The paper on Blockchain Frontrunning

No Author Given

Concordia University

Abstract.

1 Introduction

The concept of the *front-running* is decades old starting with early financial markets. Initially the traders required to be physically available in the exchange office to deliver the bids and offers, and some traders, in the knowledge of future market change, could run in front of the others to get their orders ahead of the change in the market to profit from the price change. Clearly in such scenarios front-running was possible yet not detectable as a damaging activity. However, with the rapid advances of the technology in the last few decades, trading is not carried out in person anymore, new form of front-running is feasible through high speed networks. Furthermore, Blockchain technology has introduced transparency and open markets and leads to a new era of front-running which inherently affects all decentral applications, whether financial or non-financial. Arbitrage and Censorship, are also issues that have been linked to front-running attacks on blockchain *(We need to talk about why these two are left out of the scope of this paper)*. Currently, there are quite a few decentral applications built on the blockchains, so in order for us to perform a thorough study on blockchain application front-running, we set a s specific criteria by using the top 25 decentral applications which have the highest recent user activities.[1] Using this criteria, we successfully categorized these decentral applications into 4 principal categories on which we conduct our analysis. Surprisingly, there was no ICO (initial coin offering) included in the top 25 Dapps list by the time we fetched the data from the website. We believe the reason is because the ICOs are inherently different from other groups of Daaps as they are merely highly active during their offering periods and their user activity drops dramatically after the offerings period are over. Therefore, we consider ICOs as another cases on which the front-running should be investigated (although they are not listed in the top 25 Dapps we use) [2]

[1] List of decentralized applications `https://dappradar.com/dapps`

[2] A note to the reviewer: this paper falls under "Case Studies (*e.g.,* of adoption, attacks, forks, scams *etc.*)" and also mentions methods for "Fraud Detection and Financial Crime Prevention".

2 Preliminaries

2.1 Traditional Front-running

Front-running is analogous to any course of action during which a person benefits from prior access to inside or confidential market information about upcoming transactions and trades. This problem can occur in both financial and non-financial systems, however, it is more noticeable within the trading and exchange systems as was originated back in the stock markets. In the old times, all the trades were executed on papers, so in case of receiving a large order from a client, the broker might say this loud and other people around the table could be informed. Therefore, a malicious trader would run in front of that order and put his own transaction in-between (before the trade was executed) and hence profit from the price increase of the stock. In fact, a group of market participants obtain non-public market information which allows them to front-run other users trades by putting their orders ahead of those trades and benefit from advance knowledge of pending client orders. The two significant factors which cause the front-running practice to happen within the financial markets are (i) imperfect competition and (ii) liquidity uncertainty [23]. Any sort of front-running activity within the financial markets is considered as an unethical and illegal practice as it is unfairly beneficial for a few market participants who have the privilege of acting on this information and taking advantages at the expense of the investors.

2.2 Literature on Traditional Front-running

As traditional front-running has been the subject of discussion for decades, there are many literatures on this subject. Here we focus on the historical context and how the regulations evolved to be what they are today.
Front-running was first appeared on the Chicago Board Options Exchange (*CBoE*) [27], and was defined by *Securities Exchange Commission* in 1977 as the following:

> The practice of effecting an options transaction based upon non-public information regarding an impending block transaction[3] in the underlying stock, in order to obtain a profit when the options market adjusts to the price at which the block trades. [34]

On the years after there were ongoing discussions between self-regulatory exchanges (*e.g., CBoE*) and *SEC* to regulate, detect and define laws and policies to deal with front-running [27], with *SEC* stating:

> It seems evident that such behavior on the part of persons with knowledge of imminent transactions which will likely affect the price of the derivative security constitutes an unfair use of such knowledge. [4]

[3] Block in the stock market is large number of shares, 10 000 or more, to sell which will heavily changes the price

[4] Securities Exchange Act Release No. 14156, November 19, 1977, (Letter from George A. Fitzsimmons, Secretary, Securities and Exchange Commission to Joseph W. Sullivan, President CBoE).

Defining what exactly was considered illegal front-running required more knowledge of how these new markets behaved. *CBoE* and other exchanges (and brokers) issued educational circulars for their members asserting that front-running violates existing rules, with some examples of what is considered front-running. However difference of opinion regarding the unfairness of front-running activities, insufficient exchange rules and lack of a precise definition in this area resulted in no action by self-regulator organizations citesec1978optionsmarket.

Further reading on the early details on the history and challenges of detecting and regulating front-running can be found in [27] .

Initially the front-running policies only applied to certain option markets, later on in 2002, the rule was refined to include the same prohibitions to security futures [5], which then in 2012 with the new amendment, FINRA Rule 5270, the front-running rule was extended to cover trading in an option, derivative, or other financial instruments overlying a security with some exceptions [6, 35].

2.3 Blockchain Front-running

Blockchain technology has received an exceptional amount of attention since it was first introduced as the underlying technology of the cryptocurrency Bitcoin in 2008 [31]. Many decentralized applications that are built on top of this technology represent the significance of the blockchain as it completely eliminates the central point of failure within any systems. However, blockchains have some inherent characteristic that leave them vulnerable to *front-running* behaviour. Although this data structure is known to be publicly visible to every network participants, information is layered and some of them are accessed only by insiders. Any network participants that runs a full node on the blockchain is able to obtain those information and as mentioned in Section 2.1, this leaves the application vulnerable to front-running. Decentralized exchanges are a group of blockchain financial applications where front-running can be executed. Bancor [15] is an example of such systems in which front-runners benefit from potential price increase of the market stocks (details are provided in 4.1). Decentralized namespaces, as blockchain non-financial applications, are another example. Ethereum Name Service and Ghazal [30] are such systems in which upon seeing a transaction from Alice to register `alice.eth`, front-runner can go ahead of her and register this domain name. He can later sell `alice.eth` to Alice for a higher price and profit. Another application that is vulnerable to front-running is Initial Coin Offering (ICO) smart contracts. ICO is a method to distribute tokens based on the deposits of the native coin, Ether in the case of Ethereum. After a few hacking incidents on high valued smart contracts [36], ICOs started to implement restrictions and capped how much funds can be gathered. This scarcity of the initial coins made for a competition to incentivize big investors to get in and buy the tokens at a discounted price and sell them to late comers on the open markets [22, 45]. ICOs started to experiment with different fair capping methods, such as reverse dutch auction and dynamic ceilings [18]. This competition made ICOs a good opportunity for front-running attacks. In this paper we use multiple empirical examples to show possibility of front-running on DApps, for

the sake of completeness, we use one incident on ICOs, and examples from the Top 25 DApps by recent user counts.

2.4 Literature on Blockchain Front-running

Due to the novelty characteristics of the underlying technology (blockchains) and the issue of front-running, there are only a few research carried out on this subject in the context of blockchain and decentral ledgers.
Aune *et al.* discuss how the lack of time priority between broadcasting a transaction and its validation by miners on a blockchain based system would lead to the market information leakage [4]. As mentioned in Section 2.1, the information leakage results the entire system to be vulnerable to the front-running. They also propose a cryptographic approach, similar to commit and reveal(6.2), to prevent front-running in decentral systems. Malinova and Park discuss different design settings for financial markets and trading platforms on decentral ledgers [26]. They argue that blockchain brings a new options of transparency which affects traders behavior, specially in the presence of intermediary front-runners. They model indirect trading costs and use the economic literature on search and trading at decentralized exchanges in their design settings to make front-running costly and inefficient.

Breidenbach *et al.* [10] face the front-running issue in the context of automating bug bounties for smart contracts. Upon submitting the bug bounty to the Hydra smart contract, one can front-run the bug report and claim the bounty instead of the original reporter. In order to mitigate this issue they proposed *Submarine Commitments* (6.2).

One of the properties of Blockchain technology was to overcome double-spend, however this does not apply for fast payments and zero-confirmation transactions. *Double-spend attack*, as the name implies, the adversary spends a coin more than one time. To do so, he would use the coin as an input of a transaction and broadcast it to the network, before the transaction gets validated and executed, he broadcasts another transaction associated with a higher transaction fee which spends the same input [7, 20]. This adversarial behavior is similar to the front-running attack where the attacker front-runs his own transaction by paying a higher fee when broadcasting the second transaction.

3 Who are Blockchain Front-runners?

Blockchain, as a secure ledger, gives the impression of ultimate secure platform for decentral applications and one might not consider implementing the application with security in mind. In general, all the network participants have the ability to front-run specific transactions that are sent to the network. However, miners can include any transactions they like into the block they attempt to mine. Thus, they possess special power in terms of front-running as opposed to other users of the network. In the following, we discuss and compare the two groups of blockchain potential front-runners.

Blockchain Participants Any regular (miner or non-miner) user can front-run other transactions in the network. For regular users to front-run others on the blockchain, they need to be well connected to other nodes on the network. Doing so, they are able to listen to the network and monitor all transactions that are broadcast. On the Ethereum blockchain, users have to pay for the computations in small amount of Ether called *gas* [1]. The price that users pay for transactions (a.k.a. transaction fees) can increase or decrease how quickly miners will execute them and include them within the blocks they mine. This is because the Ethereum miners consume resources to process the transactions and so receive the transaction fees after creating the blocks. Once seeing two identical transactions with different transaction fees, a rational miner will prioritize the transaction that pays higher gas price per unit of gas –gasPrice–, this is due to limited space in the blocks while maximizes their profit in the block they mine. Therefore, any regular users who run a full-node Ethereum client can front-run pending transactions by sending similar transaction with higher gas price, see Figure 1. For example, in a decentral exchange, he can monitor the network and once seeing a pending *buy* transaction which will further increase the price of the asset, a front-runner can send a transaction with a higher gas price and hope to have his transaction mined ahead of the original pending transaction. By doing so, he achieves a better rate from the buyer. Note that in this case, blockchain front-runners are visible to the network as they broadcast their transactions to all the network participants.

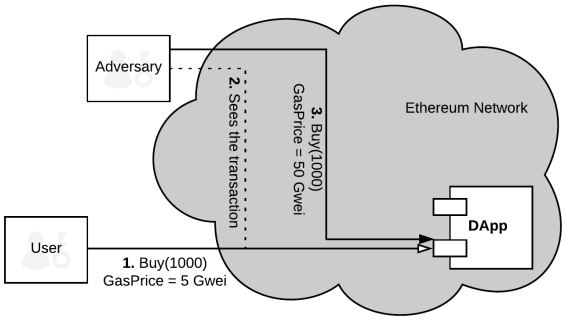

Fig. 1: The Adversary up on seeing the Buy order, sends his own buy order with higher gas to front-run the first order.

Miners Blockchain miners are the only parties who can decide on the order of transactions within a block they mine, they can easily intercept and reorder the transactions in their blocks, this is known as **transaction reordering** attack.

For example in an Ethereum based application, a miner learns about the pending *buy* transaction of 1000 units of a token*i.e.,* TKN, presumably if this transaction goes through, it causes the price of the purchased token to increase. So the dishonest miner can step in front of this transaction and place his own buy order ahead. He would simply create his *buy* of 1000 TKN and include it in front of the previous *buy* transaction of 1000 TKN in the block he mines. Doing so, he would receive a better rate than other network users, and afterwards can sell the assets he has bought and gain a price advantage at the expense of others. Similarly, a dishonest miner can sell his tokens in front, if he sees a pending *sell* transaction. In an open source blockchain application, there is no rules on how transactions must be ordered and miners are free to mine transactions in any order they prefer. However, miners can only front-run transactions within the blocks they happen to mine. They could do this also by not broadcasting their own transactions to the network, this makes the miners to be less noticeable to the network when front-running. Another example here is in the case of decentral exchanges and order cancellation. Every exchange should have the functionality to cancel the orders, especially for a volatile market. In this case, when user decides that her order is not profitable anymore, she would send a cancelation transaction. Here the miner sees the cancelation transaction and puts his own order in front of the cancellation transaction –transaction reordering– to fill the order, potentially profiting from the order. They can also include the canceled transaction in their block to collect the full gas limit used by that transaction [17], see Figure 2. Front-running can also occur in non-financial smart contracts. As an example, smart contract which adds all the participants to a party invite list, could only be closed by the smart contract owner. In this case when the owner sends the transaction to close the list, a miner can include his own list of participants before the close transaction in the block he mines. This is an issue with the design of the smart contracts and is known as transaction-ordering dependence vulnerability [25].

4 Cases of Front-running in DApps

As mentioned in the Section 1, for the purpose of this research we retrieved the top 25 DApps based on recent user activities from [5] in the first week of September 2018. We then categorize these applications into 4 principal groups of decentral applications (see Table 1). In this section, we take a deep look into one or two decentral applications from each category and pinpoint the technical details of front-running issue on each case. Although, no ICOs were spot in the list at the time it was retrieved, we will thoroughly discuss our analysis on how ICOs are highly vulnerable to front-running activities (see Section 5).

[5] List of decentralized applications `https://dappradar.com/dapps`

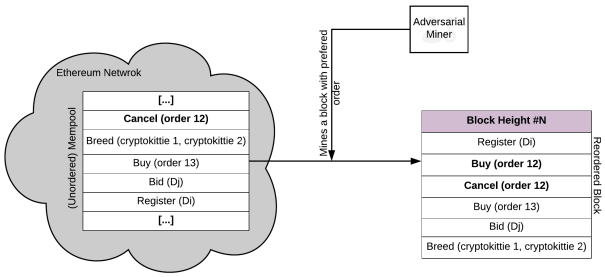

Fig. 2: The adversarial miner can monitor the Ethereum mempool for decentral exchange cancel orders and upon seeing the cancellation transaction, he puts his buy order before ahead. Doing so, the miner can profit from the underlying trade and also get the Gas included in the cancel transaction.

4.1 Markets and Exchanges.

Exchanges such as EtherDelta[6], are closest form to decentral exchanges, however their order books are stored on a central server they control and shown on their user interface (website). Other than the fact that they can re-order or censor the orders on their servers, there are other risks of front-running nature in these designs. Similarly 0xProtocol uses *Relayers* which act as the orderbook holders and are not prune to cheating. One main issue with these exchanges is *cancellation grief*. Although the orderbooks are not in the smart contract, in order to prevent denial of service attacks, user needs to send a transaction to cancel her order. In this case, when an adversarial actor sees the cancellation transaction, he sends a fill order transactions with higher gasPrice to get in front of the cancellation order and take the order before it is cancelled. As mentioned in 3, miners are in a better control to perform such attack.

Bancor is another Ethereum-based application that allows users to exchange their tokens without any counter-party risk. This protocol aims to solve the cryptocurrency liquidity issue by introducing *Smart Tokens* [15]– ERC20 compatible tokens with a built-in liquidity mechanism that are always available to users. Smart Tokens can be bought and sold through the users smart contract at an automatically calculated price which displays supply and demand. Doing so, Bancor protocol provides continuous liquidity for digital assets without relying on an order book as there is no requirement to match sellers and buyers. Recently, researchers have shown that Bancor is vulnerable to front running attacks. Implemented on the Ethereum blockchain, when Bancor transactions are broadcast to the network, they sit in a pending transaction pool known as *mempool* waiting for the miners to mine them. Since Bancor handles all the trades and exchanges on the Ethereum blockchain (unlike other existing decentralized exchanges), these transactions are all visible to the public for 16s (the

[6] also known as ForkDelta for the UI `https://forkdelta.app/`

DApp Category	DApp Names (Rank)
Exchanges	IDEX (1) *ForkDelta, EtherDelta (2)* *Bancor (7)* The Token Store(13) LocalEthereum(14) 0x Protocol (23) Kyber (22)
Crypto-Collectible Games (ERC 721)	*CryptoKitties (3)* Ethermon(4) Cryptogirl (9) Gods Unchained TCG (12) Blockchain Cuties(15) ETH.TOWN! (16) 0xUniverse (18) MLBCrypto Baseball (19) HyperDragons(25)
Gambling	Fomo3D (5) DailyDivs (6) FomoWar (10) FairDapp (11) Zethr(17) dice2.win (20) PoWH 3D (8) Ether Shrimp Farm(21)
Name Services	*Ethereum Name Service (24)*

Table 1: Top 25 DApps based on recent user activity from `Dappradar.com` on September 4th, 2018. Dapps that are represented in Italic format are analyzied and discussed in this paper.

average Ethereum blocktime) before being included within a block. This leaves this blockchain-based decentralized exchange vulnerable to the blockchain race condition attack as attackers are given enough time to front-run other transactions and gain favorable profits [14]. As mentioned in Section 3 and same as every other Ethereum based application, Bancor transactions can be front run by miners and other regular (non-miner) users. There is an implementation of a Bancor front running attack in the Python programming language which represent how a non-miner user can front run other network participants [8].

4.2 Name Services.

As mentioned in Section 2.3, although front-running attacks have been more showcased in the context of decentralized exchanges and trading systems, they are not yet limited to the financial markets. Front-running can also occur within other (non-financial) blockchain applications such as naming systems. Recently, many blockchain-based domain registrars have been designed and implemented

to eliminate the role of central parties *i.e.,* domain name system (DNS) which itself introduces single point of failures in the entire web. Ethereum Name Service (ENS)[7], is the most popular decentral naming service that uses a sealed auction to facilitate the procedures of purchasing new `.eth` domain names. These domains which can be resolved using any Ethereum web3 compatible browser. Although a sealed auction (*startAuctionsAndBid*) is used while implementing the ENS, the deposit should be equal or bigger than the bid, this information leakage could be used to front-run the transaction with the same bid amount. By doing so the front-runner invalidates the original bid and hence registers the domain name himself. Ghazal is another Ethereum-based naming system that allows users to register `.ghazal` domain names and bind certificates to those names [30]. In Ghazal model, a user would register domain name by executing the *registerdomain* function from the Ghazal smart contract with the domain name in plain text as the function argument. As mentioned before, These transactions will sit in the mempool so that it would be mined by Ethereum miners and included in the block. During this period in which the transaction is not yet confirmed, front-running attack can occur by (i) dishonest miners and/or (ii) regular non-miner user. In the first case, a miner would intercept the *registerdomain* transaction and register that name ahead of the user. A regular non-miner node in the Ethereum network can front-run other user's *registerdomain* at a good profit by paying higher transaction fees. This practice allows the adversarial party to register an unregistered domain immediately after someone searched for it, and sell the domain name to the users for a higher price. Note that, domain name front-running also occurs in the vanilla system. In this case, front-runners would monitor and identify domain names of interest, and sell it back to the other users with a higher price [12]. Front-running practice can happen in the traditional domain name system (DNS) as well. Domain name registrars can steal available domain names once seeing a query from users that looks up that domain's availability. In this case, the registrar can register the domain name ahead and sell it back to the user for a higher price.

4.3 Crypto-Collectibles Games.

Crypto-Collectibles, also known as ERC-721 Non-Fungible Token Standard [44], are one of the applications of Ethereum blockchain. Cryptokitties is known to be one of the first and the most popular use cases of ERC721 tokens [37]. Their market cap reached more than 6 million dollars in the first months their launch. Crypto-Collectibles, are similar to ERC-20 tokens and can be listed on exchanges. Most of the front-running issues applied to the exchanges also apply to DApps using ERC-721 tokens.

As an example Cryptokitties can be bred, bought or sold through the main interface[8]. See Figure 3. To buy a cryptokittie, user sends a bid transaction

[7] `https://ens.domains/`
[8] Cryptokitties website `https://www.cryptokitties.co/`

bid(uint256 _tokenId) containing the cryptokittties ID which can be found either on their website or on the ERC721 token smart contract. This is similar to open auctions, bid value and the object bidding on is visible to the network and any user could be easily front-run by sending the same transaction with higher gasPrice to replace the initial bidder.

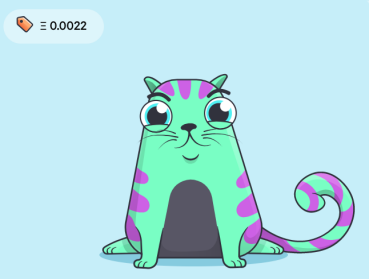

Fig. 3: Cryptokitty Number 842912

4.4 Gambling.

Due to deterministic nature of blockchains, running gambling games based on randomness is close to impossible. There had been attempts to use block headers or such as the seed to randomness but almost all have failed as every data accessible to the smart contract at the time of execution, is also accessible to other smart contracts and they could predict the output on the run, thus prone to front-running and miner's manipulation. Many infamous gambling games are now running on Ethereum network and front-running them requires a level of sophistication and knowledge of the internal of the games and Ethereum.
One of the most popular smart contracts on Ethereum blockchain belongs to a controversial gambling game named `Fomo3d`, also known as *Exit Scam*[9]. The aim of this game is to be the last person purchasing the key when the time goes to zero in which every ticket purchase increases the timer by 30 seconds. It was believed that because of the nature of the game, it will never end, however on August 22, 2018 the first round of the game ended and the winner took 10,469 Ether[10], equivalent to 2.1 Million dollars on the time of writing. It is believed that the winner, used a sophisticated approach to finish last [3], we call this approach *Denial-of-service Front-running*. The winner deployed a few high gas consumption smart contracts prior to the win, when the timer of the game reached about 3 minutes, the winner bought 1 ticket and sent multiple high gasPrice transactions to aforementioned smart contracts to lure the miners

[9] `https://exitscam.me/play`
[10] First winner of Fomo3D, won 10,469 Ether `https://etherscan.io/tx/0xe08a519c03cb0aed0e04b33104112d65fa1d3a48cd3aeab65f047b2abce9d508`

to mine these highly profitable transactions. For a DoS front-running to work, attacker floods the network with high gasPrice transactions to prevent other transactions to go through. Luckily enough the winner was able to congest the network in a way that no other ticket purchases were mined in the 3 minutes window and the miners were prioritizing the high gasPriced transactions over other transactions. All the details mentioned here are drawn from the transaction history on the blockchain and unless otherwise proven seem to be the way the winner acted to win the pot.

5 Cases of Front-running in ICOs

There has been more than 3000 ICOs on Ethereum alone, estimated to exceed 75 billion dollars in the first half of 2018 [45]. On our DApp samples there was not ICOs mainly because they tend to run for a short time with high number of participant and in the time of writing no ICOs have reached the top 25 DApps of `DAppradar.com`. Needless to say the importance of ICOs as an application of blockchain requires more in-depth analysis of front-running on these DApps. Here we focus on one ICO that our preliminary research showed that there might be a front-running attack going on. There is no doubt that similar behaviour could be seen on other instances of ICOs or similar DApps if investigated.

5.1 *Status.im* ICO

As discussed in 2.3, capped ICOs are considerable applications to be investigated in terms of the front-running attacks, *Status.im* ICO is such application. In June 2017, *Status.im* [39] started its ICO and reached the defined cap within 3 hours, resulting in close to 300,000 Ether of funds. In order to prevent few large investments to buy up all the tokens and limit the amount of Ether deposited in an investment, they used a *fair* token distribution method named *Dynamic Ceiling* as an attempt to increase the time window for smaller investors. They implemented multiple caps –ceiling– in which, each had a maximum amount that could be deposited in. In this case, every deposit was checked by the smart contract and the exceeding amount was refunded to the sender while the accepted amount was sent to their multisignature wallet address [32].

During the time the ICO was open for participation, there were reports of Ethereum network being unusable and transactions were not confirming. Further study showed that some mining pools might have been manipulating the network for their own profit. In addition, there were many transactions sent with higher gas price to front-run other transactions, however these transactions were failing due to the restriction in the ICO smart contract to reject any transactions with higher than 50 *GWei* gas price (as an attempt to prevent the known front-running behavior).

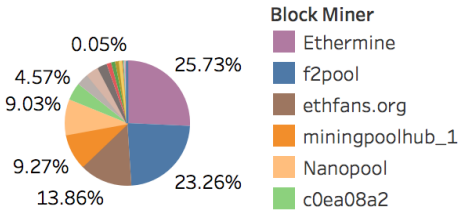

Fig. 4: The percentage of Ethereum blocks mined between block 3903900 and 3908029, this is the time frame in which Status.im ICO was running. This percentage roughly shows the hashing power ratio each miner had at that time.

5.2 Data Collection and Analysis

According to the analysis we carried out, it was discovered that the F2Pool, an Ethereum mining pool that had around 23% of the mining hash rate at the time (Figure 4), sent 100 Ether to 30 new Ethereum addresses before Status.im ICO started. On the time the ICO was opened for participation, F2Pool constructed 31 transactions to the ICO token sale smart contract from the addresses they controlled –without broadcasting these transactions to the network– and used their entire mining power to mine their own transactions and also potentially failing high gas price transactions.

Ethereum blockchain contains all transaction ever made on Ethereum, however, it is not easy to see the behavior of the participants and miners solely using the default clients nor online blockchain explorers. In order to get the required data, Ethereum blockchain data should be converted to a queryable format. For this case study, we used open source projects such as Go Ethereum implementation[11] for the full node, python script for extracting, transforming and loading ethereum blocks, named `ethereum-etl` [28] and Google BigQuery[12]. Using this software stack, we were able to query for all the transactions and blocks within the window of Status.im ICO for further analysis which was mainly done by data analysis tool `Tableau`[13]. A copy of this dataset and the initial findings can be found in our Github repository[14].

As shown in Figure 5, most of the top miners in the mentioned time frame, have mined almost the same number of failed and successful transactions which were directed toward Status.im token sale, however F2Pool's transactions indicate their successful transactions were equivalent to 10% of the failed transactions, hence maximizing the mining rewards, on gas, while censoring other

[11] Official Go implementation of the Ethereum protocol`https://github.com/ethereum/go-ethereum`

[12] A fast, highly scalable, cost-effective, and fully managed cloud data warehouse for analytics `https://cloud.google.com/bigquery/`

[13] Tableau is a data visualization software that is used for data science and business intelligence.`https://www.tableau.com/`

[14] `github.com/ANONYMOUS`

transactions to the token sale smart contract. The terminology used here is specific to smart contract transactions on Ethereum, by *"failed transaction"* we mean the transactions in which the smart contract code flow rejected and threw an exception and by *"successful transaction"* we mean the transactions that went through and received tokens from the smart contract.

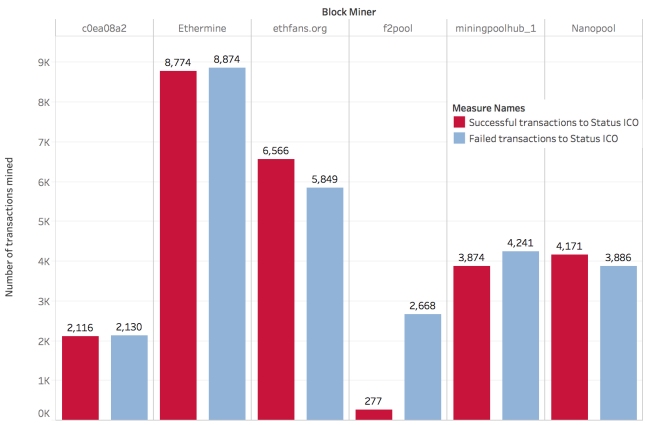

Fig. 5: This chart shows the miners behaviour on the time window that Status.im ICO was running. It is clear that the number of successful transactions mined by F2Pool do not follow the random homogeneous pattern of the rest of the network.

By tracing the transactions from these 30 addresses, we found explicit Interference of the F2Pool[15] in this scenario. As shown in Figure 6, the funds deposited by F2Pool in these addresses were sent to Status.im ICO and mined by F2Pool themselves, where the dynamic ceiling algorithm refunded a portion of the deposited funds back to these addresses, these funds were sent back to F2Pool main address a few days after.

Although this incident does not involve transaction reordering in the blocks, it shows how miners can modify their mining software to behave in a certain way to front-run other transactions and take advantage of the monetary profit. The dataset used for this analysis can be found on our GitHub repository.[16]. This analysis made us wonder how many other DApps are vulnerable to front-running attack, who can front-run and what are the mitigation methods for decentral applications?

[15] F2Pool Ethereum address was identified by their mining reward deposit address `https://etherscan.io/address/0x61c808d82a3ac53231750dadc13c777b59310bd9`

[16] Frontrunning Status.im ICO dataset: `http://github.com/TODO:ANONYMOUS`

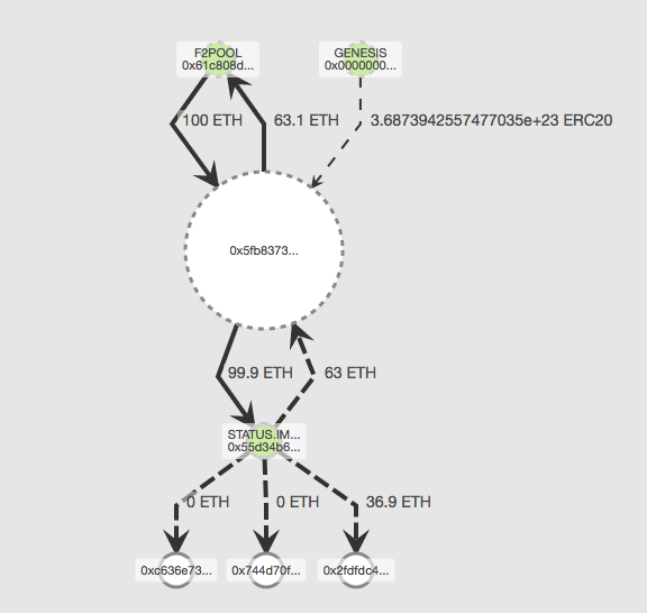

Fig. 6: F2Pool prior to Status.im ICO deposited 100 Ether in multiple new Ethereum addresses. On the time of the ICO they sent these deposits to Status ICO smart contract and prioritized mining of these transactions in their mining pool, this was to overcome the dynamic ceiling algorithm of the token sale smart contract. Later on they sent the refunded Ether back to their own address. [Graph made using Blockseer.com]

6 How to stop Front-running?

In the traditional markets, front-running is considered unethical and also illegal. In the blockchain space, the designers of the decentral applications cannot rely on the justice system for unethical behavior, and they must assume that each participant in the network acts rationally in their own self interest, the application will be operating in an adversarial environment [42]. A few decentral exchange projects such as *EtherDelta* and *0xProject* [43], have proposed solutions for front-running, which is off-chain order books. The methods discussed within these projects prevent blockchain network participants from front running the orders as the orders are private and not known to the network and will not be broadcasted. However they are introducing third parties, *e.g.,* relayers in 0xproject, to be managing the orders with the promise not to reorder or front-run other orders.

Traditional Front-running Prevention Methods. There are debates in traditional markets regarding the fact that Front-running is considered to be a form of insider trading which deemed to be illegal. Traditional methods to prevent front-running mainly involves after the fact investigation and legal action against

the front-runners [40]. As mentioned in section 2.2, educating the employees were the first step taken to prevent such issues in traditional markets, however front-running became less likely to happen mainly because of the high fine and law suits against firms whom behaved in an unethical way. Other methods such as dark pools [11,46] and sealed bids [33] were discussed and implemented in variety of regulated trading systems.

The traditional methods to prevent front-running does not apply to blockchain applications, as mainly they are based on central enforcement and limitations, also in case of blockchains the actors whom are front-running could be anonymous and the fear of lawsuits would not apply.

There are two main approaches to prevent front running, one to design a blockchain that is front-run resistant , and the other to design the application logics in a way that front running is not possible.

6.1 Front-run Resistant Blockchain

What does this mean to have a front-run resistant blockchain? There are technical difficulties to achieve such solutions as there are unknown factors within such network designs (corner/edge cases). In this section, we describe the potential solutions using which, one can design and implement a decentral application that is resistant to front-running.

Design Decision #1: Fixed Transaction Ordering
One possible solution to fix current blockchains in regards to transaction reordering by miners has been proposed by da Silva *et al.* [13]. This solution consists of an algorithm, known as Fixed Transaction Ordering and Admission (FTOA) algorithm, in the consensus protocol that enforces the order of the transactions.

Design Decision #2:Privacy Preserving Blockchain
Market participants are concerned about revealing information on their past trades, purchases, income, or liquidity needs. Privacy is a basic human right, that being said, there are limitations with the functionality of a system designed to preserve full privacy. However other than past transactions, real time transactions could leak informations about the sender or the intention of the order that could lead to front-running attacks. Privacy preserving blockchains try to solve the initial issue by keeping all the details of the transactions private, in some extent [19, 29]. As an example, ZCash [16], uses two distinct type of addresses, transparent addresses and shielded addresses. Transparent addresses work similar to Bitcoin transactions, fully transparent about the sender and receiver addresses, the amount and included data. However shielded addresses are private and do not leak any information, this is great for privacy but not good for smart contracts capabilities. Smart contract functionality requires verification by the miners which require to know the input, functionality and output

of smart contracts to be able to determine if the transaction is valid. Albeit by using cryptographic primitives such as zero-knowledge proofs this issue could also be resolved [21].

Design Decision #3: Dual Authoring

Loopring is an open protocol proposed by Wang *et al.* to build decentralized exchanges [41]. This protocol operates a hybrid model with off-chain order management and communication and merely the order matching takes place on the chain. To achieve high liquidity and price improvement, Loopring orders are grouped in *ring orders*. After the required signatures are provided by a miner and owner of the order, see Figure 7, the ring miner sends the *submitRing transaction* to the Loopring Smart Contract (LPSC) for verification and settlement. While the *submitRing transaction* sits unconfirmed in the mempool, any front-runner can create a copy of this transaction with his address instead of the ring miner's address and then sign the hash of the ring using his own private key. By paying a higher gasPrice, block miners choose the front-runner's transaction instead of the original *submitRing transaction*. However, Loopring's team solves this issue using the *dual-authoring* scheme in the new version of the Loopring protocol. This solution sets two levels of authorization for orders - one for settlement, and one for order matching. As it can be seen in Figure 8, each owner of any order generates a pair of public key (auth-address) and private key (auth-key), which are different from his address and its corresponding private key. These auth-keys are used to sign the hash of the ring and will further prevent anyone from regenerating the same *submitRing transaction*. The reason is that the auth-keys are only known to the miner of the ring as they are not part of on-chain transaction. Doing so, it is assured that (i) the orders are not modified (because they are signed using the owner's private key), (ii) no other miner can mine the ring (because the miner's address is signed using his private key), and no user can front-run the *submitRing transaction* (because other users do not have access to the auth-keys and hence are not able to generate new transactions with auth-keys signatures on them).

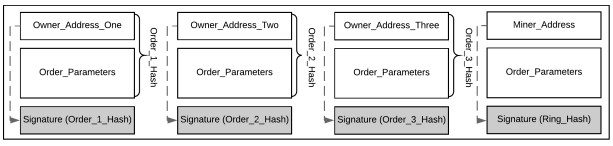

Fig. 7: *submitRing* Transaction in the old version of the Loopring protocol. Any user can regenerate this transaction by replacing the miner's address with his address and signing the hash of the ring using his private key.

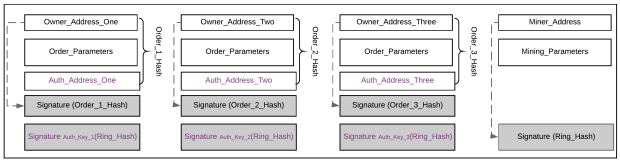

Fig. 8: *submitRing* Transaction in the new version of the Loopring protocol. Other users cannnot regenerate this transaction as they do not access auth-keys and hence are not able to create the signatures.

6.2 Application Design to Prevent Front-running

Another method to prevent front-running activities is to design the applications, whether they are decentralized (Dapps) or centralized, in such a way that no information would be available to the malicious parties to enable them to front-run the other users. Followings discuss the two techniques which can be used by the Apps/DApps developers while designing the systems to prevent front-running activities.

Design Decision #1: Commitment Scheme

Commitment scheme is a cryptographic primitive that enables one to commit to a value –binding– (*e.g.,* statement, document, data, *etc.*) while keeping it a secret and reveal the committed value on a later time [9]. Commitment scheme is a robust method to prevent information leakage from the sensitive transactions. This could be done by simply broadcasting the hash value of the committed data and later on revealing the values that would be hashed to the committed hash, see Figure 9. In the case of decentral exchanges, user can send a commit transaction which will be cryptic to network participants but will act as a placeholder in the queue for the user, after the transaction is mined, user sends reveal transaction revealing the order details which will be executed in a fair order. Depending on the Dapp design, either the order is collateralized, which means the commitment transaction includes the funds required for fulfilling the order, which will leak some information about the order, or is not collateralized and opens up the possibility of user never revealing the commitment, which will be considered as a DoS attack. Another use of such commitment scheme, is in decentralized naming systems, such as Ghazal or ENS. In this case, a user would send a commit transaction similar to a sealed bid, once the transaction is confirmed and the grace period is over, user would send a reveal transaction revealing the bid and also the details of the requested domain. Using this scheme, one is able to hide information from the adversarial parties in the system and prevent them from front-running. Similar approach applies to on-chain voting as well [2], using sealed votes, voters can make sure their votes are hidden until a later date, however their participation and weight of the vote are publicly known. Other than the collateralization issue for the commit and reveal scheme, another

obvious issue is the participatory factor. For anyone watching the DApp address, there will be a direct transaction from the user to the DApp address, It will be obvious that a specific address has participated in the auction or the DApp using a commit and reveal scheme, but the details are hidden through the scheme. Both these issues, albeit hiding the details of the order and preventing direct front-running, leak information to other participants which could lead to more sophisticated front-running attacks depending on the application design. On the other note, these method has some drawbacks, the user experience is not smooth and one might forget to reveal the committed transaction. In ENS commit-reveal process, if one forgets to reveal on the reveal phase, the committed funds are burnt and inaccessible there after. Also this scheme means two transaction are required for a functionality which is makes it more expensive to run.

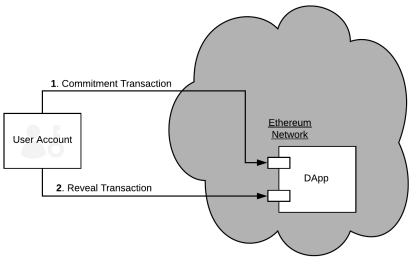

Fig. 9: Commit and Reveal. User sends a commitment transaction with the hash of the data, After the commitment period is over, user sends her reveal transaction to the DApp revealing the information that matches the commitment.

Design Decision #2: Submarine Commitments

Submarine Commitments [10, 24] are similar to the commitment schemes with some key differences such as hiding participatory factor and being fully collateralized. However, they could be design in other ways to suit the need of the DApp. With submarine send, it is possible to hide sender, receiver, value and data, the commitment transaction is identical to a transaction to a newly generated address. Submarine sends could be a solution for sealed-bid auctions on the blockchain to hide the existence of the bids for other participants.

The details of how Submarine commitments work is outside the scope of this paper, however in summary, using the way Ethereum transactions are constructed and ECDSA ECRecover functionality it is possible to generate one outgoing transaction from an address in which no private key exists. By constructing a transaction from that address to the DApp and funding that address we fulfill the commitment phase and by revealing the details and broadcast-

ing the constructed transaction we reveal the commitment [38], see Figure 10. Drawbacks of this scheme are similar to 6.2, with an addition of two more transactions to finalize the functionality, which results in more cost and complicated user experience.

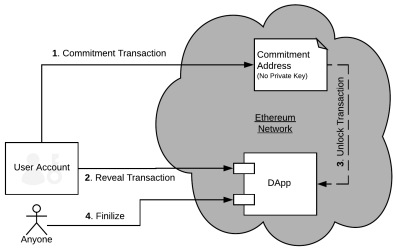

Fig. 10: Submarine Send [38]. User generates an *Unlock* transaction from which the commitment address is retrieved. By funding the *commitment address*, user is committed to the unlock transaction. After the commitment transaction, in the reveal phase, user sends the *reveal transaction* to the DApp and then after she can broadcast the Unlock transaction to unlock the funds in the commitment address. Finally after the "Auction" is over, anyone can call *Finalize* function to finalize the process.

Design Decision #3: Logic Specific Solutions

To prevent front-running, one can design the application in a way that it is not possible to be front-run:

Eliminating Time-Order Dependency: The applications can be designed to not rely on the time for the orders to be executed. For example when designing a decentral exchange, one can use *the call market design* instead of a time-sensitive order book. In a call market design, the arrival time of the orders do not matter as they are executed in batches [cite Jeremy's prediction markets paper].

Disincentivizing Front-running Actors: This solution, which is economic rather than technical, is to disincentives the miners by paying them. As mentioned, the main reasons for miners to front-run the other orders is the financial gain they receive on the price improvement. However, if the application is designed in a way that the fees of the orders are sent to the miner of the block, he would have no incentive to front-run the orders as he already gains enough financial benefit from acting fairly and executing the transactions on their correct order. For example, Malinova [26] proposes a design for a decentral exchange which uses economic models in their trading market design to make front-running expensive and more costly than profitable.

```
    function createEscrow(bytes16 _tradeID, address _seller, address _buyer,
        uint256 _value, uint16 _fee,
          uint32 _paymentWindowInSeconds, uint32 _expiry, uint8 _v, bytes32
              _r, bytes32 _s)
        payable external {
        bytes32 _tradeHash = keccak256(abi.encodePacked(_tradeID, _seller,
            _buyer, _value, _fee));
  ...
        // A signature (v, r and s) must come from localethereum to open an
            escrow
        bytes32 _invitationHash = keccak256(abi.encodePacked(
            _tradeHash,
            _paymentWindowInSeconds,
            _expiry
        ));
```

Code 1.1: Code snippet from LocalEthereum smart contract. Values V,R and S are set by LocalEtherem to have a valid signature, also the tradeHash uses buyer and seller addresses, mitigating the possibility of front-running by a third party.

Going back to Table 1, some DApps facilitate different methods to mitigate front-running issues. As an example LocalEthereum, an escrow-based exchange, uses challenge random values from the website to be included in the signature of the transaction (See 1.1), hence removing any possibility of replay attack or possibility of any other address sending the similar order.

7 Concluding Remarks

References

1. Account types, gas, and transactions. ethereum homestead 0.1 documentation. `http://ethdocs.org/en/latest/contracts-and-transactions/account-types-gas-and-transactions.html#what-is-gas`. (Accessed on 06/14/2018).
2. adChain. Dev diary i: A walkthrough of plcr voting in solidity. `https://medium.com/metax-publication/a-walkthrough-of-plcr-voting-in-solidity-92420bd5b87c`, 2017. Accessed: 2018-08-28.
3. Anonymous. How the first winner of fomo3d won the jackpot? `https://winnerfomo3d.home.blog/`, 2018. Accessed: 2018-09-09.
4. R. T. Aune, A. Krellenstein, M. OHara, and O. Slama. Footprints on a blockchain: Trading and information leakage in distributed ledgers. *The Journal of Trading*, 12(3):5–13, 2017.
5. F. I. R. Authority. *IM-2110-3. Front Running Policy.* 2002.
6. F. I. R. Authority. *5270. Front Running of Block Transactions.* 2012.
7. T. Bamert, C. Decker, L. Elsen, R. Wattenhofer, and S. Welten. Have a snack, pay with bitcoins. In *Peer-to-Peer Computing (P2P), 2013 IEEE Thirteenth International Conference on*, pages 1–5. IEEE, 2013.
8. I. Bogatyy. Implementing ethereum trading front-runs on the bancor exchange in python. `https://hackernoon.com/front-running-bancor-in-150-lines-of-python-with-ethereum-api-d5e2bfd0d798`, 2017. (Accessed on 08/13/2018).

9. G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.
10. L. Breidenbach, I. Cornell Tech, P. Daian, F. Tramer, and A. Juels. Enter the hydra: Towards principled bug bounties and exploit-resistant smart contracts. In *27th {USENIX} Security Symposium ({USENIX} Security 18)*. {USENIX} Association, 2018.
11. S. Buti, B. Rindi, and I. M. Werner. Diving into dark pools. 2011.
12. I. A. Committee. Ssac advisory on domain name front running. `https://www.icann.org/en/system/files/files/sac-022-en.pdf`, 10 2007. (Accessed on 08/15/2018).
13. P. M. da Silva, M. Matos, and J. Barreto. Fixed transaction ordering and admission in blockchains.
14. P. D. Emin Gun Sirer. Bancor is flawed. `http://hackingdistributed.com/2017/06/19/bancor-is-flawed/`, 2017. (Accessed on 06/14/2018).
15. E. Hertzog, G. Benartzi, and G. Benartzi. Bancor protocol. 2017.
16. D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification. Technical report, Technical report, 2016–1.10. Zerocoin Electric Coin Company, 2016.
17. P. D. A. J. Y. L. X. Z. Iddo Bentov, Lorenz Breidenbach. The cost of decentralization in 0x and etherdelta. `http://hackingdistributed.com/2017/08/13/cost-of-decent/`, 2017. (Accessed on 08/14/2018).
18. W. Kaal and M. Dell'Erba. Initial coin offerings: Emerging practices, risk factors, and red flags. 2017.
19. G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. An empirical analysis of anonymity in zcash. *arXiv preprint arXiv:1805.03180*, 2018.
20. G. O. Karame, E. Androulaki, and S. Capkun. Double-spending fast payments in bitcoin. In *Proceedings of the 2012 ACM conference on Computer and communications security*, pages 906–917. ACM, 2012.
21. A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *2016 IEEE symposium on security and privacy (SP)*, pages 839–858. IEEE, 2016.
22. J. Li and W. Mann. Initial coin offering and platform building. 2018.
23. D. Liang. Distressed sales and financial arbitrageurs: Front-running in illiquid markets. 2005.
24. A. J. Lorenz Breidenbach, Phil Daian and F. Tramer. To sink frontrunners, send in the submarines. `http://hackingdistributed.com/2017/08/28/submarine-sends/`, 2017. Accessed: 2018-08-28.
25. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.
26. K. Malinova and A. Park. Market design with blockchain technology. 2017.
27. J. W. Markham. Front-running-insider trading under the commodity exchange act. *Cath. UL Rev.*, 38:69, 1988.
28. E. Medvedev. Python scripts for etl (extract, transform and load) jobs for ethereum blocks. `https://github.com/medvedev1088/ethereum-etl`, 2018. Accessed: 2018-08-31.
29. A. Miller, M. Möser, K. Lee, and A. Narayanan. An empirical analysis of linkability in the monero blockchain. *arXiv preprint*, 1704, 2017.
30. S. Moosavi and J. Clark. Ghazal: toward truly authoritative web certificates using ethereum.

31. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
32. C. Petty. A look at the status.im ico token distribution. `https://medium.com/the-bitcoin-podcast-blog/a-look-at-the-status-im-ico-token-distribution-f5bcf7f00907`, 2017. Accessed: 2018-06-10.
33. R. Radner and A. Schotter. The sealed-bid mechanism: An experimental study. *Journal of Economic Theory*, 48(1):179–220, 1989.
34. SEC. 96th cong, 1st sess, report of the special study of the options markets to the securities and exchange comission. *Comm. Print 1978*, pages 183–189, 1978.
35. SEC. Notice of filing of proposed rule change to adopt finra rule 5270 (front running of block transactions) in the consolidated finra rulebook. *Release No. 34-67079; File No. SR-FINRA-2012-025*, 2012.
36. D. Siegel. Understanding the dao attack. *Web. http://www. coindesk. com/understanding-dao-hack-journalists*, 2016.
37. C. team. Cryptokitties. `http://www.cryptokittieswiki.com/CryptoKitties/`, 2018. Accessed: 2018-08-31.
38. L. Team. Libsubmarine: Temporarily hide transactions on ethereum (cheaply!). `https://hackernoon.com/libsubmarine-temporarily-hide-transactions-on-ethereum-cheaply-6910191f46f2`, 2018. Accessed: 2018-08-28.
39. S. Team. The status network, a strategy towards mass adoption of ethereum. `https://status.im/whitepaper.pdf`, 2017. Accessed: 2018-06-10.
40. D. Wang, J. Zhou, A. Wang, and M. Finestone. Lexisnexis. 2015. Accessed: 2018-09-12.
41. D. Wang, J. Zhou, A. Wang, and M. Finestone. Loopring: A decentralized token exchange protocol. *URL https://github. com/Loopring/whitepaper/blob/master/en_whitepaper. pdf*, 2018.
42. W. Warren. Front-running, griefing and the perils of virtual settlement. `https://blog.0xproject.com/front-running-griefing-and-the-perils-of-virtual-settlement-part-1-8554ab283e97`, 2017. (Accessed on 08/14/2018).
43. W. Warren and A. Bandeali. 0x: An open protocol for decentralized exchange on the ethereum blockchain. *URl: https://github. com/0xProject/whitepaper*, 2017.
44. J. E. N. S. William Entriken, Dieter Shirley. Erc-721 non-fungible token standard. `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md`, 2018. Accessed: 2018-08-31.
45. D. A. Zetzsche, R. P. Buckley, D. W. Arner, and L. Föhr. The ico gold rush: It's a scam, it's a bubble, it's a super challenge for regulators. 2018.
46. H. Zhu. Do dark pools harm price discovery? *The Review of Financial Studies*, 27(3):747–789, 2014.