FairBlock: Preventing Blockchain Front-running with Minimal Overheads

Peyman Momeni[1,2], Sergey Gorbunov[1,2], and Bohan Zhang[1]

[1] University of Waterloo, Waterloo, Canada
{pmomeni,b327zhan,sgorbunov}@uwaterloo.ca
[2] Axelar Network
[3] FairBlock Network

Abstract. While blockchain systems are quickly gaining popularity, front-running remains a major obstacle to fair exchange. In this paper, 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 and build a prototype implementation. We show that on a sample of 1000 messages with 1000 validators our system saves 42.53 MB of bandwidth which is 99.6% less compared with the standard threshold decryption paradigm.

Keywords: Blockchain · Front-running · DeFi · Identity-Based Encryption · Smart contract · Security

1 Introduction

Maximal (or Miner) Extractable Value (MEV) is one of the central problems that prevent fairness [39, 71] and trust in decentralized exchanges and other decentralized applications (dApps) [24, 29, 72, 53, 5]. 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.

This paper is accepted in 18th EAI International Conference on Security and Privacy in Communication Networks [43]

MEV is defined as the revenue other than transaction fees and block rewards which can be extracted by reordering, censoring, and adding transactions in blocks [24, 29, 72, 53, 5]. MEV is present on any blockchain infrastructure that includes a party that is responsible for transaction ordering such as miners in Ethereum [68], validators in Cosmos [22], or sequencers in Layer 2 solutions such as roll-ups [47, 42]. 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 [42, 51, 68]. These attacks lead to serious problems such as high gas fees, network congestion, and even consensus instability [24].

Threshold decryption schemes are one of the most promising and well-known methods to prevent front-running [17, 32]. The idea was proposed in 1994 [52], and recently explored by blockchain projects such as Sikka, F3, and Anoma [32, 59, 2, 70]. 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 for limitations of other front-running prevention mechanisms.

1.1 Our Contributions

In this work, 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 [9, 57, 19]. In FairBlock, a committee composed of keepers that run a distributed key generation (DKG) [49, 33] 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 smart contract blockchains. 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 [29, 17].

Compared to the solutions [40, 18, 27, 66] that leverage time-lock puzzles [7], we do not introduce significant delays or high computational complexity in decryption. Moreover, our work does not rely on secure enclaves [69] to realize a private pool [61]. 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.

1.2 Paper Organization

The remainder of the paper is organized as follows. In Section 2, we describe related works and their limitations. In Section 3, we review the cryptographic building blocks of FairBlock and define blockchain front-running. In Section 4, we present our security model, followed by describing FairBlock protocol and details of our architecture. Section 5 describes our prototype implementation and evaluation. We also indicate future research directions and challenges in Section 6, before concluding in Section 7.

2 Related Works

Several academic works and projects have attempted to either limit or prevent front-running. For instance, Flashbots [24] 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 [12] 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, 62], as the main target of front-running [42, 5] 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 [23] 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.

Recent projects including Secret Network [45] and Fairy [61] leverage secure enclaves namely Intel SGX [69] to build private mempools at the cost of potential latency, storage limits, and security risks due to several successful recent attacks on secure enclaves [54, 65]. 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 [27, 18, 40, 50] relies on the secure implementation of verifiable delay functions (VDF) [7] 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 [66].

Shutter Network [58] 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 [32], Sikka [59], Helix [2], and F3 [70] 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.

3 Background

3.1 Cryptographic Preliminaries

Identity-Based Encryption. An identity-based encryption (IBE) [9, 19, 57] allows to establish 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, and IP address. The receiver Bob, having obtained a private key associated with his identity information from Trusted Third Party (TTP), can decrypt the ciphertext. An IBE scheme consists of a tuple of algorithms: *Setup*, *Extract*, *Encrypt*, and *Decrypt* satisfying the following semantics:

- *Setup*(1^λ)**:** 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.

We build FairBlock using an IBE that is semantically secure under the BDH assumption in a random-oracle model [9]. In particular, we use the Boneh-Franklin IBE [9]. Our construction will use an IBE in a non-black box way and exploit two common properties. Other IBE schemes [41, 8] 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 [9], Shamir's secret sharing (SSS) [56] technique can replace the TTP by distributing the shares of msk among a group of keepers with an honest majority. In this work, we show how to employ a distributed key generation [49] to eliminate the trusted dealer in SSS to achieve complete decentralization.

Cryptographic Commitment. In order to ensure that transactions cannot be modified, censored, or added after decryption by relayers, our protocol should verify that decrypted transactions are in fact the ones that have been encrypted. To realize this, we have leveraged a basic non-interactive hash-based commitment [11, 44, 37] with computational binding and hiding properties in the random oracle model based on a collision-resistant hash function H_c. 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 in this work can be replaced with more advanced commitment schemes [48, 37, 36] with stronger security guarantees.

3.2 Blockchain Front-running

In this paper, we define blockchain front-running as follows:

Definition 1. *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 [47, 29, 24] 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. We refer the reader to Appendix A for a summary on the nature of front-running attacks.

4 FairBlock

In this section, we formalize the security model in Section 4.1, present FairBlock's architecture in 4.2, and finally prove the correctness and security of the protocol in 4.3 and 4.4.

4.1 Model

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) [17], or decentralized autonomous organizations (DAO) [25].
- **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) [17], or decentralized autonomous organizations (DAO) [25]. 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.

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 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.

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 [49] 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 [13] 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.

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.

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. We follow the formal notion of fairness in recent works [39, 17, 71] and aim to provide fairness by satisfying both order-fairness and secure causality preservation [28, 15]. 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 [39, 17, 71] for further formal discussion and technical detail of order-fairness. Also, the security conditions of secure causality-preservation [28, 15] 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 4.4, we show how FairBlock satisfies both secure causality-preservation and order-fairness.

4.2 Protocol

In this section, we show how to apply FairBlock to any dApp by adding special-purpose smart contracts to the system. However, one can similarly apply FairBlock at the consensus level, where keepers and relayers are replaced by the validators that maintain the shared master key and contribute to the decryption on chain [32, 59]. We will further discuss consensus-level Implementation.

Smart Contracts. To implement FairBlock using smart contracts as the communication layer, we introduce five smart contracts:

1. $\mathsf{Participate}(\mathsf{dep}, \mathsf{val})$: 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. $\mathsf{DKG}(\mathsf{m}_{\mathsf{DKG}})$: During 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. $\mathsf{IBE}(\mathsf{b^k_h})$: This contract receives block key shares b^k_h from each keeper k, so the relayer can aggregate them to construct the block key.

What follows is a brief description of FairBlock's architecture in six phases:

Phase 0: Enrollment. Keepers and relayers enroll in participating in the protocol by calling a function in Participate and sending an amount of deposit as an entry fee. Clients may also lock the value of their transaction in Participate, 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, and an associated shared master key split across all of them using a DKG protocol [49]. DKG protocols are generally slow as they typically require time quadratic in n [38, 49, 63]; however, it only runs once in the setup phase and afterward very infrequently anytime the keepers set changes. Keepers set is expected to be stable as they are collecting rewards for their honest co-operation and being penalized for malicious behavior. The following is a brief description of Pedersen's DKG protocol [49]:

1. *Sharing:* 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:
$$f_i(z) = a_{i0} + a_{i1}z + \cdots + a_{it}z^t. \tag{1}$$
P_i broadcasts Feldman [31] commitments $A_{ik} = g^{a_{ik}}$ for $k \in [0, t]$ using DKG. 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 [31] validity condition: $g^{s_{ij}} \overset{?}{=} \prod_{k=0}^{t} A_{ik}^{j^k}$.
3. *Dispute:* If t or more keepers complain against a keeper P_f by broadcasting the complaint on DKG, 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:
$$mpk = \prod_{i \in T} A_{i0} = \prod_{i \in T} g^{s_i}. \tag{2}$$

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 sum of qualified keepers' secrets: $msk = \sum_{i \in T} s_i$. Note that secret s_f for a disqualified keeper P_f is set to zero.

Phase 2: Encryption and Commitment. Clients encrypt their message m using public key mpk for block identifier h. To encrypt transaction information $t_x \in \{0,1\}^{l_1}$, a client 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 $R = g^r$ [9]. Having them, U is calculated as follows:

$$U = m \oplus H_2(\hat{e}(Q_h, mpk)^r). \tag{3}$$

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

Phase 3: Broadcasting Block Key Shares. At least $t + 1$ out of n keepers compute their block key shares b_h^k [9] and propagate it using IBE. Keeper k computes its share for block h as $b_h^k = H_1(h)^{w_k}$.

Phase 4: Decryption. Relayers compute the block key b_h after receiving 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 as fast as possible by rewards for each correct decryption. Each Relayer can extract block key b_h after the following steps:

1. *Share Verification:* Relayer verifies received shares b_h^k from each keeper k by checking the following condition [9]:

$$\hat{e}(\prod_{i=0}^{t} V_i^{k^i}, H_1(h)) \stackrel{?}{=} \hat{e}(g, b_h^k), \tag{4}$$

   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:

$$b_h = \prod_{k=1}^{t+1} (b_h^k)^{L_k}, \tag{5}$$

   where L_ks are proper Lagrange coefficients for point 0 defined as $L_k = \prod_{\substack{r=1 \\ r \neq k}}^{t+1} \frac{r}{r-k}$. The derived key b_h is indeed the IBE key for identity h that would have been extracted by the TTP. See Sect. 4.3 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:

$$m = U \oplus H_2(\hat{e}(b_h, R)). \tag{6}$$

In the event that relayers are unable to include the decrypted transactions in the blockchain before or at the block specified in the commitments, an application-specific policy determines if the submitted encrypted transactions should fail or be decrypted in a later block. However, we expect that this case is very rare with a proper incentivization mechanism.

Phase 5: Execution. Given a list of decrypted messages $m_1, ..., m_n$, $\mathsf{Process}$ 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 $\mathsf{Participate}$ 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 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 $H_c(m_1, \cdot\cdot\cdot, m_n)$ and comparing them. Finally, it executes the batch by calling the target contract for each of the decrypted transactions.

Consensus-level Implementation. An alternative to using smart contracts as the communication layer is implementing FairBlock in the consensus layer [59, 2]. In this case, there will be no need to maintain sets of keepers and relayers, as the normal validators who are responsible for mining the blocks will also perform these tasks. To be more specific, validators receive private key shares in proportion to their stake and submit their block key shares as an extension to messages in the voting round in a BFT-style consensus algorithm [13]. Next, the next block proposer computes the block key by aggregating submitted shares, decrypting, and including the plaintext transaction in the next block.

Alternatively, validators can submit their block key shares as a message in the blockchain, and any other user can compute the block key and submit decrypted transactions as a message to collect rewards. Hash of submitted decryptions should be compared to the commitments which are previously sent alongside the encrypted transactions to verify the correctness of the block key extraction and decryption process. The original sender of the encrypted message is able to submit its plaintext transaction immediately without block key extraction and decryption in order to avoid system fees. In the event that the original sender is no longer online or refuses to reveal in time, other parties will compete to decrypt the transactions as soon as possible to collect rewards.

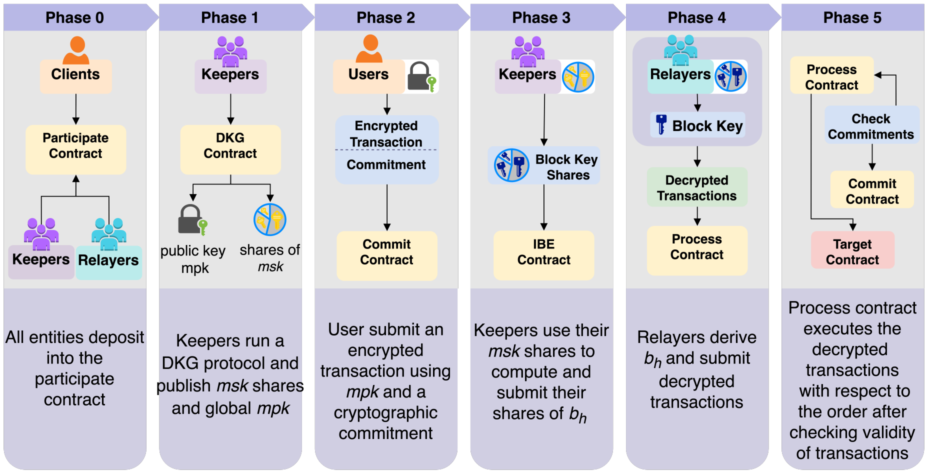
Phase 0
Clients
Participate Contract
Keepers
Relayers
All entities deposit into the participate contract
Phase 1
Keepers
DKG Contract
public key mpk
shares of msk
Keepers run a DKG protocol and publish msk shares and global mpk
Phase 2
Users
Encrypted Transaction
Commitment
Commit Contract
User submit an encrypted transaction using mpk and a cryptographic commitment
Phase 3
Keepers
Block Key Shares
IBE Contract
Keepers use their msk shares to compute and submit their shares of b_h
Phase 4
Relayers
Block Key
Decrypted Transactions
Process Contract
Relayers derive b_h and submit decrypted transactions
Phase 5
Process Contract
Check Commitments
Commit Contract
Target Contract
Process contract executes the decrypted transactions with respect to the order after checking validity of transactions

Fig. 1. Architecture of FairBlock

4.3 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. We refer the reader to Appendix B for correctness proof.

4.4 Security

Secure Causality Preservation. Assuming the honest majority of keepers described in Section 4.1, 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 client, 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:

$$\mathsf{Adv}_G(\mathcal{A}) = \mid \Pr[W] - 0.5 \mid \leq \varepsilon \tag{7}$$

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 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 [33, 49]. 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 miners in the decryption and execution phase can influence the fixed order of executed transactions. Moreover, no party including miners can insert transactions before the decrypted batch or directly submit transactions to the target contract to frontrun 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” [39, 17, 71] which favors the transactions that are received earlier. This property has been a subject of debate in the blockchain community lately [24, 17, 59]. 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. 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. [39] propose executing all the received transactions in parallel which is implemented in Chainlink fair sequencing service (FSS) [17] 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 6.

5 Implementation

5.1 Implementation Details

We have built prototype implementations of FairBlock for both consensus-level and smart contracts approaches. Smart contracts are implemented in Solidity and consensus-level blockchain is built based on Cosmos SDK [22] 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, a more efficient implementation would be submitting block key shares as an extension to messages in the voting round in a BFT-style consensus algorithm [13]. This implementation can be realized in FairBlock blockchain after release of ABCI++ [21] which allows validators in a Cosmos-based blockchain to extend their votes in the consensus voting phase with their shares of block key [32, 59].

Our implementation of the distributed IBE is built on top of Vuvuzela cryptography library [67] 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 [9] can also be described with type 3 pairings [10]. For the DKG part, we have used Pedersen’s scheme [49], as it is efficient, fast, and can be explained simply in this paper. However, this DKG scheme can be replaced with implementations and schemes such as [38, 35, 55, 34, 33, 63] to achieve better properties. Both implementations can be readily employed for auctions, gaming, and various other DeFi use cases. Moreover, other PoS blockchain networks including [30, 20, 4, 60] can also prevent front-running in their network by including FairBlock in their consensus mechanism. Source code of FairBlock including distributed IBE and smart contracts is available on GitHub[4]. Source code of FairBlock implementation in the consensus layer is also available on GitHub[5].

5.2 Performance Evaluation

To measure 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 test the implementation for systems of up to 500 keepers and present average execution times in Table 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 (PoS) and proof-of-work (PoW) public blockchains. For instance, 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 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 neglectable compared to block key extraction time and can be easily parallelized with the same execution time. For larger message sizes, our work employs hybrid encryption [26]. Using hybrid encryption, identity-based encryption is used to encrypt a key and an efficient symmetric encryption scheme such as AES-GCM or ChaCha20 [16] is used to encrypt the actual transaction with the key.

Table 1. Mean values of encryption, block key extraction, and decryption execution time for various keepers set sizes.

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

We have compared the bandwidth overhead of FairBlock and the threshold decryption approach in two realistic scenarios. In scenario I, there are 1000

[4] https://github.com/pememoni/FairBlock-SC
[5] https://github.com/pememoni/FairBlock

keepers and 1000 encrypted transactions that should be decrypted every 24 hours. In scenario II, there are 100 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 2 shows the result of our experiment. In scenario I, the total message size of IBE approach is only 0.4% of the threshold decryption approach. Similarly, the total message size of IBE approach is approximately 25 times less than the other approach in scenario II.

Table 2. Comparison of bandwidth overhead in identity-based encryption and threshold decryption (assuming two-thirds honest majority)

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

6 Challenges and Future Work

One of the main challenges that arises in all privacy-preserving implementations is to protect leakage of information through transaction metadata. In particular, although the data field will not leak any information about the encrypted transaction itself, signature of the transaction can leak the sender's identity. In theory, an adversary with just the knowledge of the transaction's sender can perform front-running. For example, traders can be front-runned just based on their regular trading times of specific assets. Preventing such attacks requires mitigations that avoid leakage of metadata as well. As an alternative to using complex ring signatures [46], a client can avoid this risk and hide the real sender of the transaction by asking another party to send its transaction; or alternatively, replace the sender's signature with a PoW puzzle. Other privacy-enhancing technologies such as [6, 64, 14] can also be applied to prevent front-running based on the sender's public key or other forms of metadata namely IP addresses.

7 Conclusions

This paper designs and implements FairBlock, the first front-running prevention mechanism based on distributed IBE. Our work does not have 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 also open-sourced.

References

1. Adams, H., Zinsmeister, N., Salem, M., Keefer, R., Robinson, D.: Uniswap v3 core. Tech. rep., Tech. rep., Uniswap (2021)
2. Asayag, A., Cohen, G., Grayevsky, I., Leshkowitz, M., Rottenstreich, O., Tamari, R., Yakira, D.: A fair consensus protocol for transaction ordering. In: 2018 IEEE 26th International Conference on Network Protocols (ICNP). pp. 55–65 (2018). https://doi.org/10.1109/ICNP.2018.00016
3. Auer, R., Frost, J., Vidal Pastor, J.M.: 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. Bartoletti, M., Chiang, J.H.y., Lluch-Lafuente, A.: Maximizing extractable value from automated market makers. arXiv preprint arXiv:2106.01870 (2021)
6. Bojja Venkatakrishnan, S., Fanti, G., Viswanath, P.: Dandelion: Redesigning the bitcoin network for anonymity. Proc. ACM Meas. Anal. Comput. Syst. **1**(1) (jun 2017). https://doi.org/10.1145/3084459, https://doi.org/10.1145/3084459
7. Boneh, D., Bonneau, J., Bünz, B., Fisch, B.: Verifiable delay functions. In: Annual international cryptology conference. pp. 757–788. Springer (2018)
8. Boneh, D., Boyen, X.: Efficient selective-id secure identity-based encryption without random oracles. In: International conference on the theory and applications of cryptographic techniques. pp. 223–238. Springer (2004)
9. Boneh, D., Franklin, M.: Identity-based encryption from the weil pairing. In: Annual international cryptology conference. pp. 213–229. Springer (2001)
10. Boyen, X.: A tapestry of identity-based encryption: practical frameworks compared. International Journal of Applied Cryptography **1**(1), 3–21 (2008)
11. Brassard, G., Chaum, D., Crépeau, C.: Minimum disclosure proofs of knowledge. Journal of computer and system sciences **37**(2), 156–189 (1988)
12. Breidenbach, L., Daian, P., Tramèr, F., Juels, A.: Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. Cryptology ePrint Archive, Report 2017/1090 (2017)
13. Buchman, E., Kwon, J., Milosevic, Z.: The latest gossip on bft consensus. arXiv preprint arXiv:1807.04938 (2018)
14. Bünz, B., Agrawal, S., Zamani, M., Boneh, D.: Zether: Towards privacy in a smart contract world. In: International Conference on Financial Cryptography and Data Security. pp. 423–443. Springer (07 2020). https://doi.org/10.1007/978-3-030-51280-4_23
15. Cachin, C., Kursawe, K., Petzold, F., Shoup, V.: Secure and efficient asynchronous broadcast protocols. In: Proceedings of the 21st Annual International Cryptology Conference on Advances in Cryptology. p. 524–541. CRYPTO '01, Springer-Verlag, Berlin, Heidelberg (2001)
16. Chacha20 and poly1305 for ietf protocols. https://www.rfc-editor.org/rfc/rfc7539.txt, (Accessed on 04/03/2022)

17. Chainlink 2.0 and the future of decentralized oracle networks — chainlink. https://chain.link/whitepaper (2021)
18. Cline, D., Dryja, T., Narula, N.: Clockwork: An exchange protocol for proofs of non front-running (2020)
19. Cocks, C.: An identity based encryption scheme based on quadratic residues. In: IMA international conference on cryptography and coding. pp. 360–363. Springer (2001)
20. Cosmos: The internet of blockchains. https://cosmos.network/, (Accessed on 03/18/2022)
21. Abci++. https://github.com/tendermint/spec/blob/master/rfc/004-abci++.md, (Accessed on 04/03/2022)
22. Cosmos sdk - cosmos network. https://v1.cosmos.network/sdk, (Accessed on 03/26/2022)
23. Cowswap - meta dex aggregator. https://cowswap.exchange/, (Accessed on 12/03/2021)
24. Daian, P., Goldfeder, S., Kell, T., Li, Y., Zhao, X., Bentov, I., Breidenbach, L., Juels, A.: Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. arXiv preprint arXiv:1904.05234 (2019)
25. Dao — aragon. https://aragon.org/dao, (Accessed on 04/01/2022)
26. Dixit, P., Gupta, A.K., Trivedi, M.C., Yadav, V.K.: Traditional and hybrid encryption techniques: a survey. In: Networking communication and data knowledge engineering, pp. 239–248. Springer (2018)
27. Doweck, Y., Eyal, I.: Multi-party timed commitments (2020)
28. Duan, S., Reiter, M.K., Zhang, H.: Secure causal atomic broadcast, revisited. In: 2017 47th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN). pp. 61–72 (2017). https://doi.org/10.1109/DSN.2017.64
29. Eskandari, S., Moosavi, S., Clark, J.: Sok: Transparent dishonesty: front-running attacks on blockchain (2019)
30. Ethereum upgrades. https://ethereum.org/en/upgrades/, (Accessed on 03/18/2022)
31. Feldman, P.: A practical scheme for non-interactive verifiable secret sharing. In: 28th Annual Symposium on Foundations of Computer Science (sfcs 1987). pp. 427–438 (1987). https://doi.org/10.1109/SFCS.1987.4
32. Ferveo. https://anoma.network/blog/ferveo-a-distributed-key-generation-scheme-for-front-running-protection/, (Accessed on 12/03/2021)
33. Gennaro, R., Jarecki, S., Krawczyk, H., Rabin, T.: Secure distributed key generation for discrete-log based cryptosystems. Journal of Cryptology **20**(1), 51–83 (2007)
34. Groth, J.: Non-interactive distributed key generation and key resharing. Cryptology ePrint Archive, Report 2021/339 (2021)
35. Gurkan, K., Jovanovic, P., Maller, M., Meiklejohn, S., Stern, G., Tomescu, A.: Aggregatable distributed key generation. In: Annual International Conference on the Theory and Applications of Cryptographic Techniques. pp. 147–176. Springer (2021). https://doi.org/10.1007/978-3-030-77870-5_6
36. Halevi, S.: Efficient commitment schemes with bounded sender and unbounded receiver. In: Proceedings of the 15th Annual International Cryptology Conference on Advances in Cryptology. p. 84–96. CRYPTO '95, Springer-Verlag, Berlin, Heidelberg (1995)
37. Halevi, S., Micali, S.: Practical and provably-secure commitment schemes from collision-free hashing. In: Koblitz, N. (ed.) Advances in Cryptology — CRYPTO '96. pp. 201–215 (1996)

38. Kate, A., Goldberg, I.: Distributed private-key generators for identity-based cryptography. In: Security and Cryptography for Networks. pp. 436–453. Springer Berlin Heidelberg (2010)
39. Kelkar, M., Zhang, F., Goldfeder, S., Juels, A.: Order-fairness for byzantine consensus. In: Annual International Cryptology Conference. pp. 451–480. Springer (2020)
40. Khalil, R., Gervais, A., Felley, G.: TEX - a securely scalable trustless exchange. IACR Cryptol. ePrint Arch. p. 265 (2019)
41. Libert, B., Quisquater, J.J.: Identity based encryption without redundancy. In: International Conference on Applied Cryptography and Network Security. pp. 285–300. Springer (2005)
42. Mev-Explore. https://explore.flashbots.net/, (Accessed on 12/03/2021)
43. Momeni, P., Gorbunov, S., Zhang, B.: Fairblock: Preventing blockchain front-running with minimal overheads. In: International Conference on Security and Privacy in Communication Systems. p. in press. Springer (2022)
44. Naor, M.: Bit commitment using pseudorandomness. J. Cryptol. **4**(2), 151–158 (jan 1991). https://doi.org/10.1007/BF00196774
45. Network, S.: Secret markets: Front running prevention for automated market makers. https://scrt.network/blog/secret-markets-front-running-prevention, (Accessed on 06/22/2022)
46. Noether, S.: Ring signature confidential transactions for monero. IACR Cryptol. ePrint Arch p. 1098 (2015)
47. Obadia, A., Salles, A., Sankar, L., Chitra, T., Chellani, V., Daian, P.: Unity is strength: A formalization of cross-domain maximal extractable value (2021)
48. Pedersen, T.P.: Non-interactive and information-theoretic secure verifiable secret sharing. In: Annual international cryptology conference. pp. 129–140. Springer (1991)
49. Pedersen, T.P.: A threshold cryptosystem without a trusted party. In: Davies, D.W. (ed.) Advances in Cryptology — EUROCRYPT '91. pp. 522–526. Springer Berlin Heidelberg, Berlin, Heidelberg (1991)
50. Protocol, V.: Blockchain derivatives. https://vega.xyz/, (Accessed on 06/22/2022)
51. Qin, K., Zhou, L., Gervais, A.: Quantifying blockchain extractable value: How dark is the forest? (2021)
52. Reiter, M.K., Birman, K.P.: How to securely replicate services. ACM Transactions on Programming Languages and Systems (TOPLAS) **16**(3), 986–1009 (1994)
53. Robinson, D., Konstantopoulos, G.: Ethereum is a dark forest - paradigm. https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest/, (Accessed on 12/03/2021)
54. van Schaik, S., Kwong, A., Genkin, D., Yarom, Y.: SGAxe: How SGX fails in practice (2020)
55. Schindler, P., Judmayer, A., Stifter, N., Weippl, E.: Ethdkg: Distributed key generation with ethereum smart contracts. Cryptology ePrint Archive, Report 2019/985 (2019)
56. Shamir, A.: How to share a secret. Commun. ACM **22**(11), 612–613 (1979). https://doi.org/10.1145/359168.359176, https://doi.org/10.1145/359168.359176
57. Shamir, A.: Identity-based cryptosystems and signature schemes. In: Workshop on the theory and application of cryptographic techniques. pp. 47–53. Springer (1984)
58. Shutter Network. https://shutter.ghost.io/, (Accessed on 12/03/2021)
59. Sikka. https://sikka.tech/projects/, (Accessed on 12/03/2021)
60. Solana. https://solana.com/, (Accessed on 03/18/2022)

61. Stathakopoulou, C., Rüsch, S., Brandenburger, M., Vukolic, M.: Adding fairness to order: Preventing front-running attacks in bft protocols using tees
62. Sushiswap. https://sushi.com/, (Accessed on 12/03/2021)
63. Tomescu, A., Chen, R., Zheng, Y., Abraham, I., Pinkas, B., Gueta, G.G., Devadas, S.: Towards scalable threshold cryptosystems. In: 2020 IEEE Symposium on Security and Privacy (SP). pp. 877–893. IEEE (2020)
64. Tornado Cash. https://tornado.cash/, (Accessed on 12/05/2021)
65. Van Bulck, J., Minkin, M., Weisse, O., Genkin, D., Kasikci, B., Piessens, F., Silberstein, M., Wenisch, T.F., Yarom, Y., Strackx, R.: Foreshadow: Extracting the keys to the Intel SGX kingdom with transient out-of-order execution. In: 27th USENIX Security Symposium (USENIX Security 18). pp. 991–1008 (2018)
66. Veedo. https://github.com/starkware-libs/veedo, (Accessed on 03/30/2022)
67. vuvuzela cryptography libraries. https://github.com/vuvuzela/crypto, (Accessed on 04/03/2022)
68. Wood, G.: Ethereum: A secure decentralized generalized transaction ledger (2014)
69. Xing, B.C., Shanahan, M., Leslie-Hurd, R.: 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. Association for Computing Machinery, New York, NY, USA (2016). https://doi.org/10.1145/2948618.2954330
70. Zhang, H., Merino, L.H., Estrada-Galinanes, V., Ford, B.: F3b: A low-latency commit-and-reveal architecture to mitigate blockchain front-running. arXiv preprint arXiv:2205.08529 (2022)
71. Zhang, Y., Setty, S., Chen, Q., Zhou, L., Alvisi, L.: Byzantine ordered consensus without byzantine oligarchy. In: 14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20). pp. 633–649. USENIX Association (Nov 2020)
72. Zhou, L., Qin, K., Torres, C.F., Le, D.V., Gervais, A.: High-frequency trading on decentralized on-chain exchanges. 2021 IEEE Symposium on Security and Privacy (SP) pp. 428–445 (2021)

A Front-running Strategies

In this appendix, 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 [24] 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 [68] 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.

B Correctness and Consistency

B.1 Consistency of IBE encryption and decryption

Let $C = (R, U)$ be 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:

$$\hat{e}(Q_h, mpk)^r = \hat{e}(Q_h, g)^{r.msk} = \hat{e}((Q_h)^{msk}, g^r) = \hat{e}(b_h, R) \tag{8}$$

B.2 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:

$$b_h = \prod_{k=1}^{t} (b_h^k)^{L_k} = \prod_{k=1}^{t} (H_1(h)^{w_k})^{L_k} = \prod_{k=1}^{t} H_1(h)^{w_k L_k} = H_1(h)^{\sum_{k=1}^{t} w_k L_k} \tag{9}$$

And by Lagrange interpolation formula we have:

$$H_1(h)^{\sum_{k=1}^{t} w_k L_k} = H_1(h)^{msk} \tag{10}$$