Flash Freezing Flash Boys: Countering Blockchain Front-Running

Haoqian Zhang, Louis-Henri Merino, Vero Estrada-Galiñanes, Bryan Ford
École Polytechnique Fédérale de Lausanne
Lausanne, Switzerland
{haoqian.zhang, louis-henri.merino, vero.estrada, bryan.ford}@epfl.ch

***Abstract*—Front-running, the practice of benefiting from advanced knowledge of pending transactions, has proliferated in the cryptocurrency space with the emergence of decentralized finance. Front-running causes devastating losses to honest participants—estimated at $280M each month—and endangers the fairness of the ecosystem. We present Flash Freezing Flash Boys (F3B), an architecture to address front-running attacks by relying on a commit-and-reveal scheme where the contents of a transaction are encrypted and later revealed by a decentralized secret-management committee (SMC) when the transaction has been committed by the underlying consensus layer. To maintain legacy compatibility, we design F3B to be agnostic to the underlying consensus algorithm and compatible with existing smart contracts. A preliminary exploration of F3B shows that with a secret-management committee consisting of 8 and 128 members, F3B presents between 0.1 and 2.2 seconds of transaction-processing latency, respectively.**

***Index Terms*—security & privacy, decentralized finance, front-running**

I. Introduction

Front-running is the practice of benefiting from advanced knowledge of pending transactions by entering into a financial position [1]–[3]. While benefiting the entities involved, this practice puts others at a significant financial disadvantage, leading regulators to judge this behavior as illegal (and unethical) in traditional markets [1].

Once a significant problem for traditional (centralized) markets resolved essentially via regulations, it has now become a significant problem for decentralized finance, given their pseudonymous nature and the difficulties involved with pursuing entities across numerous jurisdictions [1], [4]. While cryptocurrencies, such as Ethereum [5], support complex smart contracts that enable decentralized finance, many of them do not inherently provide protections for transactions given their visibility in mempool—before they are committed, allowing adversaries (*e.g.*, miners) to practice front-running at will. For example, adversaries can financially benefit from these transactions by creating their own transactions and positioning them in a specific sequence with respect to the targeted transaction.

Front-running negatively impacts all honest DeFi actors, but the automated market maker (AMM) is particularly vulnerable because of price slippage [6]. An estimate shows that front-running attacks amount to $280 million in losses for DeFi actors each month [7]. In addition, front-running attacks threaten the underlying consensus layer's security by incentivizing unnecessary forks [8], [9].

While front-running is clearly and openly being exploited in the decentralized finance space, recall that regulations are difficult to achieve in a decentralized system. Thereby, the solution must be technological in nature and achieve widespread support among the cryptocurrency community. While achieving the latter is a challenge on its own, existing technological solutions present various limitations as they focus on specific applications [10], [11], rely on multiple rounds with high latency overhead [11], [12], or are consensus-specific [13], [14].

We present Flash Freezing Flash Boys[1] (F3B), a front-running protection architecture that exhibits low transaction-processing latency overhead while remaining legacy compatible (agnostic to the underlying consensus algorithm and to the smart contract implementation). F3B addresses front-running by adopting a commit-and-reveal architecture that encrypts transactions that are revealed after being committed by the consensus layer. By encrypting transactions, malicious parties are now at a serious disadvantage since they no longer have visibility of transactions while they rest in mempool.

To overcome a notable problem where the sender needs to be online during the reveal phase, F3B builds on Calypso [15], an architecture that enables on-chain secrets by introducing a secret-management committee (SMC) that reveals those on-chain secrets when designated. In our setup, the on-chain secrets are the encrypted transactions and the designed revelation period occurs when the transaction is committed by the underlying consensus layer (e.g., Bitcoin needs 6 block confirmations while a PBFT-style consensus only needs 1 block confirmation). Notably, consensus nodes now finalize—verify and execute—a transaction after the secret-management committee reveals the contents of a transaction. At this point, it is too late for malicious actors to launch a front-running attack. F3B is independent of the consensus algorithm or smart contract implementation, but its deployment requires a fork of the underlying blockchain.

Our proposed solution notably brings about a challenge regarding the possible introduction of unsolicited transactions. F3B thus mandates, for each transaction, the sender to pay a

[1]The name *Flash Boys* comes from a popular book that reveals this aggressive market-exploiting strategy on Wall Street in 2014 [4].

storage fee to have their transaction committed, even though the consensus nodes are unable to verify and execute the transaction just yet.

We implemented a prototype of F3B in Go [16]. Our preliminary findings show that the transaction-processing latency overhead introduced is between 0.1 to 2.2 seconds for 8 to 128 trustees. To our knowledge, this is the first work to systematically evaluate threshold encryption at the transaction level and be capable of mitigating front-running attacks at this level of scalability. As part of our evaluation, we model the Ethereum blockchain with 13 seconds block time [17] and vary the block confirmations m before a transaction is considered committed or "irreversible": the total time for committing a transaction is then $13m$ seconds. Our analysis shows that for a committee size of 128, our latency overhead is 0.78% with 20 block confirmations.

Our key contributions are:

1) F3B is the first systematic exploration of threshold encryption at the transaction level to mitigate front-running attacks.
2) An preliminary experimental analysis shows that F3B has a small latency overhead with respect to Ethereum.

II. Background

We now present the background required to introduce F3B.

A. Front-running Attacks and Transaction Commitment

Recall that front-running is an attack where an entity benefits from advanced knowledge of pending transactions [1]–[3]. A front-running attack at its core is an adversary who has some advantage (e.g., network connectivity, miner) in the ordering of the transactions before they are executed and thus can place their transaction in a certain order that benefits them financially with respect to other transactions. With the advent of blockchain and expressive smart contracts, the attacks have become more complex and automated [9], [18].

While blockchain pending transactions are public information, and thus utilizing public information may not be considered illegal, front-running can still endanger finality by incentivizing forks and causing financial loss to its users [8], [9]. Cryptocurrencies suffer three main types of front-running attacks [1]: *displacement*: displacing a targeted transaction with a new transaction, *insertion*: inserting a transaction before the targeted transaction, and *suppression*: delaying a targeted transaction indefinitely.

In numerous consensus algorithms such as proof-of-work, transaction-commitment is probabilistic since miners can diverge off the main chain and create a longer chain that would overwrite any blocks since the divergence. Therefore, even if a transaction is inserted on the blockchain, an adversary can still launch a front-running attack. Given its probabilistic behavior, parties (e.g., exchanges) generally accept a transaction to be committed—no longer pending—when the transaction has been mined at a depth of m blocks, *i.e.*, the transaction has m block confirmations. While there is no universally set number for m, parties set this value based on the amount of risk they are willing to tolerate [19], [20].

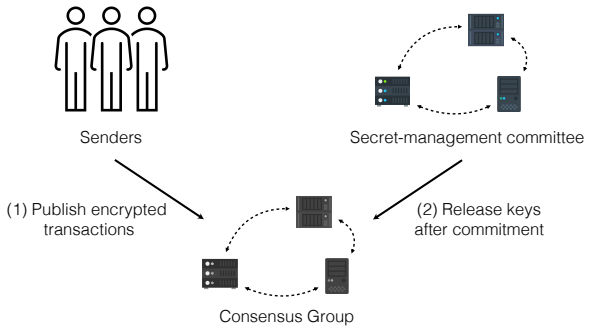

Figure 1. Senders publish encrypted transactions to the consensus group. Once the transactions are no longer pending, the secret-management committee releases the keys.

B. Threshold Encryption and Calypso

A (t, n)-threshold secret-sharing scheme enables a secret to be shared among n parties (trustees) and requires at least t parties to recover the secret; $t-1$ parties reveal *nothing* about the secret [21]. Distributed key generation (DKG) [22] is a multi-party (t, n) key-generation process to collectively generate a public-private key pair (sk, pk) without relying on a single trusted dealer; each trustee i obtains a share sk_i of the secret key sk, and collectively obtains a public key pk. Any client can now use pk to encrypt a secret, and at least t trustees must cooperate to retrieve this secret [23].

Calypso is a framework that enables on-chain secrets by adopting threshold encryption governed by a secret-management committee [15]. Calypso allows ciphertexts to be stored on the blockchain and collectively decrypted by trustees according to a predefined policy. F3B leverages Calypso to mitigate front-running attacks and extends its functionality to automatically release the transaction contents once committed.

III. System Overview

In this section, we present F3B with two strawman protocols to motivate F3B's system goals and then illustrate our proposed protocol and how it achieves those system goals.

A. Architecture Overview

F3B, shown in figure 1, mitigates front-running attacks by working with a secret-management committee to manage the storage and release of on-chain secrets. Instead of propagating their transactions in cleartext, the sender now encrypts their transactions and stores the associated secret key with the secret-management committee before propagating it— no one can now read or understand the content of pending transactions. Once the transaction is committed, the secret-management committee releases the secret keys so that consensus nodes of the underlying blockchain can verify and execute transactions.

Notably, F3B encrypts the entire transaction, not just inputs, as other information such as the smart contract address may

provide enough information to launch a front-running attack, such as the Fomo3D attack [1] or a speculative attack [6].

B. Modeling the underlying blockchain as the baseline

We assume that the underlying blockchain has a consensus protocol that commits transactions as part of blocks. We model the underlying blockchain as our baseline protocol for comparison. W.l.o.g., we model the underlying blockchain's block time as L_b seconds, *e.g.*, on average, Ethereum has a block time of 13 seconds [17]. We consider a transaction to be committed once its block has a sufficient number of confirmations for proof-of-work and proof-of-stake consensus algorithms. In reality, exchanges require multiple block confirmations before the funds are credited to users' accounts, *e.g.*, Kraken and Coinbase require 20 and 35 block confirmations, respectively, for Ethereum transactions [19], [20]; we define a transaction as committed after m block confirmations. For PBFT-style consensus, the required m is 1. Thereby, in our baseline, the transaction latency[2] is mL_b. Further, we assume that the underlying blockchain has a throughput of T_b tps.

C. Strawman Protocols

In order to explore the challenges inherent in building a framework like F3B, we first examine a couple of promising but inadequate strawman approaches.

1) (Strawman I) Commit-and-Reveal by User: The first strawman divides a transaction into two steps: commit and reveal. First, Alice creates her transaction, and instead of propagating the transaction, she commits to her transaction by propagating her transaction hash to the consensus group. Once the consensus group commits the hash into the underlying blockchain, Alice propagates her transaction, revealing its contents (execution parameters such as inputs and any smart contract interactions). Consensus nodes finally execute the transaction with respect to its hash order committed on the blockchain, assuming the transaction contents matches the transaction hash provided earlier by Alice.

This simple strawman mitigates front-running attacks since the consensus group committed the execution order before Alice revealed her transaction, but it comes with some significant disadvantages: a) Alice must continuously monitor the blockchain to find out when to reveal her transaction, and b) Alice may not be able to reveal her transaction due to a cryptokitties storm or blockchain DoS attacks, and c) this approach is subject to output bias as consensus nodes may deliberately choose not to include certain transactions into the blockchain [6], and d) this approach brings high latency overhead than the baseline protocol, as Alice needs to propagate two times.

2) (Strawman II) The Trusted Custodian: A straightforward method to remove Alice from the equation after committing to a transaction is by employing a trusted custodian. Instead of releasing a transaction hash, Alice encrypts her entire transaction with a symmetric key and shares this symmetric key with a trusted custodian. After the encrypted transaction is committed, the trusted custodian releases the key to the consensus group, which decrypts and executes the transaction.

This strawman also mitigates front-running attacks, as the nodes cannot read the content of transactions before ordering, with similar latency and throughput to the baseline protocol. However, it introduces a centralized and opaque component: the trusted custodian who may secretly act maliciously, such as colluding with front-running actors for their own profit. In addition, the trusted custodian represents a single point of failure where consensus nodes cannot decrypt and execute a transaction if the custodian crashes. To mitigate those two risks, we need decentralization to mitigate the single point of failure and make collusion significantly more difficult.

D. System Goals

1) **Front-running protection:** Preventing entities from launching front-running attacks.
2) **Decentralization:** No single point of failure or compromise.
3) **Confidentiality:** Transactions are only revealed after it is committed by the underlying consensus layer.
4) **Compatibility:** The system is agnostic to the underlying consensus algorithm and smart contract implementation.
5) **Low latency overhead**: The system presents a low latency transaction-processing overhead.

E. System and Network Model

F3B's architecture consists of three components: *senders* that publish (encrypted) transactions, the *secret-management committee* that manages and releases secrets, and the *consensus group* that maintains the *underlying blockchain*. While F3B supports various consensus algorithms, such as proof-of-work like Ethereum and PBFT-style consensus algorithms like ByzCoin [24], F3B does require a forked instance of the underlying blockchain to allow the consensus group to commit encrypted transactions and to finalize transactions after obtaining the keys. In a permissioned blockchain, the secret-management committee and the consensus nodes can consist of the same set of servers. For clarity, however, we discuss them as separate entities. In addition, we use the name "Alice" to represent a generic sender.

For the underlying network, we assume that all honest blockchain nodes and trustees of the secret-management committee are well connected, and their communication channels are synchronous, *i.e.*, if an honest node or trustee broadcasts a message, all honest nodes and trustees receive the message within a known maximum time delay [25].

F. Threat Model

We assume that the adversary is computationally bounded, that cryptographic primitives we use are secure, and that the Diffie-Hellman problem is hard. We further assume that all messages are signed and that consensus nodes and the

[2]For simplicity, we leave out the time for blockchain nodes to verify and execute transactions, and assume that a transaction propagates the network within one block time as to not contribute to transaction-processing latency.

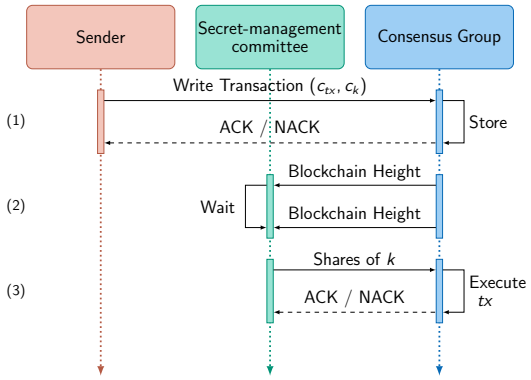

Figure 2. F3B per-transaction protocol steps: (1) Sending an encrypted transaction to the underlying blockchain, (2) Waiting for the transaction commitment, (3) Releasing the key and executing the transaction

secret-management committee only process correctly signed messages.

The secret-management committee has n trustees of which f can fail or behave maliciously. We require $n \geq 2f + 1$ and set the secret-recovery threshold to $t = f + 1$. We assume that the underlying blockchain is secure: *e.g.*, at most c of $3c + 1$ consensus nodes fail or misbehave in a PBFT-style blockchain, or the adversary controls less than 50% computational power in a proof-of-work blockchain like Bitcoin or Ethereum.

G. Protocol and Overhead Analysis

In the beginning, the secret-management committee runs a DKG protocol to generate a private key share sk^i_{smc} for each member and a public key pk_{smc}. The secret-management committee only needs to run DKG once per reconfiguration; thus, it does not contribute to the transaction latency. We use the TDH2 cryptosystem as the threshold encryption scheme that provides protection to chosen ciphertext attacks and provides NIZK proofs to validate the secret shares [23]. We unpack the per-transaction protocol (see figure 2) as follows:

1) Alice, as the sender, first chooses a symmetric key k and encrypts it with pk_{smc} obtaining the ciphertext c_k. Next, Alice encrypts transaction $c_{tx} = enc_k(tx)$, and sends (c_{tx}, c_k) to the consensus group, which writes the pair into the blockchain.
2) The secret-management committee waits for (c_{tx}, c_k) to be committed on the blockchain (after m block confirmations).
3) Each secret-management committee trustee reads c_k from the underlying blockchain and releases their decrypted share of k along with a NIZK proof of correctness for the decryption process. Consensus nodes verify the decrypted shares and use them to reconstruct k using Lagrange interpolation of shares when there are at least t valid shares. The consensus group then acquires the original $tx = dec_k(c_{tx})$ and verifies and executes tx. We denote the time for this step as L_r.

Steps 1 and 2 commit a transaction on the underlying blockchain and wait until its committed, which takes mL_b time based on the baseline model (§III-B). Compared with the baseline, step 3 is an additional step, and we denote this overhead to be L_r. As we are relying on another component, the secret-management committee, it may become a bottleneck with respect to the system throughput, and thus the throughput is $min(T_b, T_{smc})$, assuming the secret-management committee's throughput is T_{smc}.

H. Achieving System Goals

Front-running protection: *Preventing entities from launching front-running attacks.*

We reason the protection offered by F3B from the definition of front-running: if an adversary cannot benefit from pending transactions, he cannot launch front-running attacks. In F3B, no entity except the sender knows the content of pending transactions and is financially incentivized *not* to release its contents. The key is released only when a transaction is committed; thus, by definition, the attacker loses their advantage to launch a front-running attack. However, we acknowledge that attackers may use metadata of the encrypted transaction to launch *speculative* front-running attacks, as discussed in §VI.

Decentralization: *No single point of failure or compromise.*

The secret-management committee can handle up to $t - 1$ malicious trustees and up to $n - t$ offline trustees.

Confidentiality: *Transactions are only revealed after it is committed by the underlying consensus layer.*

Each transaction is encrypted with a symmetric key generated by the sender, and the symmetric key is encrypted with the secret-management committee's public key. Per our threat model, only f trustees may behave maliciously, ensuring that even with collusion, these f trustees cannot obtain the symmetric key. When $f + 1$ nodes follow the protocol by waiting for transaction commitment, the secret-management committee can reconstruct and release the key so that the consensus group can decrypt the transaction.

Compatibility: *The system is agnostic to the underlying consensus algorithm and smart contract implementation.*

Since we are encrypting the entire transaction and require a fork at the consensus layer to adapt to F3B's protocol, F3B does not require modifications to existing smart contract implementations nor to the consensus algorithm.

Low latency overhead: *The system presents a low latency transaction-processing overhead.*

We refer to section IV for a full discussion and evaluation of the latency overhead.

I. Incentives and Abuse Protection

Given that transactions are encrypted, consensus nodes cannot verify or execute transactions, opening up an availability attack that would otherwise not exist in an open system. A malicious adversary could spam the blockchain with inexecutable transactions (e.g., not enough execution costs, malformed transactions), thus significantly hindering the throughput for honest transactions. To prevent this attack, we introduce a storage fee alongside the traditional execution fee

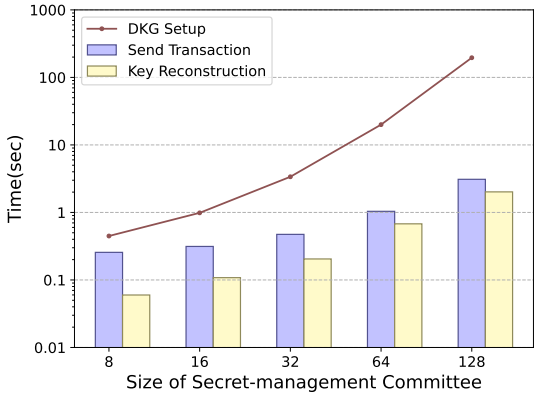

Figure 3. Overhead of F3B for varying sizes of the secret-management committee.

(e.g., Ethereum gas) that makes it costly for an attacker to operate this attack. The storage fee paid to miners covers the placement of the transaction on the blockchain and can vary based on the size of the transaction. The secret-management committee nodes will also need an incentive to operate. One option is requiring each consensus node to have the ability to run a secret-management committee node. The execution fee is not calculated until after the transaction is revealed by the secret-management committee, given the lack of knowledge of the transaction's contents.

J. Reconfiguration of secret-management committee

The membership of the secret-management committee needs to be reconfigured at some predefined interval (each epoch) to allow for new trustees or the removal of others and to prevent silent violation of our threat model over a long period of time. In a new epoch, the old secret-management committee continues to operate until all transactions encrypted with their public key are revealed while new transactions are encrypted with the new secret-management committee's public key.

IV. Evaluation

We implemented a prototype of F3B in Go [16], built on Calypso [15] and supported by Kyber [26], an advanced cryptographic library. We use ByzCoin [24], a PBFT-style consensus protocol as our underlying blockchain. We instantiate our cryptographic primitives using Edward25519 elliptic curve with 128-bit security.

Our preliminary evaluation focuses on latency. We ran our experiment on a single server with 32GB of memory and 20 CPU cores running at 2.1GHz. In figure 3, while varying the number of secret-management committee trustees, we present the total time a) for setting up a DKG, b) for sending a transaction to the Byzcoin blockchain (Step 1), and c) for reconstructing the key (Step 3). Recall that DKG setup is a one-time operation per epoch that can be bootstrapped during the previous epoch; thus, b) and c) represent the true latency of our solution.

Table I
Latency Overhead for Ethereum Blockchain

	Latency Overhead for different sizes of SMC				
Confirmations	8	16	32	64	128
5	0.09%	0.17%	0.31%	1.04%	3.10%
10	0.05%	0.08%	0.16%	0.52%	1.55%
20	0.02%	0.04%	0.08%	0.26%	0.78%
30	0.02%	0.03%	0.05%	0.17%	0.52%
50	0.01%	0.02%	0.03%	0.10%	0.31%

We consider the transaction's overall latency in F3B to be expressed as $mL_b + L_r$ (§III-G). To evaluate F3B with Ethereum's consensus model, we adopt 13 seconds as the expected block time [17]. Since there is no standard number of confirmations before a block is considered committed, we vary both the number of confirmations and the size of the secret-management committee summarized in Table I. Our result shows that for a committee size of 128 with 20 block confirmations, F3B brings a 0.78% latency overhead.

V. Related Work

Front-running attacks on the blockchain were first systematized by Eskandari et al. [1] and quantified by Daian et al. [9]. Previous works [13], [27] explore the idea of applying threshold encryption at the transaction level to mitigate front-running attacks. Schmid [13] proposed the secure causal atomic broadcast protocol with threshold encryption to prevent front-running attacks but did not provide a solution for integrating with proof-of-work blockchain. CodeChain [27] proposed to reveal transactions by their trustees using the Shamir secret sharing scheme. To our knowledge, our work is the first solution of this kind with compatibility to the consensus algorithm and smart contract implementation while achieving low latency overhead.

Other research adopts different approaches to mitigate front-running. A series of recent studies focus on fair ordering [28]–[30], but it alone can not prevent a rushing adversary [6]. Wendy explores the possibility of combining fair ordering with commit and reveal [30] but does not present quantitative overhead analysis. Some research adopts time-lock puzzles [31] to blind transactions. Injective protocol [32] deploys a verifiable delay function [33] as proof-of-elapsed-time to prevent front-running attacks. However, it is still an open challenge to link the time-lock puzzle parameters to the actual real-world delay [6].

VI. Limitations & Future Extensions

This section introduces future extensions of F3B.

a) Metadata Leakage: In our architecture, adversaries can only observe encrypted transactions until they are committed, thus preventing the revelation of transaction inputs to launch front-running attacks. Nevertheless, adversaries can still utilize side channels such as transaction metadata to launch speculative attacks. Specifically, since the sender needs to pay the storage fee (§III-I) for publishing an encrypted

transaction to the underlying blockchain, this leaks the sender's address. Knowledge of the sender's address can be helpful in launching a front-running attack since an adversary may be able to predict the sender's behavior based on historical transactions. In order to prevent this second-order front-running attack, an underlying blockchain needs to offer anonymous payment to users, such as Zerocash [34] or a mixing service [35]. Another side-channel leakage is the size of the encrypted transaction or the time the transaction is propagated; a possible remedy for mitigating metadata leakage is PURBs [36].

b) Scaling secret-management committee: The secret-management committee may become the bottleneck of the underlying consensus protocol due to its inability to achieve the same throughput. Nevertheless, a secret-management committee can operate independently from another secret-management committee. We thus suggest adopting a *sharding* strategy by running numerous secret-management committees in parallel and load balancing pending transactions between them. In addition, it may be possible to reconstruct a secret on a per-block basis rather than on a per-transaction basis using identity-based encryption [37], thus helping increase throughput even further.

VII. Conclusion

This paper introduces F3B, an architecture that addresses front-running attacks by encrypting pending transactions with threshold encryption. We performed a preliminary exploration demonstrating that F3B can protect the Ethereum blockchain with a low latency overhead.

References

[1] S. Eskandari, S. Moosavi, and J. Clark, "Sok: Transparent dishonesty: front-running attacks on blockchain," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 170–189.

[2] D. Bernhardt and B. Taub, "Front-running dynamics," *Journal of Economic Theory*, vol. 138, no. 1, pp. 288–296, 2008.

[3] "Nasdaq: Front running," https://www.nasdaq.com/glossary/f/front-running, 2018(?), accessed: 2022-04-17.

[4] M. Lewis, *Flash boys: a Wall Street revolt*. WW Norton & Company, 2014.

[5] C. Dannen, *Introducing Ethereum and solidity*. Springer, 2017, vol. 1.

[6] C. Baum, J. H.-y. Chiang, B. David, T. K. Frederiksen, and L. Gentile, "Sok: Mitigation of front-running in decentralized finance," *Cryptology ePrint Archive*, 2021.

[7] E. Mikalauskas. (2021) 280 million stolen per month from crypto transactions. Accessed: 2022-02-16. [Online]. Available: https://cybernews.com/crypto/flash-boys-2-0-front-runners-draining-280-million-per-month-from-crypto-transactions

[8] dapp.org, "Uniswap v2 audit report," 2020, accessed: 2022-01-22. [Online]. Available: https://dapp.org.uk/reports/uniswapv2.html

[9] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels, "Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges," *arXiv preprint arXiv:1904.05234*, 2019.

[10] M. Ciampi, M. Ishaq, M. Magdon-Ismail, R. Ostrovsky, and V. Zikas, "Fairmm: A fast and frontrunning-resistant crypto market-maker," *Cryptology ePrint Archive*, 2021.

[11] H. A. Kalodner, M. Carlsten, P. Ellenbogen, J. Bonneau, and A. Narayanan, "An empirical study of namecoin and lessons for decentralized namespace design." in *WEIS*. Citeseer, 2015.

[12] LibSubmarine, "Defeat front-running on ethereum," 2017(?), accessed: 2022-01-24. [Online]. Available: https://libsubmarine.org

[13] N. Schmid. (2021) Secure causal atomic broadcast. [Online]. Available: https://crypto.unibe.ch/archive/theses/2021.bsc.noah.schmid.pdf

[14] C. Stathakopoulou, S. Rüsch, M. Brandenburger, and M. Vukolić, "Adding fairness to order: Preventing front-running attacks in bft protocols using tees," in *2021 40th International Symposium on Reliable Distributed Systems (SRDS)*. IEEE, 2021, pp. 34–45.

[15] E. Kokoris-Kogias, E. C. Alp, L. Gasser, P. Jovanovic, E. Syta, and B. Ford, "Calypso: Private data management for decentralized ledgers," *Cryptology ePrint Archive*, 2018.

[16] Go, "The go programming language," 2009. [Online]. Available: https://go.dev

[17] "Blocks," 2022, accessed: 2022-02-24. [Online]. Available: https://ethereum.org/en/developers/docs/blocks

[18] G. K. Dan Robinson, "Ethereum is a dark forest," 2020. [Online]. Available: https://www.paradigm.xyz/2020/08/ethereum-is-a-dark-forest

[19] Coinbase, "Coinbase confirmations," 2022(?), accessed: 2022-03-03. [Online]. Available: https://help.coinbase.com/en/coinbase/getting-started/crypto-education/glossary/confirmations

[20] Kraken, "Cryptocurrency deposit processing times," 2022(?), accessed: 2022-03-03. [Online]. Available: https://support.kraken.com/hc/en-us/articles/203325283-

[21] A. Shamir, "How to share a secret," *Communications of the ACM*, vol. 22, no. 11, pp. 612–613, 1979.

[22] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 295–310.

[23] V. Shoup and R. Gennaro, "Securing threshold cryptosystems against chosen ciphertext attack," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1998, pp. 1–16.

[24] E. K. Kogias, P. Jovanovic, N. Gailly, I. Khoffi, L. Gasser, and B. Ford, "Enhancing bitcoin security and performance with strong consistency via collective signing," in *25th usenix security symposium (usenix security 16)*, 2016, pp. 279–296.

[25] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 643–673.

[26] "https://github.com/dedis/kyberThe Kyber Cryptography Library," 2010 – 2022.

[27] S. Najafi. (2020) Front-running attacks on blockchain. [Online]. Available: https://medium.com/codechain/front-running-attacks-on-blockchain-1f5ba28cd42b

[28] M. Kelkar, S. Deb, and S. Kannan, "Order-fair consensus in the permissionless setting," *Cryptology ePrint Archive*, 2021.

[29] K. Kursawe, "Wendy, the good little fairness widget: Achieving order fairness for blockchains," in *Proceedings of the 2nd ACM Conference on Advances in Financial Technologies*, 2020, pp. 25–36.

[30] K. Klaus, "Wendy grows up: More order fairness," in *International Conference on Financial Cryptography and Data Security*. Springer, 2021, pp. 191–196.

[31] R. L. Rivest, A. Shamir, and D. A. Wagner, "Time-lock puzzles and timed-release crypto," 1996.

[32] E. Chen and A. Chon, "Injective protocol: A collision resistant decentralized exchange protocol," 2018.

[33] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch, "Verifiable delay functions," in *Annual international cryptology conference*. Springer, 2018, pp. 757–788.

[34] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza, "Zerocash: Decentralized anonymous payments from bitcoin," in *2014 IEEE symposium on security and privacy*. IEEE, 2014, pp. 459–474.

[35] J. H. Ziegeldorf, R. Matzutt, M. Henze, F. Grossmann, and K. Wehrle, "Secure and anonymous decentralized bitcoin mixing," *Future Generation Computer Systems*, vol. 80, pp. 448–466, 2018.

[36] K. Nikitin, L. Barman, W. Lueks, M. Underwood, J.-P. Hubaux, and B. Ford, "Reducing metadata leakage from encrypted files and communication with purbs," *Proceedings on Privacy Enhancing Technologies*, vol. 2019, no. 4, pp. 6–33, 2019.

[37] D. Boneh and M. Franklin, "Identity-based encryption from the weil pairing," in *Annual international cryptology conference*. Springer, 2001, pp. 213–229.