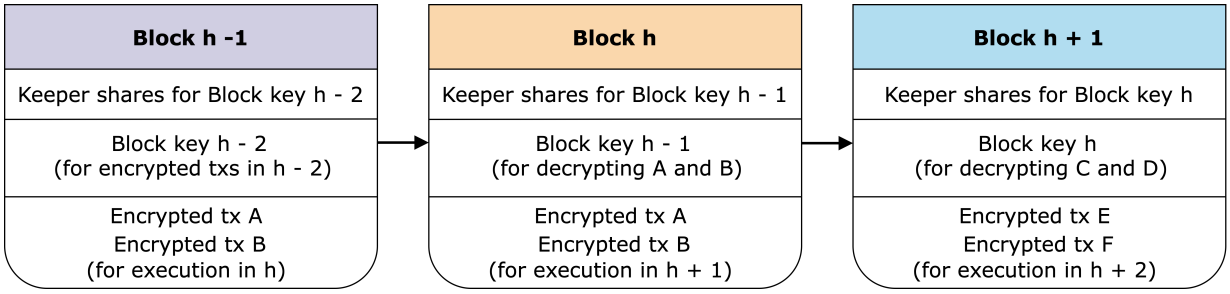

Figure 4.3: Block structure in consensus-level instantiation

4.3.3 Consensus-level instantiation

An alternative to using smart contracts or oracle networks as the communication layer is implementing FairBlock in the consensus layer [92, 2]. In this case, there will be no need to maintain external sets of keepers and relayers, as the normal validators who are responsible for mining the blocks will also play these roles. Moreover, implementing FairBlock in the consensus layer eliminates some of the user experience, composability, and additional costs of the previous approaches. In particular, users broadcast their encrypted transactions directly to the public mempool of the underlying blockchain without submitting it to Commit. Also, this approach does not enforce additional costs for storing on-chain data, transaction fees, and maintaining keepers and relayers sets. Last but not least, this implementation enables sending general transactions e.g. transfers and is not limited to a target contract for executing decrypted transactions.

In this design, the target contract is hosted in Host blockchain which is a proof-of-stake blockchain with BFT-style consensus mechanism such as Tendermint [21] or Ethereum Beacon Chain [105]. In this design, consensus validators of Host blockchain play the roles of keepers and relayers, and we use the terms validators, keepers, and relayers interchangeably. We now briefly show how to integrate the FairBlock technique in the consensus layer:

Phase 0: Enrollment

In this design, there is no additional enrollment step for validators except the default process of adding a node to the validator set in Host blockchain. Normally, to become a validator, it is necessary to keep a minimum amount of Host blockchain tokens in stake. This stake can be also used for slashing the dishonest validators in this design.

Phase 1: Distributed key generation

Keepers generate a shared public key mpk, and shares w_k of the associated master key msk split across all of them using a DKG protocol [82] by broadcasting messages in Host blockchain. In this approach, validators receive master key shares in proportion to their stake. In Section 4.4, we discuss the DKG protocol in more detail.

Phase 2: Encryption

Users encrypt their message m (which includes their transaction information) using public key mpk to be included in block $h-1$ and executed after the finalization of block h. To encrypt transaction information $t_x \in \{0,1\}^{l_1}$, a user computes $Q_h = H_1(h)$ followed by selecting a random integer $r \in \mathbb{Z}_q^*$, setting $R = g^r$ [16]. Having them, it computes the ciphertext value U as:

$$U = t_x \oplus H_2(\hat{e}(Q_h, mpk)^r). \tag{4.9}$$

Finally, it broadcasts an encrypted message $C = (R, U)$ in Host blockchain's public mempool to be included in block $h-1$.

Phase 3: Transaction sequencing

Consensus validators of Host blockchain receive a list of encrypted transactions which are submitted to the public mempool, and run a consensus algorithm to agree on their inclusion order in block $h-1$. Consequently, the fixed execution order will be publicly readable from block $h-1$ at no cost.

Phase 4: Broadcasting block key shares

Right after the finalization of block $h-1$, keepers compute their block key shares b_h^k [16] and broadcast them in Host Blockchain's public mempool to be included in the block h. Keeper k computes its share for block h as $b_h^k = H_1(h)^{w_k}$. Keepers should sign their shares with their own private key. This key pair should not be the same key pair of the consensus mechanism and should be generated, distributed, and updated safely through standard key management protocols. This key pair enables keepers to send their block key shares without paying any transaction fees.

Phase 5: Block key extraction

The proposer of block h (as the relayer) includes all block key shares in block h and also computes and includes the block key b_h after reading at least $t+1$ valid shares. The proposed block h will only be verified by other validators if it has the correct block key, as other validators can also compute the key locally. Note that block h will also include encrypted transactions for block $h+1$. The next block proposer can extract the block key b_h in the following steps:

1. Keeper verification: The block proposer verifies the sender of each of the received shares b_h^k with each keeper's public key.

2. *Share verification:* The block proposer verifies each of the received shares b_h^k from keeper k by checking the following condition [16]:

$$\hat{e}(\prod_{i=0}^{t}(V_i)^{k^i}, H_1(h)) \stackrel{?}{=} \hat{e}(g, b_h^k), \tag{4.10}$$

   where V_is are public verification values for keepers $i \in [0,\ t]$ in the DKG protocol defined as $V_i = \prod_{k \in T} A_{ki}$.

3. *Block key extraction:* After verifying the shares, the block key for block h is extracted as follows:

$$b_h = \prod_{k=1}^{t+1}(b_h^k)^{L_k}, \tag{4.11}$$

   where L_ks are Lagrange coefficients for point 0 defined as $L_k = \prod_{\substack{r=0 \\ r \neq k}}^{t} \frac{r}{r-k}$. The derived key b_h is indeed the IBE key for identity h that would have been extracted by the TTP. See Section 4.5.1 for correctness discussion.

4. *Decryption:* Let $C = (R, U)$ be the ciphertext for block identifier h. A Relayer decrypts C using the private key b_h as:

$$t_x = U \oplus H_2(\hat{e}(b_h, R)). \tag{4.12}$$

Phase 6: Decryption and execution

Once the block h which contains the block key for encrypted transactions in block $h-1$ is finalized, all the nodes in the blockchain can read the encrypted transactions from block $h-1$, decrypt them, and execute them in order. Reading transaction from the previous block and decryption process should be added to the state transition logic of each node, and the rest of the execution logic follows the same process of current PoS blockchains.

Figure 4.3 illustrates the high-level workflow of consensus-level instantiation of FairBlock.

4.4 Distributed key generation

As described in Section 4.3, keepers are generating block keys in a distributed way so a single keeper cannot decrypt transactions before the desired block number. To do so, keepers need to get their secret shares, so they can extract their shares of the block key. In this section, we explain how distributed key generation schemes can be integrated into FairBlock so keepers can generate their master secret key shares w_k and the associated system-wide public key mpk without trusting any third party. First, we describe the basic Shamir secret sharing (SSS) [89] technique and then proceed to introduce DKG protocols with desired properties that have been built on top of it.

4.4.1 Shamir secret sharing

A trusted dealer can distribute the master secret key between keepers in a way that at least $t+1$ of n keepers cooperate to reconstruct it. To realize this, Shamir secret sharing (SSS) [89] employs the Lagrange interpolation theorem which states that a polynomial of degree t or less can be uniquely determined by $t+1$ points.

In SSS, the trusted dealer generates a master secret key $msk \in \mathbb{Z}_q^*$ and constructs a random polynomial with msk as the constant ($a_0 = msk$):

$$f(z) = msk + a_1 z + a_2 z^2 + \cdots + a_t z^t. \tag{4.13}$$

Next, the dealer should privately send shares w_k to all n keepers. For each keeper k , the share is simply computed as:

$$w_k = f(k). \tag{4.14}$$

Consequently, any $t+1$ keepers can aggregate their shares w_i to reconstruct msk using Lagrange Interpolation formula as:

$$msk = \sum_{k=0}^{t} w_k L_k, \tag{4.15}$$

where L_ks are Lagrange coefficients for point 0 defined as:

$$L_k = \prod_{\substack{j=0 \\ j \neq k}}^{t} \frac{j}{j-k}. \tag{4.16}$$

The dealer can also compute and broadcast the associated mpk as g^{msk}.

4.4.2 Verifiable secret sharing

In SSS, we assumed that the dealer is sharing correct shares; however, the dealer can maliciously send inconsistent shares. To protect against malicious shares, keepers should be able to verify the received shares. Feldman [48] has extended SSS and proposed a verifiable secret sharing (VSS) technique to solve this issue. In VSS, the dealer also broadcasts homomorphic commitments to coefficients of the random polynomial f. The homomorphic commitment to coefficients a_i for $i \in [0, t]$ is computed as:

$$A_i = g^{a_i}. \tag{4.17}$$

Consequently, any keeper can verify its received share w_k by checking the following condition:

$$g^{w_k} \stackrel{?}{=} \prod_{i=0}^{t} A_i^{k^i}. \tag{4.18}$$

The correctness of this validity check follows the fact that:

$$g^{w_k} = g^{f(k)} = g^{a_0 + a_1 k + a_2 k^2 + \cdots + a_t k^t} = g^{\sum_{i=0}^{t} a_i k^i} = \prod_{i=0}^{t} g^{a_i k^i} = \prod_{i=0}^{t} A_i^{k^i}. \tag{4.19}$$

4.4.3 Pedersen's distributed key generation

Although VSS allows keepers to verify the received shares, the need for a trusted dealer is contradictory to the distributed nature of blockchains. In particular, a trusted dealer constitutes a single point of failure which can maliciously leak block keys before finalization of batched encrypted transactions, or be a target of DOS attacks. A distributed key generation protocol (DKG) allows keepers to generate a master secret shared key, and its associated mpk while preventing any single keeper from computing the msk. In this section, we describe Pedersen's DKG protocol [82] (Ped-DKG) as a simple and efficient DKG protocol which is the basis of many more complex DKG schemes with different properties. The following is a description of Joint-Feldman DKG protocol [53] (JF-DKG) which is Ped-DKG with a simplified dispute phase:

1. *Sharing:* Keepers run n parallel VSS, each acting as a dealer. To be more specific, each keeper P_i randomly picks a secret $s_i \in \mathbb{Z}_q^*$. Next, P_i sets $a_{i0} = s_i$ and chooses a random polynomial $f_i(z)$ over $\mathbb{Z}_q^*$ of degree t as follows:

$$f_i(z) = a_{i0} + a_{i1}z + \cdots + a_{it}z^t. \tag{4.20}$$

   P_i broadcasts Feldman's [48] commitments $A_{ik} = g^{a_{ik}}$ for $k \in [0, t]$. P_i computes the share $s_{ij} = f_i(j) \bmod q$ for $j \in [1, n]$ and sends s_{ij} through secure private channels to P_j.

2. *Share verification:* Each keeper P_j verifies each received share s_{ij} sent by P_i. To do so, P_j checks Feldman's VSS [48] validity condition: $g^{s_{ij}} \stackrel{?}{=} \prod_{k=0}^{t} A_{ik}^{j^k}$.

3. *Dispute:* If t or more keepers complain against a keeper P_f by broadcasting the complaint, P_f will be considered faulty and disqualified. Subsequently, P_f can make a complaint and claim its honesty by revealing the share s_{fv} for each complaining user P_v . If any of the revealed shares fails the check again, P_f is disqualified.

4. *Public Key:* Assuming that T is the set of qualified (not disqualified in the previous phase) keepers, the system-wide public key mpk is computed as follows:

$$mpk = \prod_{i \in T} A_{i0} = \prod_{i \in T} g^{s_i}. \tag{4.21}$$

5. *Master key shares:* Each keeper $k \in T$ compute its master key share $w_k = \sum_{i \in T} s_{ik}$.

Although there is no need to reconstruct the msk through our protocol, it is defined as the sum of qualified keepers' secrets: $msk = \sum_{i \in T} s_i$. Note that the secret s_f for a disqualified keeper P_f is set to zero.

4.4.4 Improving Pedersen's DKG

In this section, we briefly explain some of the most popular DKG schemes which are relevant for this application. In addition to proposing the described JF-DKG, Gennaro et al. [53] showed that uniform randomness is not guaranteed for the generated shared secret key in Ped-DKG and JF-DKG. Further, they proposed a DKG protocol with guaranteed uniform randomness for the shared secret at the cost of increasing the number of communication rounds. Nevertheless, the same authors revisited the problem in [52] and suggested that the discrete log problem in Ped-DKG and JF-DKG is still sufficiently hard for some threshold cryptographic schemes including this work. Furthermore, as the dispute claim in JF-DKG may be invalid and requires further interactions from the accused party to prove its honesty, a non-interactive zero-knowledge proof technique proposed by [88] can be used to support dispute claims resulting in reduced communication between parties. Although Ped-DKG and JF-DKG offer excellent performance due to their simplicity, they suffer from a complicated dispute phase which can increase their communication complexity. Aggregatable DKG(Agg-DKG) [58] improves $O(n^2)$ time complexity of Ped-DKG, JF-DKG, and many other proposed DKG including [65] to $O(n\,log\,n)$ by eliminating the dispute round. However, the main challenge with Agg-DKG is that threshold cryptographic schemes such as IBE require secret key shares in scalar fields, whereas generated secret key shares in Agg-DKG are group elements.

4.4.5 DKG in FairBlock

In this section, we describe more detailed mechanisms for integrating a DKG scheme in FairBlock architecture so n keepers can generate their secret shares w_k and the associated public key mpk. Pedersen's DKG is relatively slow compared the encryption scheme as it requires local share verification time quadratic in n [65, 82, 96] and up to three rounds of broadcasts. Moreover, even more sophisticated DKG schemes that we introduced in Section 4.4.4 do not offer enough scalability in terms of time and message complexity for frequent execution in this application. However, this work requires running a DKG protocol once in the setup phase and afterward very infrequently to reflect keepers' set changes. As long as the number of compromised parties is below the threshold, the changes can be batched and executed at once. Also, keepers' set is expected to be stable as they are collecting rewards for honest co-operation and being slashed for malicious behavior.

To be more specific, FairBlock needs to run the DKG process infrequently in regular windows to a) invalidate secret shares of keepers that have left or slashed during the window, b) issue shares for newly joined keepers, c) adjust secret share weights (if we distribute

secret shares in proportion to keepers' stake e.g. in the consensus layer implementation). Moreover, this periodic update of secret shares offers Proactive Secret Sharing (PSS) [61] in the sense that the vulnerability window for compromising or leaking sufficient key shares is limited. Particularly, now the adversary should control sufficient key shares (more than the threshold) within the refreshing window. Additionally, key refreshment can be done in a way that the system public key and the underlying master secret key remain the same. To do so, keepers should run another run of DKG with the difference that s_i should be set to zero in their new random polynomial. Consequently, secret shares w_k can be updated by adding the resulted secret shares to their old shares.

Another simple technique is also proposed [2] to remove the need for running DKG every time that a keeper wants to join the set. In this approach, keepers can generate extra secret shares and distribute them among them in a way that no keeper can reconstruct it independently. As a result, a newly joined keeper can aggregate these shares from all keepers to reconstruct its secret share without running DKG again. However, this approach only helps with hiring new keepers but does not contribute to the security of the system in the sense that the threshold always remains that of the smaller set. Also, more complex secret redistribution and proactive secret sharing techniques with different security properties can be adopted in the spirit of [104, 40, 61, 43].

As discussed earlier in each design, different mechanisms such as DAOs, standard validator staking, and auctions can be used to hire and slash keepers. Keepers use the blockchain (either using a smart contract or by inclusion in the block) for broadcasting their public messages, and private communication channels or standard public-key cryptography to send their private shares. Normally, each DKG process is expected to be done in two blocks of the underlying blockchain network; so each DKG process should be initiated at least two blocks before the desired deadline. In Section 5.1.2, we discuss DKG implementation and libraries.

4.5 Correctness and security

4.5.1 Correctness

Correctness follows by the linearity of secret sharing and IBE extraction algorithm. In particular, it is easy to see that shares of the private key can be reconstructed to obtain the IBE key.

Correctness proof for distributed private key extraction

The following proof shows that b_h is indeed the IBE key that a trusted third party extracts for the identity h by raising the hash of the identity $H_1(h)$ to its private key msk:

$$b_h = \prod_{k=0}^{t} (b_h^k)^{L_k} = \prod_{k=0}^{t} (H_1(h)^{w_k})^{L_k} = \prod_{k=0}^{t} H_1(h)^{w_k L_k} = H_1(h)^{\sum_{k=0}^{t} w_k L_k}. \tag{4.22}$$

And by Lagrange interpolation formula we have:

$$H_1(h)^{\sum_{k=0}^{t} w_k L_k} = H_1(h)^{msk}. \tag{4.23}$$

Consistency of IBE encryption and decryption

Let $C = (R, U)$ be the encryption of message m for block identifier h using the public key mpk. In encryption, m is bitwise XORed with the hash of $\hat{e}(Q_h, mpk)^r$. Subsequently in decryption, U is bitwise XORed with the hash of $\hat{e}(b_h, R)$. These two masks are equal since:

$$\hat{e}(Q_h, mpk)^r = \hat{e}(Q_h, g)^{r.msk} = \hat{e}((Q_h)^{msk}, g^r) = \hat{e}(b_h, R). \tag{4.24}$$

4.5.2 Security

Secure causality-preservation

Assuming the honest majority of keepers described in Section 4.2.3, and at least one honest relayer, we show that our system satisfies causality-preservation based on the security of DKG and IBE schemes.

In particular, at the end of phase 1, the adversary cannot learn any information of the msk given $t \leq \frac{n-1}{2}$ shares of the msk. Furthermore, given block keys b_h for block identifiers $h \in S_h$, the adversary cannot learn about the block key of other block identifiers $h^* \notin S_h$ by the properties of IBE.

We prove security of FairBlock by defining a security game G between a polynomially bounded adversary $\mathcal{A}$ controlling at most t keepers and a challenger. The adversary's goal is to front-run a user, defined as being able to distinguish between two challenge encrypted messages containing transaction information. We define the security game as follows:

- The challenger runs DKG and IBE $Setup(1^\lambda)$ algorithm.

- The adversary receives shares of msk, $(w_1, ..., w_k)$ for $k \leq t$.
- The adversary computes $Encrypt(mpk, h, m)$ for arbitrary message m and any block identifier $h \in S_h$.
- The adversary receives $q \leq n$ shares for the block key b_h.
- The adversary chooses two distinct message m_0, m_1, and a block identifier $h^* \notin S_h$ and sends them to the challenger.
- The challenger selects random bit b and sends $C^* = Encrypt(mpk, h^*, m_b)$ to the adversary alongside up to $t-k$ shares of the block key sk_{h^*}. The number of received shares in this phase cannot be more than $t-k$, as the adversary can exploit its shares of msk and extract additional k shares for sk_{h^*}. In case of receiving more than $t-k$ shares of the challenge block key, the adversary can trivially extract the challenge block key by combining more than t shares of sk_{h^*} shares.
- The adversary can still query the oracle to get $q \leq n$ shares of the block key b_h for any $h \neq h^*$, and finally outputs a guess for b.

Let W be the event that an adversary succeeds in the game G by correctly guessing b in polynomial time, and ε be a negligible function of the security parameter λ which is fed to the scheme in the setup phase. We say that the protocol is secure against front-running if:

$$\mathsf{Adv}_G(\mathcal{A}) = \mid \Pr[W] - 0.5 \mid \leq \varepsilon \tag{4.25}$$

To show that our scheme is secure according to the definition above, let us assume that there exists an adversary $\mathcal{A}$ which can win the game with a non-negligible probability. We can show that in turn this adversary $\mathcal{A}$ should either break the security of our distributed IBE scheme or the underlying DKG protocol. In the former case, an adversary $\mathcal{B}$ can obtain a private key d_{ID} for arbitrary identity ID alongside the two challenge ciphertexts from the challenger in the security game of standard IBE. Next, it runs a secret sharing algorithm on d_{ID} to generate shares with the same distribution of block key shares and submits the generated shares, two challenge ciphertexts, and ID to $\mathcal{A}$. Consequently, $\mathcal{A}$ outputs b which can be sent to the challenger by $\mathcal{B}$ to break the security of standard IBE with a non-negligible probability. To break the DKG security in the latter, $\mathcal{A}$ should have the ability to distinguish between the distribution of master key shares and master public key tuple $(w_1, ..., w_k, mpk)$ as the output of a simulator Sim and $(w_1, ..., w_k, mpk')$ as the output of the real DKG protocol [53, 82]. Consequently, as we have not modified the DKG protocol in FairBlock, an adversary $\mathcal{C}$ can simply use $\mathcal{A}$ as an oracle to break the security of the original DKG protocol.

Order-fairness

FairBlock achieves order-fairness by executing transactions in the order that their commitments have been received and written to Commit or alternatively distributed public ledger (in the case of consensus-level implementation) without duplication. No party including block proposers in the decryption and execution phase can influence the fixed order of executed transactions. Moreover, no party can insert transactions before the decrypted batch or directly submit transactions to the target contract to front-run or out-race FairBlock transactions as the target contract only accepts messages received through FairBlock.

To achieve order-fairness, we follow the literature on "fairness" [66, 28, 110] which favors the transactions that are received earlier. This property has been a subject of debate in the blockchain community lately [37, 28, 92]. In some applications e.g. auctions or networks, it is vital to preserve the order of received transactions for the correctness of the auction or incentivize parties to act with the lowest possible latency e.g. arbitragers in AMMs [36]. The other side of this trade-off is that this property may be exploited to perform blind front-running in some applications e.g. initial coin offerings (ICO) or attacks based on metadata. To the extent of our knowledge, blind front-running and attacks based on metadata are negligible in current applications. However, FairBlock can be easily modified to prevent this type of attack by shuffling the ordering of transactions. The source of shuffling can be the hash of the concatenation of random strings x of all messages which cannot be pre-determined or influenced. Kelkar et al. [66] propose executing all the received transactions in parallel which is implemented in Chainlink fair sequencing service (FSS) [28] and also compatible with FairBlock's encryption mechanism. We have further discussed other solutions to combat metadata-based attacks by anonymizing the transaction's sender in Section 4.6.

4.6 Practical considerations

4.6.1 Transaction execution

In Section 4.3, we proposed several designs to fix the ordering of encrypted transactions to prevent front-running attacks. However, we did not discuss the practical considerations of enforcing this fixed order and executing the batch of decrypted transactions.

Here we present different design options for processing and executing decrypted transactions:

Direct mempool submission

The natural way of executing transactions is broadcasting them in the public mempool of the target blockchains. In this approach, the relayer will submit the decrypted transactions as a standard transaction of the target blockchain sequentially. Block proposers of the target blockchain is responsible to put these transactions with respect to the fixed order. One way to enforce it is using the transaction's nonce in blockchains such as Ethereum [105]. Nonce is a simple transaction counter which assures that transactions of an individual sender will be executed in order. Other alternative approaches such as appending the ordering number to the transactions, or reading the order from the commit contract can realize this property in blockchains where nonce is not used. Also, we have assumed that the target applications prioritize decrypted transactions of FairBlock to prevent direct transactions. Consequently, while the decrypted transactions will be submitted in the public mempool in clear, another player cannot front-run them by sending another transaction and front-running the whole batch.

The main challenge with the direct mempool submission is the power of the block proposer to censor specific transactions from the decrypted batch. Although front-running the decrypted batch is not possible, block proposers can still refuse to put all the transactions in their block for more profit. In this case, the target smart contract should wait until all the decrypted transactions are included in the block, which makes this approach impractical for sensitive applications with many transactions such as DEXes. However, this censorship problem does not affect the consensus-layer and layer two implementation as the transaction has been already included in the block before decryption.

Meta-transaction

Decrypted transactions can be batched together and sent as a single meta-transaction to the target application; to be more specific, users should include necessary information for their transaction e.g. their desired trade and their signature on the trade in the encrypted transaction. Consequently, the relayer aggregates information of each committed transaction after decryption and puts them in the data field of a standard Ethereum transaction [105]. As a result, all transactions will be included in the block at the same time, and the block proposer cannot censor some of them. In addition to front-running protection and censorship resistance, this approach also offers significant savings on transaction fees. In particular, users should not pay transaction fees for each of their transactions to be included in the block as this fee can be paid only once for all the transactions in the batch. A minor technical challenge with this approach is designing the target application

in a way that it can execute the meta-transaction. Alternatively, a proxy smart contract can be used to receive the meta-transaction, parse them, and send them sequentially to the target contract. Although this approach may be more expensive, it has the advantage of interacting with the target application almost seamlessly without significant modification in the target contract. We have further explored the implementation of the described meta-transaction in more technical detail in Section 5.1.3.

4.6.2 Metadata front-running

One of the main challenges that arises in all privacy-preserving implementations is to protect leakage of information through transaction metadata. In Section 4.3, we focused on designing protocols that protect transaction data, and not transaction metadata, since currently almost all of the known front-running strategies are just using plaintext transaction data in the mempool.

However, transaction metadata such as the sender's address, IP addresses, and transaction fees can also be used in more sophisticated front-running strategies. In this section, we explore some of the metadata front-running opportunities and possible countermeasures that can be applied to FairBlock as additional layers of security to mitigate leakage of metadata as well:

Hiding sender address

In theory, an adversary with just the knowledge of the transaction's sender address can perform front-running attacks. For example, traders can be front-runned just based on their regular trading pattern for specific assets. These are some possible countermeasures for hiding the sender's address:

- **Ring signatures:** Ring signatures [77] can be used to hide the transaction sender's address. A ring signature is a type of digital signature in which a group of possible signers are merged to produce a distinctive signature that can authorize a transaction. However, implementing and using these signatures have been challenging and complex in popular public blockchains such as Ethereum [105].
- **Zero-Knowledge proofs:** Zero knowledge proofs (ZKP) such as zk-SNARKs can be also utilized to prove the validity of a transaction e.g. sufficient sender's balance for the transaction value, without leaking any other information about the transaction.

However, despite the recent growth of ZKP research, implementations, and applications, this approach can be expensive due to its additional overhead in computation, bandwidth, and integration complexities.

- **Relayer services:** As a simpler and more practical alternative to cryptographic solutions, users can hide the real sender of their transaction by asking another party to send their transactions. Since this approach can also hide transaction fees, we will discuss it in more detail in Section 4.6.2 along with hiding transaction fees to prevent duplication.

Other techniques including [13, 97, 25] can also be applied to achieve better privacy.

Hiding transaction fee

It is also possible to analyze transaction fees to perform front-running. Particularly, transaction fees in smart contract blockchains such as Ethereum [105] are calculated based on the amount of computational effort required to execute the transaction. Consequently, the fee can leak minimal information about the encrypted transactions that can be used in complex front-running strategies. Simply encrypting the transaction fee along with transaction data will enable an adversary to perform DDOS attacks as the adversary can spam the network with many free and invalid transactions. Some of the alternative approaches to hide the transaction fees are the following:

- **Cryptographic schemes:** Transaction fee can be also encrypted alongside the transaction, and zero-knowledge proof schemes such as zk-SNARKs can be potentially used to prove that a transaction have enough funding to pay for transaction fees after decryption. Moreover, functional encryption (FE) schemes [17] can also prove that the encrypted offered transaction fee is sufficient for block inclusion and execution. Functional encryption allows computing a function on encrypted data without leaking any other data. In this case, a simple comparison function can be computed on the encrypted offered transaction fee to make sure that it is greater than a certain threshold without revealing its actual value. However, these cryptographic approaches can result in high computational and bandwidth costs.

- **Hybrid fee:** In some blockchain networks such as Ethereum (after EIP-1559 [24, 51, 105]), the transaction fee consists of a minimum block inclusion fee as well as a computation fee for the execution of a smart contract. Although the block inclusion

fee can determine the order of transactions in the block, it does not carry any information about the execution cost of the smart contract and therefore not useful for front-runners [37]. Consequently, this minimum fee can be broadcasted in plaintext to assure the block proposers of their reward for including their transaction in the block; while the sensitive computation fee can be encrypted alongside other parts of the transaction. Once the encrypted computation fee is included in the block, it can be decrypted and validated. If the decrypted computation fee is not sufficient for the smart contract execution, the transaction can be immediately rejected. Transaction sender will lose the block inclusion fee which can be considered as an incentivization mechanism to prevent invalid transactions.

- **Relayer services:** As an alternative to cryptographic solutions, users can hide the transaction fee by asking another party to pay the transaction fee. Users can sign a message containing information about their transaction and privately send it to a relayer server through an API or RPC. Relayer services such as Gas Station Network [57] can send a user's transaction and pay for the transaction fee from the relayer's wallet. Transaction fees can be either transferred to the relayer service indirectly (even with other tokens or fiat), or even the target application can sponsor the transaction fees itself to make the user experience much easier as users do not need to have the native token, pay for transaction fees directly, and risk their privacy. Sponsoring the transaction fees can be even an economically beneficial choice for target applications, as a straightforward UX normally leads to more user interactions with applications and therefore more income which can potentially compensate for the transaction fees. Also, to prevent potential DOS attacks to relayers, PoW puzzles such as [7] or standard CAPTCHA solutions can be employed on the user side.

Hiding IP address

Although it is not possible to establish a link between sender addresses and IPs solely based on on-chain data, the traffic and outgoing transactions from a particular IP address can be monitored to establish a relation between them. As an additional security layer to FairBlock, Tor network [41] practically hides IP addresses and prevents front-running based on capturing the traffic data by randomizing sender's IP addresses.

4.6.3 Chicken-out attack

During this project, we have discovered a potential attack strategy to FairBlock, and also related works such as Chainlink FSS [28] or Shutter Network [91]. This attack is only possible in implementations which allow transaction values to be deducted directly from users' wallets; for example, in the case that users encrypt normal transactions and relayers submit them directly to the public mempool after decryption. This strategy can be described in the following steps:

- An adversary submits an encrypted transaction to be executed in block h.
- Once the batch of decrypted transactions is submitted to the public mempool e.g. as a meta-transaction, the adversary can learn about other transactions in the batch.
- The adversary can analyze the market again with respect to other transactions in the batch. If the adversary decides that the transaction is not profitable anymore, she can deliberately make it fail.
- To make the transaction fail, the adversary can submit a transaction from her wallet to another address and front-run the whole batch of the transactions by offering a high transaction fee.

As a consequence, the adversary can have the ability to commit to a transaction only if it is profitable, mitigate her loss by making it fail, and possibly sabotage other users' profits who have submitted an encrypted transaction. To make things worse, an attacker can also commit to two messages in different directions e.g. two trades to buy and sell a token, or different votes and make the one which is not profitable fail, while keeping the profitable one.

Due to the technical complexity of this attack, unknown opportunities in failing a transaction, and additional fees such as the need to pay a very high transaction fee to front-run the whole batch (which can be a meta-transaction with a very high combined transaction fee), and losing deposits; this attack may not be practical in real-world use cases. However, we think it is important to discuss the possible countermeasures:

- **Consensus layer modification:** Consensus mechanism can be changed in a way that block proposers put the decrypted transactions on top of the block, so an adversary cannot withdraw her funds before the execution of the committed transaction. This simple approach can be implemented in target blockchains that are willing

to modify their consensus mechanism. By default, consensus-layer MEV prevention mechanisms offer this by only executing encrypted transactions or executing encrypted transactions before normal transactions. However, making such changes in popular public blockchains such as Ethereum [105] would be very challenging in practice.

- **Security deposit:** Users can lock a certain amount as a deposit to a smart contract, which can be slashed in case of such attacks. The smart contract is hard-coded to return the deposit after execution of the protocol without such adversarial behaviour of the user. This deposit makes the bar to the profitability of this attack even higher; however, there is no limit to the profit of the attacker and there may be a situation where the profit is higher than the sum of the deposit and additional fees such as the high price to front-run the whole batch.

- **Locked transaction value and fees:** Users can have relayer accounts with enough balance for their transaction's amount and transaction fees which cannot be withdrawn directly before the execution of their committed transactions. Consequently, target applications can make sure the users' transactions will go through even if the sender's wallet is empty. Another similar approach to achieve this is through locking the transaction fees and amount in a smart contract (which can be the commit contract itself) during the process.

Chapter 5

Implementation and Evaluation

5.1 Implementation details

5.1.1 FairBlock prototype implementations

We have built prototype implementations of FairBlock for both consensus-level and smart contract-based approaches. Smart contracts are implemented in Solidity and consensus-level blockchain is built based on Cosmos SDK [34] in Go. For consensus-level implementation, validators can submit their block key shares as a message in the FairBlock blockchain, and other parties can compute the block key and submit decrypted transactions as a message to collect rewards. However, it is worth mentioning that an as alternative to sending the block key shares via messages in the public mempool, validators can send their block

Table 5.1: Mean values of distributed IBE execution time

Keepers	Block key extraction (ms)	Decryption (ms)	Encryption (ms)
5	8.07 ± 0.05	1.57 ± 0.04	5.29 ± 0.03
10	16.97 ± 0.06	1.54 ± 0.04	5.24 ± 0.02
20	29.5 ± 0.10	1.50 ± 0.02	5.21 ± 0.01
50	72.91 ± 0.18	1.52 ± 0.04	5.22 ± 0.02
100	147.39 ± 0.29	1.60 ± 0.07	5.28 ± 0.04
200	294.90 ± 0.63	1.53 ± 0.04	5.30 ± 0.02
500	771.72 ± 1.38	1.59 ± 0.03	5.35 ± 0.03

key shares as an extension to the voting round in a BFT-style consensus algorithm [21, 92]. This alternative approach can be implemented in the FairBlock blockchain after the release of ABCI++ [33] which allows validators in a Cosmos-based blockchain to extend their votes in the consensus voting phase with their shares of block key [49, 92].

Both implementations can be readily employed for auctions, gaming, and various other DeFi use cases. Moreover, other PoS blockchain networks including Cosmos ecosystem, Ethereum Beacon chain, Avalanche, and Solana [47, 32, 4, 93] can also prevent front-running in their network by integrating FairBlock in their consensus mechanism. The source code of FairBlock including distributed IBE and smart contracts is available on GitHub[1]. The source code of FairBlock implementation in the consensus layer is also available on GitHub[2]. Our implementation of the distributed IBE is built on top of the Vuvuzela cryptography library [101] in Go and assembly. For simplicity, we have described FairBlock using symmetric pairings with the same source groups. However, we have implemented our protocol using type 3 pairings (BLS12-381) with different source groups for better efficiency as the Boneh-Franklin BasicIdent IBE [16] can also be described with type 3 pairings [18].

5.1.2 Distributed key generation implementation

In FairBlock, we have employed the described JF-DKG [53] due to its simplicity and performance. We have developed our DKG protocol based on the open-source Shutter's implementation [91] of this scheme. However, this DKG scheme may be replaced with implementations and schemes such as [65, 58, 88] to achieve better properties. Specifically, Ferveo [49] claims that their cryptographic primitives and synchrony assumptions enable them to enjoy less time complexity of Agg-DKG while generating secret key shares in the scalar field which makes it suitable for threshold cryptography schemes such as IBE. As a result, this implementation can be also used in the future after further scrutiny of their unpublished scheme.

5.1.3 Meta-transaction implementation

As discussed in Section 4.6.1, we need to send a single transaction which contains all the ordered decrypted transactions to prevent front-running even for legacy target applications (similar to roll-ups in nature). However, blockchain networks such as Ethereum [105] do

[1]https://github.com/pememoni/FairBlock-SC
[2]https://github.com/pememoni/FairBlock

not have native support of batching, and we need to use two additional components: a) relayer (or batcher) and b) dispatcher. Relayer is an off-chain component that is responsible for batching multiple transactions, and dispatcher is an on-chain component which parses the single transaction and executes each of the included meta-transactions. In this work, relayers simply concatenate decrypted meta-transactions and put them in the data field of a standard transaction. For dispatching, we simply unmarshall the data field and call the intended target contract for each of the meta-transactions in a loop. Listing 5.1.3 shows our simple Solidity implementation for the dispatcher component. For further optimization, more sophisticated solutions such as iBatch [102] and Gnosis Mutlisend [54] can be also employed for achieving better properties such as saving more fees.

Listing 5.1: Implementation of meta-transaction processing

```
pragma solidity ^0.8.10;

contract MultiCall {
    struct Transaction {
        address targetContract;
        bytes data;
        uint256 value;
    }

    function multiCall(Transaction[] memory txs)
        public
        returns (bytes[] memory returnData)
    {
        returnData = new bytes[](txs.length);
        for (uint256 i = 0; i < txs.length; i++) {
            (bool success, bytes memory result)
            = txs[i].targetContract.call(txs[i].data);
            require(success);
            returnData[i] = result;
        }

        return (returnData);
    }
}
```

5.1.4 User interface implementation

We have developed a simple front-end interface for users that receives plaintext information about their desired transaction. In particular, we have used CryptID [103] library to encrypt the transaction information automatically on the client-side using mpk and h. Due to the high cost of on-chain stored data in designs that are relying on Commit, we have also developed the option to store encrypted transactions on the IPFS network [12] using Infura IPFS API [62]. To be specific, all of the transaction content including the encrypted transaction and cryptographic commitments will be uploaded to IPFS in JSON format. A content identifier (CID) which is a hash of the JSON file can be simply used as a reference to access the uploaded transaction content. Consequently, the CID and desired block range for decryption will be automatically signed by the user's wallet, i.e., MetaMask [72] and sent to Commit as a transaction. After finalization of the block, relayers read submitted CIDs for the desired block range, and fetch them from IPFS for subsequent decryption and execution. As the committed transactions on IPFS will be executed in minutes (in the worst case), IPFS's automatic garbage collection in every hour does not delete uploaded data before execution. Although IPFS is free, a better execution and persistency guarantee can be achieved nodes' pinning service for a negligible fee. The source code of this prototype user interface is available on GitHub[3].

5.2 Performance evaluation

To measure the performance of our Distributed IBE implementation, we use a 2nd Gen Intel Xeon 2.50 GHz server with 1 core and 2 GB of RAM. In order to determine an average performance, we ran the experiments 100 times for each keepers set size. We tested the implementation for systems of up to 500 keepers and present average execution times in Table 5.1 along with 95% two-sided confidence intervals. Our results show the feasibility of our basic implementation using basic hardware resources for even the fastest proof-of-stake and proof-of-work public blockchains. For instance, the average block key extraction time (composed of block key shares aggregation, verification, and block key computation) for 100 keepers is 147.39 ms which is significantly less than the block finalization time of PoW blockchains such as Ethereum (12-14 seconds), and current fastest PoS blockchain namely Avalanche [4] (1-3 seconds). We have also measured the encryption and decryption execution time of random 256 byte messages for 1000 runs. On average, decryption takes 1.54 ms and encryption takes 5.27 ms which are negligible compared to block key extraction

[3]https://github.com/pememoni/FairFace

Table 5.2: Comparison of bandwidth overhead in IBE and threshold decryption

System size		Bandwidth overhead	
Transactions	Keepers	Identity-Based Encryption	Threshold Decryption
1000	1000	170.8 KB	42.7 MB
100	100	17.2 KB	326.4 KB

time and can be easily parallelized with the same execution time. For larger message sizes, our work employs hybrid encryption [42]. Using hybrid encryption, Identity-Based Encryption is used to encrypt a key and an efficient symmetric encryption scheme such as AES-GCM or ChaCha20 [27] is used to encrypt the actual transaction with the key.

We have compared the bandwidth overhead of FairBlock and the threshold decryption approach in two realistic scenarios. In scenario I, there are 1000 keepers and 1000 encrypted transactions that should be decrypted every 24 hours. In scenario II, there are 1000 keepers and 100 transactions to be decrypted in 10 seconds. Using IBE, we need at least two-thirds of keepers to send their shares (of size 256 byte in our implementation) for the block key extraction. In threshold decryption, at least two-thirds of keepers should compute partial decryptions (of size 64 byte in our implementation) for each of the committed transactions. Table 5.2 shows the result of our experiment. In scenario I, the total message size of the IBE approach is only 0.4% of the threshold decryption approach. Similarly, the total message size of the IBE approach is approximately 25 times less than the other approach in scenario II.

5.3 Comparison and discussion

In this section, we first summarize the differences of our three proposed architectures in Section 4.3, and then compare the general properties of our FairBlock technique with other commit-reveal approaches. In short, the smart-contract based instantiation does not rely on a side-chain or modification to the consensus layer and is suitable for dApps on any smart-contract blockchain or other blockchain architectures such as cross-chain protocols, and roll-ups. However, due to the high cost of storage and slow communication on popular blockchain such as Ethereum [105], we proposed the oracle-based instantiation which significantly reduces these costs while still being suitable for dApps on any smart-contract blockchain such as cross-chain protocols, and roll-ups. It also facilities the incentivization and staking mechanisms for keepers. The smart-contract based and the oracle-base in-

Table 5.3: Comparison of commit-reveal mechanisms

	MEV security	Execution delay	Consensus overhead	Hardware dependency
Secure enclave	Cost to break one secure enclave	1 block delay	Delays block proposing	Requires secure enclave
Timelock encryption	Trade-off against delay	T blocks (Trade-off against security)	Negligible	Additional hardware requirements for decryption
Threshold decryption	Cost to break consensus, DKG, or encryption scheme	1 block delay	Quadratic bandwidth complexity	No additional hardware
FairBlock	Cost to break consensus, DKG, or encryption scheme	1 block delay	Linear bandwidth complexity	No additional hardware

stantiation require a minimal modification in dApps to accept transaction only through FairBlock to prevent front-running with direct transaction. The consensus-layer instantiation does not require the modification in dApps, and the whole process happens seamlessly for users and dApps. However, alongside the technical challenges for adoption of this approach in popular blockchain networks, this instantiation is mainly suitable for PoS blockchains and not other blockchain architectures.

We also provide a detailed comparison of other commit-reveal approaches with our work in Table 5.3 with respect to the following security properties [92]:

- **MEV security:** The cost to break the security of the commit-reveal system with the goal of leaking information about encrypted transactions and front-running them.
- **Execution delay:** The resulted delay in execution of committed transactions compared to the normal flow of execution in the similar but unprotected architecture.
- **Consensus overhead:** Complexity of the resulted overhead in communication and block processing.
- **Hardware dependency:** Dependency of the approach to additional hardware.

In summary, FairBlock's security does not rely on hardware for VDF evaluation [14] or secure enclave [107]. This makes FairBlock and the threshold decryption approach cheaper and more scalable in the sense that players do not need access to a specific hardware [67, 30, 44, 100, 94, 76]. Moreover, security of FairBlock and the threshold decryption approach is not dependent on the security of the hardware which normally has a complex trade-off

with execution delay and a history of security breaches [99, 98]. Further, while the standard threshold decryption approach has a scalability issue with a large number of transactions, FairBlock's linear bandwidth overhead can afford larger transaction batches and keeper sets. One common drawback among all commit-reveal approaches is the execution delay which may be unpleasant for certain applications in an underlying blockchain with a high block finality time where users expect faster finality for their orders. However, ordering the committed transactions on a faster side-chain can mitigate this drawback by reducing the execution delay to the side chain's block finality time.

Chapter 6

Future Work and Challenges

6.1 MEV prevention in layer two solutions

One of the main challenges of current public blockchains such as Ethereum is their slow finality and transaction throughput [105]. The most promising scalability solution is perhaps through rollups [23, 79] which execute a batch of transactions off-chain (in a roll-up specific blockchain), compress transactions' data, and send an update to the underlying layer one blockchain as a single transaction. In this approach, a centralized operator or a decentralized network of operators named sequencer is responsible for ordering the batch of transactions, thereby offering it the opportunity to perform front-running attacks by the sequencer. Although layer 2 solutions and rollups vary in their implementations and architectures, the same FairBlock technique can be applied to them for preventing malicious front-running attacks. To be more specific, the sequencer receives encrypted transactions, batches them, and commits to the batch by signing the hash of the batch. Next, keepers send their key shares and the decryption key can be extracted and used to decrypt the transactions for building the rollup. The sequencer cannot effectively front-run the batch since it has committed to the included transactions and their order without any knowledge of their contents [91]. Chainlink [28] has proposed a decentralized oracle network for ordering transactions with respect to the notion of fairness defined in [66]. However, this fair ordering prevention does not guarantee front-running prevention completely and can be combined with FairBlock for stronger fairness guarantees. Moreover, FairBlock and similar techniques such as [91] eliminate the need for high-latency decentralized sequencers for censorship prevention by preventing censorship just by using a single sequencer.

6.2 Cross-chain MEV

In recent years, several independent blockchain networks such as Ethereum [105], Cosmos [32], and Avalanche [4] with distinct and unique functionalities have emerged in the blockchain ecosystem. Although validators and miners in these networks are theoretically assumed to be independent, extremely profitable opportunities in these domains have resulted in many common or colluding players. Furthermore, decentralized cross-chain protocols such as Axelar [5], Layer Zero [108], and Wormhole [106] have connected these networks with cross-chain message passing and cross-chain asset transfer functionalities. These common players and cross-chain functionalities will open the door to the complex world of cross-chain MEV by connecting the states of all networks in the ecosystem. Obadia et al. [78] have formalized cross-chain MEV opportunities and the resulted negative externalities in the blockchain ecosystem. In the simplest cross-chain MEV scenario, a single validator (or two colluding validators) which is responsible for sequencing the transactions in two blocks of different networks, can order transactions with respect to each other to extract additional profit. As a negative effect, the potential high profit in a network can incentivize a party to sabotage the consensus in the other network.

In addition to the described cross-chain arbitrage and sandwiching opportunities, we also envision further cross-chain MEV opportunities, particularly in the implementation level of cross-chain infrastructures. For instance, once cross-chain transfers are approved in Axelar network [5], relayers who are responsible for posting the verified transfer transactions in the target blockchain can arbitrarily reorder them in order to extract profits with minimal risks. Moreover, due to the fact that cross-chain transactions should be verified by validators in another network e.g. Axelar network, transactions will be pending in clear for a longer period of time resulting in more front-running opportunities. Cross-chain MEV can be mitigated using commit-reveal approaches such as FairBlock in different ways. First, users can commit to encrypted transactions in different domains prior to a shared deadline to be executed after the finalization of blocks in different networks. However, technical complexities including synchronization of the deadline through possible approaches such as timestamps or cross-chain messaging needs to be examined more rigorously [49, 78]. Second, FairBlock can also mitigate front-running in bridges and decentralized cross-chain protocols. To be more specific, these networks can initially finalize the order of cross-chain transferred data on encrypted data and then reveal actual transferred data to the target blockchains. As a result, potential front-runners such as relayers or validators cannot abuse their information asymmetry to extract profit by reordering the verified transactions or sandwiching them. We leave exploring this approach's actual performance and practical trade-offs as future work.

6.3 Multi-party Computation

Recent interest in applications such as federated learning [70] has rapidly transformed multi-party computation (MPC) from a purely theoretical approach to an object of interest for practical privacy-preserving applications [111]. In short, MPC allows multiple parties to jointly compute a function while keeping their individual inputs private. Baum et al. [11] have proposed using insured MPC [10] to match multiple exchange orders off-chain while keeping the transactions private. This work also offers a prototype implementation using Fresco [63] to show the feasibility of front-running prevention using MPC. However, this work is still very limited for practical applications in DeFi. In particular, the online execution time for matching 128 orders using only three servers can take up to two seconds.

In a concurrent work, Govindarajan [55] et al. have also employed MPC to address the same problem and improved the former work by supporting order rate confidentiality as well. However, this work is not currently practical for many applications in the blockchain space due to the same high latency and low decentralization limitations e.g. 6 seconds latency for matching 128 orders on three servers. These limitations make these works impractical for many target applications due to: a) fast block finality times (1-5 seconds) in blockchains such as Avalanche and Cosmos, and b) centralization threat due to the few number of servers. These drawbacks cannot be addressed easily since there is a trade-off between servers and computation time. Moreover, these solutions only prevent front-running in DEXes (which are just a subset of dApps) with an order-book matching mechanism which has been replaced by automatic market makers (AMM) in most of the popular DEXes such as Uniswap [1].

Implementing practical MPC libraries (in terms of less latency and more decentralization) in blockchains and exploring possible approaches to prevent front-running in general decentralized applications can be promising for future work.

Chapter 7

Conclusion

In this thesis, we studied the problem of MEV and front-running attacks in the context of blockchains. We motivated the problem by defining and describing front-running strategies which have been notoriously harmful in the blockchain ecosystem. Followed by a discussion on limitations of current approaches, we designed and implemented FairBlock, the first front-running prevention mechanism based on distributed IBE. Our work addresses many limitations of previous front-running mechanisms. Specifically, FairBlock significantly outperforms the most well-known approach based on threshold decryption in bandwidth overhead. We have implemented and evaluated our prototype using both smart contracts and consensus-layer as the communications layer. The source code of our implementation is open-sourced.

References

[1] Hayden Adams, Noah Zinsmeister, Moody Salem, River Keefer, and Dan Robinson. Uniswap v3 core. Technical report, Tech. rep., Uniswap, 2021.

[2] Avi Asayag, Gad Cohen, Ido Grayevsky, Maya Leshkowitz, Ori Rottenstreich, Ronen Tamari, and David Yakira. A fair consensus protocol for transaction ordering. In *2018 IEEE 26th International Conference on Network Protocols (ICNP)*, pages 55–65, 2018.

[3] Raphael Auer, Jon Frost, and Jose Maria Vidal Pastor. Miners as intermediaries: extractable value and market manipulation in crypto and defi. https://www.bis.org/publ/bisbull58.htm. (Accessed on 07/07/2022).

[4] Avalanche whitepaper. https://www.avalabs.org/whitepapers. (Accessed on 12/03/2021).

[5] Secure cross-chain communication for web3. https://axelar.network/. (Accessed on 07/01/2022).

[6] Kushal Babel, Philip Daian, Mahimna Kelkar, and Ari Juels. Clockwork finance: Automated analysis of economic security in smart contracts, 09 2021.

[7] Adam Back et al. Hashcash-a denial of service counter-measure, 2002.

[8] Joonsang Baek and Yuliang Zheng. Simple and efficient threshold cryptosystem from the gap diffie-hellman group. In *GLOBECOM '03. IEEE Global Telecommunications Conference (IEEE Cat. No.03CH37489)*, volume 3, pages 1491–1495 vol.3, 2003.

[9] Massimo Bartoletti, James Hsin-yu Chiang, and Alberto Lluch-Lafuente. Maximizing extractable value from automated market makers. *arXiv preprint arXiv:2106.01870*, 2021.

[10] Carsten Baum, Bernardo David, and Rafael Dowsley. Insured mpc: Efficient secure computation with financial penalties. In *Financial Cryptography and Data Security: 24th International Conference, FC 2020 , Kota Kinabalu, Malaysia, February 10–14, 2020 Revised Selected Papers*, page 404–420, Berlin, Heidelberg, 2020. Springer-Verlag.

[11] Carsten Baum, Bernardo David, and Tore Kasper Frederiksen. P2dex: privacy-preserving decentralized cryptocurrency exchange. In *International Conference on Applied Cryptography and Network Security*, pages 163–194. Springer, 2021.

[12] Juan Benet. Ipfs powers the distributed web. https://ipfs.io/. (Accessed on 06/24/2022).

[13] Shaileshh Bojja Venkatakrishnan, Giulia Fanti, and Pramod Viswanath. Dandelion: Redesigning the bitcoin network for anonymity. *Proc. ACM Meas. Anal. Comput. Syst.*, 1(1), jun 2017.

[14] Dan Boneh, Joseph Bonneau, Benedikt Bünz, and Ben Fisch. Verifiable delay functions. In *Annual international cryptology conference*, pages 757–788. Springer, 2018.

[15] Dan Boneh and Xavier Boyen. Efficient selective-id secure identity-based encryption without random oracles. In *International conference on the theory and applications of cryptographic techniques*, pages 223–238. Springer, 2004.

[16] Dan Boneh and Matt Franklin. Identity-based encryption from the weil pairing. In *Annual international cryptology conference*, pages 213–229. Springer, 2001.

[17] Dan Boneh, Amit Sahai, and Brent Waters. Functional encryption: Definitions and challenges. In *Theory of Cryptography Conference*, pages 253–273. Springer, 2011.

[18] Xavier Boyen. A tapestry of identity-based encryption: practical frameworks compared. *International Journal of Applied Cryptography*, 1(1):3–21, 2008.

[19] Gilles Brassard, David Chaum, and Claude Crépeau. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences*, 37(2):156–189, 1988.

[20] Lorenz Breidenbach, Philip Daian, Florian Tramèr, and Ari Juels. Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. Cryptology ePrint Archive, Report 2017/1090, 2017.

[21] Ethan Buchman, Jae Kwon, and Zarko Milosevic. The latest gossip on bft consensus. *arXiv preprint arXiv:1807.04938*, 2018.

[22] Vitalik Buterin. EIP-86: Abstraction of transaction origin and signature. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-86.md. (Accessed on 12/03/2021).

[23] Vitalik Buterin. A rollup-centric ethereum roadmap. https://ethereum-magicians.org/t/a-rollup-centric-ethereum-roadmap/4698. (Accessed on 06/21/2022).

[24] Vitalik Buterin, Eric Conner, Rick Dudley, Matthew Slipper, Ian Norden, and Abdelhamid Bakhta. Ethereum eip-1559. https://github.com/ethereum/EIPs/blob/master/EIPS/eip-1559.md. (Accessed on 06/28/2022).

[25] Benedikt Bünz, Shashank Agrawal, Mahdi Zamani, and Dan Boneh. Zether: Towards privacy in a smart contract world. In *International Conference on Financial Cryptography and Data Security*, pages 423–443. Springer, 07 2020.

[26] Christian Cachin, Klaus Kursawe, Frank Petzold, and Victor Shoup. Secure and efficient asynchronous broadcast protocols. In *Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '01, page 524–541, Berlin, Heidelberg, 2001. Springer-Verlag.

[27] Chacha20 and poly1305 for ietf protocols. https://www.rfc-editor.org/rfc/rfc7539.txt. (Accessed on 04/03/2022).

[28] Chainlink 2.0 and the future of decentralized oracle networks — chainlink. https://chain.link/whitepaper, 2021.

[29] Michele Ciampi, Muhammad Ishaq, Malik Magdon-Ismail, Rafail Ostrovsky, and Vassilis Zikas. Fairmm: A fast and frontrunning-resistant crypto market-maker. *Cryptology ePrint Archive*, 2021.

[30] Dan Cline, Thaddeus Dryja, and Neha Narula. Clockwork: An exchange protocol for proofs of non front-running, 2020.

[31] Clifford Cocks. An identity based encryption scheme based on quadratic residues. In *IMA international conference on cryptography and coding*, pages 360–363. Springer, 2001.

[32] Cosmos: The internet of blockchains. https://cosmos.network/. (Accessed on 03/18/2022).

[33] Abci++. https://github.com/tendermint/spec/blob/master/rfc/004-abci++.md. (Accessed on 04/03/2022).

[34] Cosmos sdk - cosmos network. https://v1.cosmos.network/sdk. (Accessed on 03/26/2022).

[35] Cowswap - meta dex aggregator. https://cowswap.exchange/. (Accessed on 12/03/2021).

[36] Philip Daian. Mev... wat do? https://pdaian.com/blog/mev-wat-do/. (Accessed on 07/06/2022).

[37] Philip Daian, Steven Goldfeder, Tyler Kell, Yunqi Li, Xueyuan Zhao, Iddo Bentov, Lorenz Breidenbach, and Ari Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. *arXiv preprint arXiv:1904.05234*, 2019.

[38] Dao — aragon. https://aragon.org/dao. (Accessed on 04/01/2022).

[39] Yvo Desmedt and Yair Frankel. Threshold cryptosystems. In *Conference on the Theory and Application of Cryptology*, pages 307–315. Springer, 1989.

[40] Yvo Desmedt and Sushil Jajodia. Redistributing secret shares to new access structures and its applications, 1997.

[41] Roger Dingledine, Nick Mathewson, and Paul Syverson. Tor: The second-generation onion router. Technical report, Naval Research Lab Washington DC, 2004.

[42] Pooja Dixit, Avadhesh Kumar Gupta, Munesh Chandra Trivedi, and Virendra Kumar Yadav. Traditional and hybrid encryption techniques: a survey. In *Networking communication and data knowledge engineering*, pages 239–248. Springer, 2018.

[43] Shlomi Dolev, Karim ElDefrawy, Joshua Lampkins, Rafail Ostrovsky, and Moti Yung. Brief announcement: Proactive secret sharing with a dishonest majority. In *Proceedings of the 2016 ACM Symposium on Principles of Distributed Computing*, pages 401–403, 2016.

[44] Yael Doweck and Ittay Eyal. Multi-party timed commitments, 2020.

[45] Sisi Duan, Michael K. Reiter, and Haibin Zhang. Secure causal atomic broadcast, revisited. In *2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*, pages 61–72, 2017.

[46] Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark. Sok: Transparent dishonesty: front-running attacks on blockchain, 2019.

[47] Ethereum upgrades. https://ethereum.org/en/upgrades/. (Accessed on 03/18/2022).

[48] P. Feldman. A practical scheme for non-interactive verifiable secret sharing. In *28th Annual Symposium on Foundations of Computer Science (sfcs 1987)*, pages 427–438, 1987.

[49] Ferveo. https://anoma.network/blog/ferveo-a-distributed-key-generation-scheme-for-t (Accessed on 12/03/2021).

[50] Eiichiro Fujisaki and Tatsuaki Okamoto. Secure integration of asymmetric and symmetric encryption schemes. In *Advances in Cryptology - CRYPTO '99*, volume 1666 of *Lecture Notes in Computer Science*, pages 537–554. Springer, 1999.

[51] Gas and fees — ethereum. https://ethereum.org/en/developers/docs/gas/. (Accessed on 06/07/2022).

[52] Rosario Gennaro, Stanislaw Jarecki, and Hugo Krawczyk. Revisiting the distributed key generation for discrete-log based cryptosystems, 12 2002.

[53] Rosario Gennaro, Stanislaw Jarecki, Hugo Krawczyk, and Tal Rabin. Secure distributed key generation for discrete-log based cryptosystems. *Journal of Cryptology*, 20(1):51–83, 2007.

[54] Gnosis. Multisend: An npm package for crafting multi-send transaction. https://github.com/gnosis/ethers-multisend. (Accessed on 07/11/2022).

[55] Kavya Govindarajan, Dhinakaran Vinayagamurthy, Praveen Jayachandran, and Chester Rebeiro. Privacy-preserving decentralized exchange marketplaces, 11 2021.

[56] Jens Groth. Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339, 2021.

[57] Gas station network. https://docs.opengsn.org/, month = , year = note = (Accessed on 04/03/2022).

[58] Kobi Gurkan, Philipp Jovanovic, Mary Maller, Sarah Meiklejohn, Gilad Stern, and Alin Tomescu. Aggregatable distributed key generation. In *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, pages 147–176. Springer, 2021.

[59] Shai Halevi. Efficient commitment schemes with bounded sender and unbounded receiver. In *Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology*, CRYPTO '95, page 84–96, Berlin, Heidelberg, 1995. Springer-Verlag.

[60] Shai Halevi and Silvio Micali. Practical and provably-secure commitment schemes from collision-free hashing. In Neal Koblitz, editor, *Advances in Cryptology — CRYPTO '96*, pages 201–215, 1996.

[61] Amir Herzberg, Stanisław Jarecki, Hugo Krawczyk, and Moti Yung. Proactive secret sharing or: How to cope with perpetual leakage. In *Advances in Cryptology — CRYPT0' 95*, pages 339–352, Berlin, Heidelberg, 1995. Springer Berlin Heidelberg.

[62] Infura. Ethereum and ipfs api. https://infura.io/. (Accessed on 06/24/2022).

[63] Alexandra Institute. Fresco: A framework for efficient secure computation. https://github.com/aicis/fresco. (Accessed on 06/23/2022).

[64] Aljosha Judmayer, Nicholas Stifter, Philipp Schindler, and Edgar Weippl. Estimating (miner) extractable value is hard, let's go shopping! Cryptology ePrint Archive, Paper 2021/1231, 2021.

[65] Aniket Kate and Ian Goldberg. Distributed private-key generators for identity-based cryptography. In *Security and Cryptography for Networks*, pages 436–453. Springer Berlin Heidelberg, 2010.

[66] Mahimna Kelkar, Fan Zhang, Steven Goldfeder, and Ari Juels. Order-fairness for byzantine consensus. In *Annual International Cryptology Conference*, pages 451–480. Springer, 2020.

[67] Rami Khalil, Arthur Gervais, and Guillaume Felley. TEX - a securely scalable trustless exchange. *IACR Cryptol. ePrint Arch.*, page 265, 2019.

[68] Offchain Labs. Arbitrum rollup. https://developer.offchainlabs.com/docs/rollup_basics. (Accessed on 06/21/2022).

[69] Benoît Libert and Jean-Jacques Quisquater. Identity based encryption without redundancy. In *International Conference on Applied Cryptography and Network Security*, pages 285–300. Springer, 2005.

[70] Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas. Communication-efficient learning of deep networks from decentralized data. In *Artificial intelligence and statistics*, pages 1273–1282. PMLR, 2017.

[71] Alfred Menezes. An introduction to pairing-based cryptography, 2005.

[72] MetaMask. The crypto wallet for defi, web3 dapps and nfts. https://metamask.io/. (Accessed on 06/24/2022).

[73] Mev-Explore. https://explore.flashbots.net/. (Accessed on 12/03/2021).

[74] Peyman Momeni, Sergey Gorbunov, and Bohan Zhang. Fairblock: Preventing blockchain front-running with minimal overheads. In *International Conference on Security and Privacy in Communication Systems*, page in press. Springer, 2022.

[75] Moni Naor. Bit commitment using pseudorandomness. *J. Cryptol.*, 4(2):151–158, jan 1991.

[76] Secret Network. Secret markets: Front running prevention for automated market makers. https://scrt.network/blog/secret-markets-front-running-prevention. (Accessed on 06/22/2022).

[77] Shen Noether. Ring signature confidential transactions for monero. *IACR Cryptol. ePrint Arch*, page 1098, 2015.

[78] Alexandre Obadia, Alejo Salles, Lakshman Sankar, Tarun Chitra, Vaibhav Chellani, and Philip Daian. Unity is strength: A formalization of cross-domain maximal extractable value. *arXiv preprint arXiv:2112.01472*, 2021.

[79] Optimism. https://www.optimism.io/. (Accessed on 06/21/2022).

[80] Osmosis. https://app.osmosis.zone/. (Accessed on 12/03/2021).

[81] Torben Pryds Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Annual international cryptology conference*, pages 129–140. Springer, 1991.

[82] Torben Pryds Pedersen. A threshold cryptosystem without a trusted party. In Donald W. Davies, editor, *Advances in Cryptology — EUROCRYPT '91*, pages 522–526, Berlin, Heidelberg, 1991. Springer Berlin Heidelberg.

[83] HD Phaneendra et al. Identity-based cryptography and comparison with traditional public key encryption: A survey. *International Journal of Computer Science and Information Technologies*, 5(4):5521–5525, 2014.

[84] Vega Protocol. Blockchain derivatives. https://vega.xyz/. (Accessed on 06/22/2022).

[85] Kaihua Qin, Liyi Zhou, and Arthur Gervais. Quantifying blockchain extractable value: How dark is the forest?, 2021.

[86] Michael K Reiter and Kenneth P Birman. How to securely replicate services. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):986–1009, 1994.

[87] Dan Robinson and Georgios Konstantopoulos. Ethereum is a dark forest - paradigm. https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest/. (Accessed on 12/03/2021).

[88] Philipp Schindler, Aljosha Judmayer, Nicholas Stifter, and Edgar Weippl. Ethdkg: Distributed key generation with ethereum smart contracts. Cryptology ePrint Archive, Report 2019/985, 2019.

[89] Adi Shamir. How to share a secret. *Commun. ACM*, 22(11):612–613, 1979.

[90] Adi Shamir. Identity-based cryptosystems and signature schemes. In *Workshop on the theory and application of cryptographic techniques*, pages 47–53. Springer, 1984.

[91] Shutter Network. https://shutter.ghost.io/. (Accessed on 12/03/2021).

[92] Sikka. https://sikka.tech/projects/. (Accessed on 12/03/2021).

[93] Solana. https://solana.com/. (Accessed on 03/18/2022).

[94] Chrysoula Stathakopoulou, Signe Rüsch, Marcus Brandenburger, and Marko Vukolic. Adding fairness to order: Preventing front-running attacks in bft protocols using tees.

[95] Sushiswap. https://sushi.com/. (Accessed on 12/03/2021).

[96] Alin Tomescu, Robert Chen, Yiming Zheng, Ittai Abraham, Benny Pinkas, Guy Golan Gueta, and Srinivas Devadas. Towards scalable threshold cryptosystems. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 877–893. IEEE, 2020.

[97] Tornado Cash. https://tornado.cash/. (Accessed on 12/05/2021).

[98] Jo Van Bulck, Marina Minkin, Ofir Weisse, Daniel Genkin, Baris Kasikci, Frank Piessens, Mark Silberstein, Thomas F Wenisch, Yuval Yarom, and Raoul Strackx. Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In *27th USENIX Security Symposium (USENIX Security 18)*, pages 991–1008, 2018.

[99] Stephan van Schaik, Andrew Kwong, Daniel Genkin, and Yuval Yarom. SGAxe: How SGX fails in practice, 2020.

[100] Veedo. https://github.com/starkware-libs/veedo. (Accessed on 03/30/2022).

[101] vuvuzela cryptography libraries. https://github.com/vuvuzela/crypto. (Accessed on 04/03/2022).

[102] Yibo Wang, Qi Zhang, Kai Li, Yuzhe Tang, Jiaqi Chen, Xiapu Luo, and Ting Chen. iBatch: saving ethereum fees via secure and cost-effective batching of smart-contract invocations. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, aug 2021.

[103] WebAssembly. Browser-friendly identity-based encryption library powered by webassembly. https://github.com/cryptid-org/cryptid-js. (Accessed on 06/24/2022).

[104] Theodore M Wong, Chenxi Wang, and Jeannette M Wing. Verifiable secret redistribution for threshold sharing schemes. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA SCHOOL OF COMPUTER SCIENCE, 2002.

[105] Gavin Wood. Ethereum: A secure decentralized generalized transaction ledger, 2014.

[106] Wormhole. https://wormholenetwork.com/. (Accessed on 07/01/2022).

[107] Bin Cedric Xing, Mark Shanahan, and Rebekah Leslie-Hurd. Intel software guard extensions (Intel SGX) software support for dynamic memory allocation inside an enclave. In *Proceedings of the Hardware and Architectural Support for Security and Privacy 2016*, New York, NY, USA, 2016. Association for Computing Machinery.

[108] Ryan Zarick, Bryan Pellegrino, and Caleb Banister. Layerzero: Trustless omnichain interoperability protocol. *arXiv preprint arXiv:2110.13871*, 2021.

[109] Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galinanes, and Bryan Ford. F3b: A low-latency commit-and-reveal architecture to mitigate blockchain front-running. *arXiv preprint arXiv:2205.08529*, 2022.

[110] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 633–649. USENIX Association, November 2020.

[111] Chuan Zhao, Shengnan Zhao, Minghao Zhao, Zhenxiang Chen, Chong-Zhi Gao, Hongwei Li, and Yu-an Tan. Secure multi-party computation: theory, practice and applications. *Information Sciences*, 476:357–372, 2019.

[112] Liyi Zhou, Kaihua Qin, Christof Ferreira Torres, Duc Viet Le, and Arthur Gervais. High-frequency trading on decentralized on-chain exchanges. *2021 IEEE Symposium on Security and Privacy (SP)*, pages 428–445, 2021.