# SoK: Transparent Dishonesty: front-running attacks on Blockchain.

Shayan Eskandari[†‡], Seyedehmahsa Moosavi[†], Jeremy Clark[†]

[†] Gina Cody School of Engineering and Computer Science
Concordia University
[‡] ConsenSys Diligence

**Abstract.** We consider *front-running* to be a course of action where an entity benefits from prior access to privileged market information about upcoming transactions and trades. Front-running has been an issue in financial instrument markets since the 1970s. With the advent of the blockchain technology, front-running has resurfaced in new forms we explore here, instigated by blockchain's decentralized and transparent nature. In this paper, we draw from a scattered body of knowledge and instances of front-running across the top 25 most active decentral applications (DApps) deployed on Ethereum blockchain. Additionally, we carry out a detailed analysis of Status.im initial coin offering (ICO) and show evidence of abnormal miner's behavior indicative of front-running token purchases. Finally, we map the proposed solutions to front-running into useful categories.



## 1 Introduction

Blockchain technology enables decentralized applications (DApps) or smart contracts. Function calls (or transactions) to the DApp are processed by a decentralized network. Transactions are finalized in stages: they (generally) first relay around the network, then selected by a miner and put into a valid block, and finally, the block is well-enough incorporated that is unlikely to be reorganized. Front-running is an attack where a malicious node observes a transaction after it is broadcast but before it is finalized, and attempts to have its own transaction confirmed before or instead of the observed transaction.

The mechanics of front-running works on all DApps but front-running is not necessarily beneficial, depending on the DApp's internal logic and/or as any mitigations it might implement. Therefore, DApps need to be studied individually or in categories. In this paper, we draw from a scattered body of knowledge regarding front-running attacks on blockchain applications and the proposed solutions, with series of case studies of DApps deployed on Ethereum (a popular blockchain supporting DApps). We do case studies on decentralized exchanges (*e.g.,* Bancor), crypto-collectibles (*e.g.,* CryptoKitties), gambling services (*e.g.,* Fomo3D), and decentralized name services (*e.g.,* Ethereum Name Service). We also study initial coin offerings (ICOs). Finally, we provide a categorization of techniques

to eliminate or mitigate front-running including transaction sequencing, cryptographic techniques like commit/reveal, and redesigning the functioning of the DApp to provide the same utility while removing time dependencies.

## 2 Preliminaries & Related Work

### 2.1 Traditional Front-running

*Front-running* is a course of action where someone benefits from early access to market information about upcoming transactions and trades, typically because of a privileged position along the transmission of this information and is applicable to both financial and non-financial systems. Historically, floor traders might have overheard a broker's negotiation with her client over a large purchase, and literally race the broker to buy first, potentially profiting when the large purchase temporarily reduces the supply of the stock. Alternatively, a malicious broker might front-run their own client's orders by purchasing stock for themselves between receiving the instruction to purchase from the client and actually executing the purchase (similar techniques can be used for large sell orders). Front-running is illegal in jurisdictions with established securities regulation.

Cases of front-running are sometimes difficult to distinguish from related concepts like insider trading and arbitrage. In front-running, a person sees a concrete transaction that is set to execute and reacts to it before it actually gets executed. If the person instead has access to more general privileged information that might predict future transactions but is not reacting at the actual pending trades, we would classify this activity as insider trading. If the person reacts after the trade is executed, or information is made public, and profits from being the fastest to react, this is considered arbitrage and is legal and encouraged because it helps markets integrate new information into prices quickly.

### 2.2 Literature on Traditional Front-running

Front-running originates on the Chicago Board Options Exchange (*CBoE*) [41]. The Securities Exchange Commission *(SEC)* in 1977 defined it as: "The practice of effecting an options transaction based upon non-public information regarding an impending block transaction[1] in the underlying stock, in order to obtain a profit when the options market adjusts to the price at which the block trades. [2]" Self-regulating exchanges (*e.g., CBoE*) and the *SEC* spent the ensuing years planning how to detect and outlaw front-running practices [41]. The *SEC* stated: "It seems evident that such behaviour 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.[2]" The *CBoE* tried to educate their

[1] A block in the stock market is a large number of shares, 10 000 or more, to sell which will heavily change the price.

[2] 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).

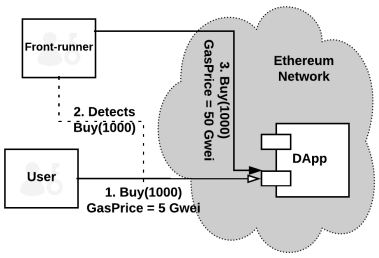


**Fig. 1:** The front-runner upon spotting the profitable transaction *Buy(1000)* sends his own transaction with higher gas price to bribe the miners to prioritize his transaction over initial transaction.

members on existing rules, however, differences in opinion regarding the unfairness of front-running activities, insufficient exchange rules and lack of a precise definition in this area resulted in no action [2] until the SEC began the regulation. We refer the reader interested in further details on this early regulatory history to Markham [41]. The first front-running policies applied only to certain option markets. In 2002, the rule was expanded to cover all security futures [3]. In 2012, it was expanded further with the new amendment, FINRA Rule 5270, to cover trading in options, derivatives, or other financial instruments overlying a security with only a few exceptions [6,5]. Similar issues have been seen with domain names [4,25] as well.

### 2.3 Background on Blockchain Front-running

Blockchain technology (introduced via Bitcoin in 2008 [48]) strives to disintermediate central parties that participate in a transaction. However, blockchains also introduce new participants in the process of relaying and finalizing transactions. Miners are in the best position to conduct these attacks as they hold fine-grained control over the exact set of transactions that will execute and in what order and can mix in their own (late) transactions without broadcasting them. Miners do however have to commit to what their own transactions will be before beginning the proof of work required to solve a block.

Any user monitoring network transactions (*e.g.,* running a full node) can see unconfirmed transactions. On the Ethereum blockchain, users have to pay for the computations in a small amount of Ether called gas [1]. The price that users pay for transactions, gasPrice, can increase or decrease how quickly miners will execute them and include them within the blocks they mine. A profit-motivated miner who sees identical transactions with different transaction fees will prioritize the transaction that pays a higher gas price due to limited space in the blocks. This has been called a gas auction [32]. Therefore, any regular user who runs a full-node Ethereum client can front-run pending transactions by sending adaptive transactions with a higher gas price (see Figure 1).

Finally, well-positioned relaying nodes on the network (or part of the broader internet backbone) can attempt to influence how transactions are propagated through the network, which can influence the order miners receive transactions, or if they receive them at all [30,40].

### 2.4 Literature on Blockchain Front-running

Given the purpose of this entire paper is systemizing the existing literature, we do not re-enumerate the literature here. However, we note two points. First, we are not aware of any other systematic study of this issue. Second, front-running is related to two well-studied concepts: double-spending and rushing adversaries [38].

Double-spending attacks in Bitcoin are related to front-running [11,36]. In this attack, a user broadcasts a transaction and is able to obtain some off-blockchain good or service before the transaction has actually been (fully) confirmed. The user can then broadcast a competing transaction that sends the same unspent coins to herself, perhaps using higher transaction fees, arrangements with miners or artifacts of the network topology to have the second transaction confirmed instead of the first. This can be considered a form of self-front-running. In the cryptographic literature, front-running attacks are modeled by allowing a so called 'rushing' adversary to interact with the protocol [12]. In particular, ideal functionalities of blockchains (such as those used in simulation-based proofs) need to capture this adversarial capability, assuming the real blockchain does not address front-running. See *e.g.,* Bitcoin backbone [29] and Hawk [38].

## 3 A Taxonomy of Front-running Attacks

As we will illustrate with examples through-out the paper, front-running attacks can often be reduced to one of a few basic templates. We emphasize what the adversary is trying to accomplish (without worrying about how) and we distinguish three cases: displacement, insertion, and suppression attacks. In all three cases, Alice is trying to invoke a function on a contract that is in a particular state, and Mallory will try to invoke her own function call on the same contract in the same state before Alice.

In the first type of attack, a *displacement attack*, it is not important to the adversary for Alice's function call to run after Mallory runs her function. Alice's can be orphaned or run with no meaningful effect. Examples of displacement include: Alice trying to register a domain name and Mallory registering it first [35]; Alice trying to submit a bug to receive a bounty and Mallory stealing it and submitting it first [16]; and Alice trying to submit a bid in an auction and Mallory copying it. In some cases, Alice's function and Mallory's function are different: *e.g.,* Alice trying to cancel an offer and Mallory fulfilling the offer first. We call this *asymmetric displacement*. Finally, in some cases, Mallory is trying to run a large set of functions: for example Alice and others are trying to buy a limited set of shares offered by a firm on a blockchain. We call this *bulk displacement*.

In an *insertion attack*, after Mallory runs her function, the state of the contract is changed and she needs Alice's original function to run on this modified state. For example, if Alice places a purchase order on a blockchain asset at a higher price than the best offer, Mallory will insert two transactions: she will purchase at the best offer price and then offer the same asset for sale at Alice's

| DApp Category | Names | Rank |
|---|---|---|
| Exchanges | IDEX | 1 |
| | **ForkDelta, EtherDelta** | 2 |
| | **Bancor** | 7 |
| | The Token Store | 13 |
| | LocalEthereum | 14 |
| | Kyber | 22 |
| | **0x Protocol** | 23 |
| Crypto-Collectible Games (ERC-721 [26]) | **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 |
| | PoWH 3D | 8 |
| | FomoWar | 10 |
| | FairDapp | 11 |
| | Zethr | 17 |
| | dice2.win | 20 |
| | 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. The DApps that are in bold are discussed in this paper.

slightly higher purchase price. If Alice's transaction is then run after, Mallory will profit on the price difference without having to hold the asset.

In a *suppression attack*, after Mallory runs her function, she tries to delay Alice from running her function. After the delay, she is invariant to whether Alice's function runs or not. We only observe this attack pattern in one DApp and the details are quite specific to it, so we defer discussion until Section 4.3.

## 4 Cases of Front-running in DApps

To find example DApps to study, we used the top 25 DApps based on recent user activity from `DAppradar.com` in September 2018.[3] User activity is admittedly an imperfect metric for finding the 'most significant' DApps: significant DApps might be lower volume overall or for extended periods of time (*e.g.,* ICOs, which we remedy by studying independently in Section 5). However, user activity is

[3] List of decentralized applications `https://DAppradar.com/DApps`

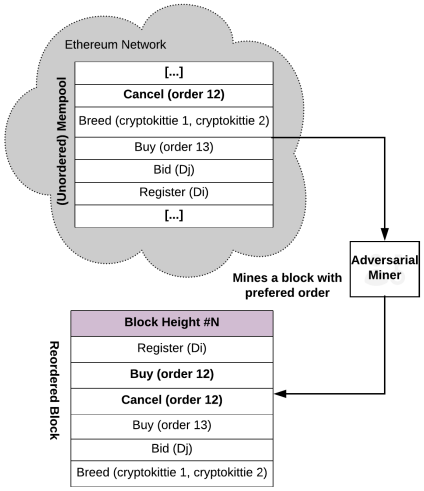


**Fig. 2:** The adversarial miner monitors the Ethereum mempool for decentralized exchange transactions. Upon spotting a profitable cancellation transaction, he puts his buy order prior to the cancel transaction in the block he mines. Doing so, the miner can profit from the underlying trade and also get the gas included in the cancel transaction.



an objective criteria, data on it is available, and the list captures our intuition about which DApps are significant. It suffices for a first study in this area, and is preferable over an ad hoc approach. Using the dataset, we categorized the top 25 applications into 4 principal use cases. The details are given in Table 1.

### 4.1 Markets and Exchanges

The first category of DApp in Table 1 are financial exchanges for trading ether and Ethereum-based tokens. Exchanges such as EtherDelta[4], purport to implement a decentralized exchange, however, their order books are stored on a central server they control and shown to their users with a website interface. Central exchanges can front-run orders in the traditional sense, as well as re-order or block orders on their servers. 0xProtocol [65] uses *Relayers* which act as the order book holders and could front-run the orders they relay.

As seen in traditional financial markets, one method to manipulate the spot price of an asset, is to flood the market with orders and cancel them when there are filling orders ("taker's griefing" [7]). Placing an order in a partially centralized exchange is free, but to prevent taker's griefing attacks, the user needs to send an Ethereum transaction to cancel each of his orders. Cancelling orders is most important when prices change faster than order execution. In this case, when an adversarial actor sees a pending cancellation transaction, he sends a fill order transaction with higher gasPrice to get in front of the cancellation order and take the order before it is canceled (this is known as *cancellation grief*). This attack follows the (asymmetric) insertion template and is illustrated in Figure 2.

Designing truly decentralized exchanges, where the order book is implemented directly on a public blockchain, is being pursued by a number of projects [24].

[4] Also known as ForkDelta for the user interface: `https://forkdelta.app/`

These designs are generally vulnerable to front-running attacks following a displacement or insertion template. For example, a front-running full node or miner might gauge the demand for trades at a given price by the number of pending orders, and try to displace them at the same price assuming the demand is the result of the accurate new information about the asset. Alternatively, the front-runner might observe a large market order (*i.e.,* it will execute at any price). The adversary can try to insert a pair of limit orders that will bid near the best offer price and offer at a higher price. If the pair executes ahead of the market order, the front-runner profits by scalping the price of the shares. Finally, if adversary has pre-existing offers likely to be reached by the market order, she could insert cancelations and new offers at a higher price.

Bancor is an exchange DApp that allows users to exchange their tokens without any counter-party risk. The protocol aims to solve the cryptocurrency liquidity issue by introducing *Smart Tokens* [31]. Smart tokens are ERC20-compatible that can be bought or sold through a DApp-based dealer that is always available and implements a market scoring rule to manage its prices. Bancor provides continuous liquidity for digital assets without relying on brokers to match buyers with sellers. Implemented on the Ethereum blockchain, when 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 chain (unlike other existing decentralized exchanges), these transactions are all visible to the public for some time before being included within a block. This leaves Bancor vulnerable to the blockchain race condition attack as attackers are given enough time to front-run other transactions, in which they can gain favourable profits by buying before the order and fill the original order with slightly higher price [58]. Researchers have shown and implemented a proof of concept code to front-run Bancor as a non-miner user [13].

### 4.2 Crypto-Collectibles Games

The second category of DApp in Table 1 is crypto-collectables. Consider Cryptokitties [9], the most active DApp in this category and third most active overall. Each kitty (see Appendix A) is a cartoon kitten with a set of unique features to distinguish it from other cryptokitties, some features are rarer and harder to obtain. They can be bought, sold, or bred with other cryptokitties. At the Ethereum level, the kitty is a token implemented with *ERC-721: Non-Fungible Token Standard* [26]. Kitties are generally bought and sold on-chain through auction smart contracts. See Sections 4.1 and 4.4 for more details on auction-based front-running attacks.

Specific to Cryptokitties protocol, they can breed and give birth. When cryptokitties breed, the smart contract sets from which future block the pregnancy of the cat can be completed. Anyone can complete the pregnancy by calling `giveBirth()` after the birthing block and they will receive a reward in ether[5].

[5] As there are no automated function calls in Ethereum, this incentive model –known as *Action Callback* [52]– is used to encourage users to call these functions.

Even though front-running these calls would not affect the protocol workflow, but this displacement attack could result in financial profit for front-runners [68,37].

### 4.3 Gambling

The third category of DApp in Table 1 is gambling services. While a large category of gambling games are based on random outcomes, DApps do not have unique access to an unpredictable data stream to harvest for randomness [51]. Any candidate source of randomness (such as block headers) is accessible to all DApp functions and can also be manipulated to an extent by miners.

Fomo3D is an example of a game style (known as Exit Scam[6]) not based on random outcomes, and it is the most active game on Ethereum in our sample. The aim of this game is to be the last person to have purchased a ticket when a timer goes to zero in a scenario where anyone can buy a ticket and each purchase increases the timer by 30 seconds. Many speculated such a game would never end but on August 22, 2018, the first round of the game ended with the winner collecting 10,469 Ether[7] equivalent to $2.1M USD at the time. Blockchain forensics indicate a sophisticated winning strategy to displace any new ticket purchases [10,57] that would reset the counter. The winner appears to have started by deploying many high gas consumption DApps unrelated to the game. When the timer of the game reached about 3 minutes, the winner bought 1 ticket and then sent multiple high gasPrice transactions to her own DApps. These transactions congested the network and bribed miners to prioritize them ahead of any new ticket purchases in Fomo3D. Recall this basic form of bribery is called a *Gas Auction*; See related work [43,14] for more sophisticated bribery contracts.

We classify this in the unique category of a suppression attack in our taxonomy (see Section 3). At first glance, it seemed like an extreme version of an asymmetric/bulk displacement attack on any new ticket purchase transactions. However the key difference is that the front-runner does not care at all about the execution of her transactions—if miners mined empty blocks for three minutes, that would also be acceptable. Thus, bulk displacement[8] is simply a means-to-an-end and not the actual end goal of the adversary.

### 4.4 Name Services

The final category in Table 1 is name services, which are primarily aimed at disintermediating central parties involved in web domain registration (*e.g.,* ICAAN and registrars) and resolution (*e.g.,* DNS). For simple name services (such as some academic work like Ghazal [47]), domains purchases are transactions and

[6] `https://exitscam.me/play`

[7] The first winner of Fomo3D, won 10,469 Ether `https://etherscan.io/tx/0xe08a519c03cb0aed0e04b33104112d65fa1d3a48cd3aeab65f047b2abce9d508`

[8] Also known as Block Stuffing Attack [59]

front-runners can displace other users attempting to register domains. This parallels front-running attacks seen in regular (non-blockchain) domain registration [4]. Ethereum Name Service (ENS) [34] is the most active naming service on Ethereum. Instead of allowing new `.eth` domain names to be purchased directly, they are put up for a sealed bid auction which seals the domain name in a bid, but not the bid amount. Most implementations use the more user friendly but less confidential method for starting and bidding on a domain name: `startAuctionsAndBid()`. This method leaks the hash of the domain and the initial bid amount in the auction. Original names can be guessed from the hashes (*e.g.,* rainbow tables, used in ENS Twitter bot[9]) or people can bid on domains even though they do not know what they are because of speculation on its value.

Users are allowed to bid for 3 days before the 2-day reveal phase begins (see 6.2), in which all bidders (winners and losers) must send a transaction to reveal their bids for a specific domain or sacrifice their bid amount . Also note that if two bidders bid the same price, the first to reveal wins it [23]. Using the leaked information, the domain squatter can win the auction with the same price of the original bidder by revealing it first. This is similar to front-running as it relies on inserting an action before the user, however we do not consider this specific action as front-running attack.

## 5 Cases of Front-running in ICOs

Initial coin offerings (ICOs) have changed how blockchain firms raise capital. More than 3000 ICOs have been held on Ethereum, and the market capitalization of these tokens appears to exceed $75B USD in the first half of 2018 [67]. At the DApp level, tokens are offered in short-term sales that see high transaction activity while the sale is on-going and then the activity tapers off to occasional owner transfers. When we collected the top 25 most active DApps on `DAppRadar.com`, no significant ICOs were being sold. The ICO category slips through our sampling method, but we identify it as a major category of DApp and study it here.

### 5.1 *Status.im* ICO

To deal with demand, ICOs cap sales in a variety of ways to mitigate front-running attacks. In June 2017, *Status.im* [8] started its ICO and reached the predefined cap within 3 hours, collecting close to 300,000 Ether. In order to prevent wealthy investors purchasing all the tokens and limit the amount of Ether deposited in each investment, they used a *fair* token distribution method called *Dynamic Ceiling* as an attempt to increase the opportunity for smaller investors. They implemented multiple caps (ceilings) 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 multi-signature wallet address [50].

[9] `https://twitter.com/ensbot`

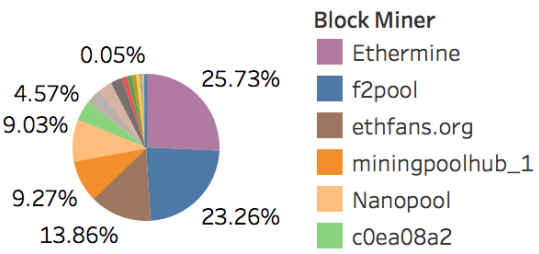


**Fig. 3:** The percentage of Ethereum blocks mined between block 3903900 and 3908029, this is the time frame in which Status.im ICO was running. This percentage roughly shows the hashing power ratio each miner had at that time.

During the time frame 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 a higher gas price to front-run other transactions, however, these transactions were failing due to the restriction in the ICO smart contract to reject transactions with higher than 50 *GWei* gas price (as a mitigation against front-running).

### 5.2 Data Collection and Analysis

According to the analysis we carried out, we discovered that the F2Pool—an Ethereum mining pool that had around 23% of the mining hash rate at the time (Figure 3)—sent 100 Ether to 30 new Ethereum addresses before the Status.im ICO started. When the ICO opened, F2Pool constructed 31 transactions to the ICO smart contract from their addresses, without broadcasting the transactions to the network[10]. They used their entire mining power to mine their own transactions and some other potentially failing high gas price transactions.

Ethereum's blockchain contains all transaction ever made on Ethereum. While the default client and online blockchain explorers offer some limited query capabilities, in order to analyze this case, we built our own database. Specifically, we used open source projects such as Go Ethereum implementation[11] for the full node, a python script for extracting, transforming and loading Ethereum blocks, named `ethereum-etl` [45] and Google BigQuery.[12] Using this software stack, we were able to isolate transactions within the Status.im ICO. We used 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 4, most of the top miners in the mentioned time frame, have mined almost the same number of failed and successful transactions which

[10] Note that we do not have an authoritative copy of the mempool over time, however, the probability of these transactions being broadcasted to the network and exclusively get mined by the same pool as the sender is low.
[11] Official Go implementation`https://github.com/ethereum/go-ethereum`.
[12] `https://cloud.google.com/bigquery/`
[13] `https://www.tableau.com/`
[14] `http://bit.ly/madibaFrontrunning`

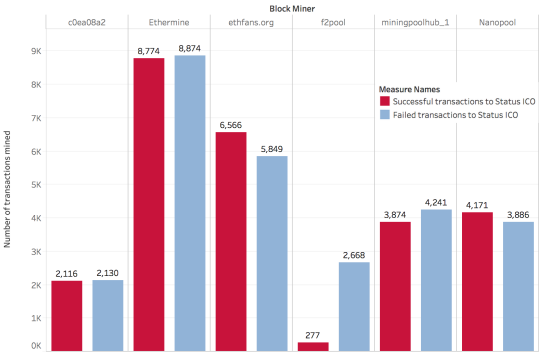


**Fig. 4:** This chart shows the miners behaviour on the time frame that Status.im ICO was running. It is clear that the number of successful transactions mined by F2Pool do not follow the random homogeneous pattern of the rest of the network.

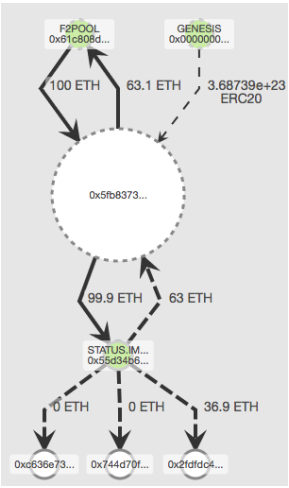


**Fig. 5:** Prior to *Status.im* ICO *F2Pool* deposited 100 Ether in multiple new Ethereum addresses. On the time of the ICO, transactions sent from these addresses to *Status* ICO smart contract were prioritized in their mining pool, resulting in purchasing *ERC20* tokens. This method was used to overcome the dynamic ceiling algorithm of the ICO smart contract. Later on they sent the refunded Ether back to their own address.[15]

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 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 rejected and threw an exception and by *"successful transaction"* we mean the transactions that went through and received tokens from the smart contract.

By tracing the transactions from these 30 addresses, we found explicit interference by F2Pool[16] in this scenario. As shown in Figure 5, 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. A few days after these funds were sent back to F2Pool main

[15] Graph was made using `Blockseer.com` blockchain explorer.

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

address and the tokens were aggregated later in one single address. 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 by *bulk displacement* to gain monetary profit.

## 6 Key Mitigations

As we studied front-running attacks on the blockchain, we also encountered a number of ways of preventing, detecting or mitigating front-running attacks. Instead of providing the details of exact solutions which will change over time, we extract the main principles or primitives that address the attack. A particular system may implement more than one in a layered mitigation approach.

We classify the mitigations into three main categories. In the first category, the blockchain removes the miner's ability to arbitrarily order transactions and tries to enforce some ordering, or queue, for the transactions. In the second category, cryptographic techniques are used to limit the visibility of transactions, giving the potential front-running less information to base their strategy on. In the final category, DApps are designed from the bottom-up to remove the importance of transaction ordering or time in their operations. We also note that for DApps that are legally well-formed (*e.g.,* with identified parties and a clear jurisdiction), front-running attacks can violate laws, which is its own deterrent. We do not discuss this further here; see Appendix C.



### 6.1 Transaction Sequencing

Ethereum miners store pending transactions in pools and draw from them when forming blocks. As the term 'pool' implies, there is no intrinsic order to how transactions are drawn and miners are free to sequence them arbitrarily.[17] The vanilla Go-Ethereum (geth) implementation prioritizes transactions based on their gas price and nonce [27]. Because no rule is enforced, miners can sequence transactions in advantageous ways. A number of proposals attempt to thwart this attack by enforcing a rule about how to sequence transactions.

First-in-first-out (FIFO) is generally not possible on a distributed network because transactions can reach different nodes in a different order. While the network could theoretically form a consensus based on locally observed FIFO, this would increase the rate of orphaned blocks, as well as adding complexity to the protocol. A trusted third party can be used to assign sequential numbers to transactions (and sign them), but this is contrary to blockchain's core innovation of distributed trust. None the less, some exchanges do centralize time-sensitive functionalities (*e.g., EtherDelta* and *0xProject*) in off-chain order books [65,64].

One alternative is to sequence transactions pseudorandomly. This can be seen in proposals like Canonical Transaction Ordering Rule (CTOR) by Bitcoin

[17] Sometimes the pool is called a 'queue.' It is important to note is a misnomer as queues enforce a first-in-first-out sequence.

Cash ABC [60] which adds transactions in lexicographical order according to their hash [61]. Note that Bitcoin does not have a front-running problem for standard transactions. While this could be used by Ethereum to make front-running statistically difficult, the protection is marginal at best and might even exacerbate attacks. A front-runner can construct multiple equivalent transactions, with slightly different values, until she finds a candidate that positions her transaction a desirable location in the resulting sequence. She broadcasts only this transaction and now miners that include her transaction will position it in front of transactions they heard about much earlier.

Finally, transactions themselves could enforce order. For example, they could specify the current state of the contract as the only state to execute on. This transaction chaining only prevents certain types of front-running; *i.e.,* it prevents insertion attacks but not displacement attacks (recall our taxonomy in Section 3). As transaction chaining only allows one state-changing transaction per state, at most one of a set of concurrent transactions can be confirmed; a drawback for active DApps.



### 6.2 Confidentiality

*Privacy-Preserving Blockchains.* All transaction details in Bitcoin are made public and participant identities are only lightly protected. A number of techniques increase confidentiality [19,42] and anonymity [46,49,56] for cryptocurrencies. A current research direction is extending these protections to DApps [66,55]. It is tempting to think that a confidential DApp would not permit front-running, as the front-runner would not know the details of the transaction she is front-running. However, there are some nuances here to explore.

A DApp includes the following components: (1) the code of the DApp, (2) the current state of the DApp, (3) the name of the function being invoked, (4) the parameters supplied to the function, (5) the address of the contract the function is being invoked on, and (6) the identity of the sender. Confidentiality applied to a DApp could mean different levels of protection for each of these. For front-running, function calls (3,4) are the most important, however, function calls could be inferred from state changes (2). Hawk [38] and Ekiden [21] are examples of (2,3,4)-confidentiality (with limitations we are glossing over).

The applicability of privacy-preserving blockchains needs to be evaluated on a case-by-case base. For example, one method used by traditional financial exchanges in dealing with front-running from high frequency traders is a dark market: essentially a (2,3,4)-confidential order book maintained by a trusted party. A DApp could disintermediate this trusted party. Users whose balances are affected by changes in the contract's state would need to be able to learn this information. Further, if the contract addresses are known (*i.e.,* no 5-confidentiality), front-runners can know about the traffic pattern of calls to contracts which could be sufficient grounds for attack; for example, if each asset on an exchange has its own market contract, this leaks trade volume information. As a contrasting example, consider again decentralized domain registration: hiding state changes

(2-confidentiality) defeats the entire purpose of the DApp, and protecting function calls is ineffective with a public state change since the state itself reveals the domain being registered.

*Commit/Reveal.* While confidentiality appears insufficient for solving domain name front-running alone, a hybrid approach of sequencing and confidentiality can be effective and is, in fact, an example of an older cryptographic trick known as commit/reveal. The essence of the approach is to protect the function call (*e.g.,* (3,4)- or (4)-confidentiality) until the function is enqueued in a sequence of functions to be executed. Once the sequence is established, the confidentiality is lifted and the function can only be executed in the place it was given (or, generally speaking, not at all).

Recall that a commitment scheme enables one to commit to a digital value (*e.g.,* a statement, transaction, data, *etc.*) while keeping it a secret (*hiding*), and then open it (and only it: *binding*) at a later time of the committer's choosing [15]. A common approach (conjectured to be hiding) is to submit the cryptographic hash of the value with a random nonce (for low entropy data) to a smart contract, and later reveal the original value and nonce which can be verified by the contract to correctly hash to the commitment (see Figure 8 in Appendix D).

An early application of this scheme to blockchain is Namecoin, a Bitcoin-forked DApp for name services [35]. In Namecoin, a user sends a commit transaction which registers a new hidden domain name, similar to a sealed bid. Once this first transaction is confirmed, a time delay begins. After the delay, a second transaction reveals the details of the requested domain. This prevents front-running if the reveal transaction is confirmed faster than an adversarial node or miner can redo the entire process.

Commit/reveal is a two-round protocol, and aborting after the first round (early aborts) could be an issue for this (along with most multi-round cryptographic protocols). For example, in a financial exchange where the number of other orders might be in a predictable interval, an adversary can spray the sequence (*i.e.,* a price-time priority queue) with multiple committed transactions and no intention of executing them all. She then only reveal the ones that result in an advantageous trade.[18] There are other ways of aborting; if payments are required but not collateralized, the aborting party can ensure that payment is not available for transfer. One mitigation to early aborts that blockchain is uniquely positioned to make is having users post a fidelity bond of a certain amount of cryptocurrency that can be automatically dispensed if they fail to fully execute committed transactions (this is used in multi-round blockchain voting [44]). Finally, we note that any multiple round protocol will have usability challenges: users must be aware that participating in the first round is not sufficient for completing their intention.

[18] This is analogous to behavior in traditional financial markets where high-frequency traders will make and cancel orders at many price points (flash orders or pinging). If they can cancel faster than someone can execute it—someone who has only seen the order and not the cancelation—then the victim reveals their price information.

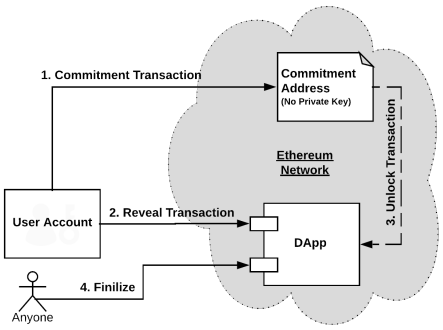


**Fig. 6:** Submarine Send [18]. User generates an *Unlock* transaction from which the commitment address is retrieved using ECDSA ECRecover. 1. by funding the *commitment address*,user is committed to the transaction. 2. User sends the *reveal transaction* to the DApp, revealing the nature of the commitment transaction. 3. She broadcasts the *unlock transaction* to unlock the funds in the commitment address. 4. After the *"Auction"* is over, anyone can call *Finalize* function to finalize the process.

*Enhanced Commit/Reveal.* Submarine Commitments [18,17] extend the confidentiality of the commit and reveal, so that the commitment transaction is identical to a transaction to a newly generated Ethereum address. They initially hide the contract address being invoked, providing (3,4,5)-confidentiality during the commit phase; and they ensure that if a revealed transaction sent funds, the funds were fully collateralized at commit time and are available to the receiving smart contract. See Figure 6.

### 6.3 Design Practices

The final main category of mitigation is to assume front-running is unpreventable and to thus responsively redesign the functionality of the DApp to remove any benefit from it. For example, when designing a decentralized exchange, one can use a call market design instead of a time-sensitive order book [22] to sidestep and disincentivize front-running. In a call market design, the arrival time of orders does not matter as they are executed in batches[19] . The call market solution pivots profitable gains that front-running miners stand to gain into fees that they collect [22], removing the financial incentive to front-run.

In the finance literature, Malinova and Park discuss front-running mitigations for blockchain-based trading platforms [39]. Instead of studying DApps, they develop an economic model where transactions, asset holdings, and traders' identities have greater transparency than in standard economic models—transparency they argue that could be accomplished by blockchain technology. However, in their model, they assume entities can interact directly over private channels to arrange trades. They define front-running in the context of private offers, where parties might adjust their position before accepting or countering a received offer. This model is quite different than the DApp-based model we study here.

Another example in the design of ERC20 standard [62], is on the allowance functionality. *approve()* function in the specification allows a second entity to be able to spend N tokens from the sender's balance. In order to change the

[19] Also known as batch auctions [63]

allowance, sender must send a transaction to set the new allowance value. Using the insertion attack, attacker could front-run the new allowance transaction and spend the old value before the new value is set [54,33], and then additionally spend the new amount at a later time. Solutions such as *decreaseApproval()/increaseApproval()* were added in updated implementations.

## 7 Concluding Remarks

Front-running is a pervasive issue in Ethereum DApps. While some DApp-level application logic could be built to mitigate these attacks, its ubiquity across different DApp categories suggests mitigations at the blockchain-level would perhaps be more effective. We highlight this as an important research area.

**Acknowledgements.** The authors thank the Autorité des Marchés Financiers (AMF) for sponsoring this research through the Education and Good Governance Fund (EGGF), as well as NSERC through a Discovery Grant.



## 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. 96th cong, 1st sess, report of the special study of the options markets to the securities and exchange comission, 1978.
3. Im-2110-3. front running policy. Financial Industry Regulatory Authority, 2002.
4. Ssac advisory on domain name front running. ICANN Advisory Committee, 10 2007. (Accessed on 08/15/2018).
5. 5270. front running of block transactions. Financial Industry Regulatory Authority, 2012.
6. Notice of filing of proposed rule change to adopt finra rule 5270 (front running of block transactions) in the consolidated finra rulebook. SECURITIES AND EXCHANGE COMMISSION, 2012.
7. Security review of 0x smart contracts. ConsenSys-Diligence, 2017.
8. The status network, a strategy towards mass adoption of ethereum. Status Team, 2017. Accessed: 2018-06-10.
9. Cryptokitties. Cryptokitties team, 2018. Accessed: 2018-08-31.
10. Anonymous. How the first winner of fomo3d won the jackpot? `https://winnerfomo3d.home.blog/`, 2018. Accessed: 2018-09-09.
11. 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.
12. D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Workshop on the Theory and Application of of Cryptographic Techniques*, pages 307–323. Springer, 1992.
13. 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).

14. J. Bonneau, E. W. Felten, S. Goldfeder, J. A. Kroll, and A. Narayanan. Why buy when you can rent? bribery attacks on bitcoin consensus. 2016.
15. G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.
16. 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.
17. L. Breidenbach, P. Daian, A. Juels, and F. Tramer. To sink frontrunners, send in the submarines. `http://hackingdistributed.com/2017/08/28/submarine-sends/`, 2017. Accessed: 2018-08-28.
18. L. Breidenbach, T. Kell, S. Gosselin, and S. Eskandari. Libsubmarine: Defeat frontrunning on ethereum. `https://libsubmarine.org/`, 2018. Accessed: 2018-12-07.
19. B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, volume 00, pages 319–338.
20. S. Buti, B. Rindi, and I. M. Werner. Diving into dark pools. 2011.
21. R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. *arXiv preprint arXiv:1804.05141*, 2018.
22. J. Clark, J. Bonneau, E. W. Felten, J. A. Kroll, A. Miller, and A. Narayanan. On decentralizing prediction markets and order books. In *Workshop on the Economics of Information Security, State College, Pennsylvania*, 2014.
23. E. Discussion. Handling frontrunning in the permanent registrar. 2018.
24. distribuyed. A comprehensive list of decentralized exchanges (dex) of cryptocurrencies, tokens, derivatives and futures, and their protocols. `https://distribuyed.github.io/index/`, 2018. Accessed: 2018-09-24.
25. B. Edelman. Front-running study: Testing report, 2009.
26. W. Entriken, D. Shirley, J. Evans, and N. Sachs. Erc-721 non-fungible token standard. `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-721.md`, 2018. Accessed: 2018-08-31.
27. Ethereum. worker.go - commitnewwork(). 2018. Accessed: 2018-12-07.
28. FinancialTimes. Barclays trader charged with front-running by us authorities. 2018.
29. J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015.
30. E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoins peer-to-peer network. In *USENIX Security*, pages 129–144, Washington, D.C., 2015. USENIX Association.
31. E. Hertzog, G. Benartzi, and G. Benartzi. Bancor protocol. 2017.
32. initc3.org. Frontrun me. *URL: http://frontrun.me/*, 2018.
33. G. Issue. Method 'decreaseapproval' in unsafe. 2017.
34. N. Johnson. Ethereum domain name service - specification. 2016.
35. 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.
36. 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.
37. M. B. Koch. Exploring cryptokitties - part 2: The cryptomidwives. 2018.

38. 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.
39. K. Malinova and A. Park. Market design with blockchain technology. 2017.
40. Y. Marcus, E. Heilman, and S. Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. Cryptology ePrint Archive, Report 2018/236, 2018. `https://eprint.iacr.org/2018/236`.
41. J. W. Markham. Front-running-insider trading under the commodity exchange act. *Cath. UL Rev.*, 38:69, 1988.
42. G. Maxwell. Confidential transactions. *URL: https://people. xiph. org/˜ greg/confidential_values. txt (Accessed 09/05/2016)*, 2015.
43. P. McCorry, A. Hicks, and S. Meiklejohn. Smart contracts for bribing miners. *IACR Cryptology ePrint Archive*, 2018:581, 2018.
44. P. McCorry, S. F. Shahandashti, and F. Hao. A smart contract for boardroom voting with maximum voter privacy. In *International Conference on Financial Cryptography and Data Security*, pages 357–375. Springer, 2017.
45. E. Medvedev. Python scripts for etl (extract, transform and load) jobs for ethereum blocks. `https://github.com/medvedev1088/ethereum-etl`, 2018.
46. I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.
47. S. Moosavi and J. Clark. Ghazal: toward truly authoritative web certificates using ethereum.
48. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
49. S. Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. `https://eprint.iacr.org/2015/1098`.
50. 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.
51. C. Pierrot and B. Wesolowski. Malleability of the blockchain's entropy. *Cryptography and Communications*, 10(1):211–233, 2018.
52. E. Piqueras. Generalized ethereum frontrunners, an implementation and a cheat. 2019.
53. R. Radner and A. Schotter. The sealed-bid mechanism: An experimental study. *Journal of Economic Theory*, 48(1):179–220, 1989.
54. R. Rahimian. Multiple withdrawal attack. 2018.
55. C. Reitwiessner. An update on integrating zcash on ethereum (zoe). *URL: https://blog.ethereum.org/2017/01/19/update-integrating-zcash-ethereum/*, 2017.
56. 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 (SP)*, pages 459–474. IEEE, 2014.
57. SECBIT. How the winner got fomo3d prize—a detailed explanation. `https://medium.com/coinmonks/how-the-winner-got-fomo3d-prize-a-detailed-explanation-b30a69b7813f`, 2018. Accessed: 2018-12-09.
58. E. G. Sirer and P. Daian. Bancor is flawed. `http://hackingdistributed.com/2017/06/19/bancor-is-flawed/`, 2017. (Accessed on 06/14/2018).
59. O. Solmaz. The anatomy of a block stuffing attack. *URL: https://osolmaz.com/2018/10/18/anatomy-block-stuffing/*, 2018.
60. R. Ver and J. Wu. Bitcoin cash planned network upgrade is complete. 2018. Accessed: 2018-12-07.

61. J. Vermorel, A. Séchet, S. Chancellor, and T. van der Wansem. Canonical transaction ordering for bitcoin. 2018. Accessed: 2018-12-07.
62. F. Vogelsteller and V. Buterin. Erc-20 token standard. `https://github.com/ethereum/EIPs/blob/master/EIPS/eip-20.md`, 2015. Accessed: 2018-08-31.
63. T. Walther. Multi-token batch auctions with uniform clearing prices. 2018.
64. 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).
65. W. Warren and A. Bandeali. 0x: An open protocol for decentralized exchange on the ethereum blockchain. *URl: https://github. com/0xProject/whitepaper*, 2017.
66. D. Z. J. Williamson. The aztec protocol. *URL: https://github.com/AztecProtocol/AZTEC/*, 2018.
67. 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.
68. Y. Zhou, D. Kumar, S. Bakshi, J. Mason, A. Miller, and M. Bailey. Erays: Reverse engineering ethereums opaque smart contracts. In *USENIX Security*, 2018.
69. H. Zhu. Do dark pools harm price discovery? *The Review of Financial Studies*, 27(3):747–789, 2014.

# 8 Appendix

**Note to Reviewers:** If accepted, the camera-ready version will link to a full version of the paper on eprint that will include the information in this appendix. We include it here for reference.

## A CryptoKitty

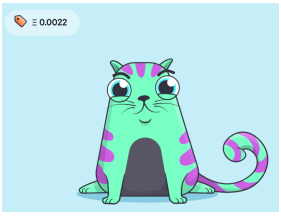


**Fig. 7:** Cryptokitty Number 842912

Figure 7 shows an example of a Cryptokitty.

## B LocalEthereum

```
    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.

Code 1.1 shows a mitigation technique employed by LocalEthereum.

## C 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 [28]. As mentioned in section 2.2, defining front-running and 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 lawsuits against firms who behaved in an unethical way. Other methods such as dark pools [69,20] and sealed bids [53] were discussed and implemented in a 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 who are front-running could be anonymous and the fear of lawsuits would not apply.

## D Commit-and-Reveal



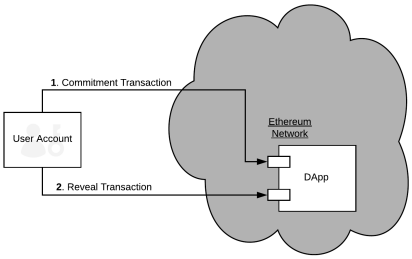


**Fig. 8:** Commit and Reveal. User sends a commitment transaction with the hash of the data, After the commitment period is over, user sends her reveal transaction to the DApp revealing the information that matches the commitment.

Figure 8 illustrates the commit/reveal approach.