SoK: Transparent Dishonesty. Front-running attacks on Blockchain.

Shayan Eskandari, Seyedehmahsa Moosavi, and Jeremy Clark

Gina Cody School of Engineering and Computer Science
Concordia University

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 traditional financial instrument markets since the 1970's. With the advent of the blockchain technology, front-running resurfaced in new forms we explore here, due to 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 decentral applications (DApps) deployed on Ethereum. Additionally, we carry out a detailed analysis of Status.im initial coin offering (ICO) and show evidence of abnormal miner's behaviour indicative of front-running token purchases. Developers don't have the mindset to design DApps with front-running in mind. This is an attempt to bring forward the subject and make them more aware of these type of attacks. Finally, we introduce a framework for categorizing proposed solutions to front-running and examine the applicability to the identified issues.

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 are 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 proposed solutions, with series of case studies of DApps deployed on Ethereum (a popular blockchain supporting DApps) from each category of the top 25 most active, based on user activity.[1] We do case studies on decentralized exchanges

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

(*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), which by happenstance, did not appear in the top 25 on our sample day. After a few hacking incidents of high valued smart contracts [41], 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 latecomers on the open markets [48,31]. ICOs started to experiment with different *fair* capping methods, such as reverse dutch auction and dynamic ceilings [26]. We show empirical evidence of a miner purchasing tokens ahead of other users in Status.im. Finally, proposals to eliminate or mitigate front-running from DApps are scattered across forums, proposed standards (called EIPs in Ethereum) and academic papers; we systemize them in the last section.

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. In traditional stock exchanges, floor traders might overhear a broker's negotiation with her client over a large purchase, and literally race the broker to buy first, potentially profiting when the large sale temporarily reduces the supply of the stock. A malicious broker might also 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 unethical and illegal in jurisdictions with mature 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*) [33]. The Securities Exchange Commission *(SEC)* in 1977 defines it as: "The practice of effecting an options transaction based upon non-public information regarding

an impending block transaction[2] 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 [33]. 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.[3]" The *CBoE* tried to educate their 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 regulation. We refer the reader interested in further details on this early regulatory history to Markham [33]. 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].

2.3 Background on Blockchain Front-running

In one sense, blockchain technology (which was introduced via Bitcoin in 2008 [38]) strives to disintermediate certain central parties that participate in a transaction. However, blockchains also introduce new participants in the relaying and finalization (*i.e.,* mining) of transactions that can act as front-runners. Any user monitoring blockchain network transactions (*e.g.,* running a full node) can see an unconfirmed transactions and broadcast a reactionary transaction that might be confirmed ahead of the original transaction. For regular users to front-run others on the blockchain, they need to be well connected to other nodes on the network and listen to the network to monitor all transactions that are broadcast. 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. Once seeing two identical transactions with different transaction fees, a rational miner will prioritize the transaction that pays a higher gas price, due to limited space in the blocks. Therefore, any regular user who runs a full-node Ethereum client can front-run pending transactions by sending similar transaction with a higher gas price. Also, 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 in case of malicious reordering is known as transaction reordering attack.

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

[3] 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).

2.4 Literature on Blockchain Front-running

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 market information leakage [11]. They also propose a cryptographic approach, similar to commit and reveal (see Section 5.2) to prevent front-running. Malinova and Park discuss different design settings for financial markets and trading platforms on decentralized ledgers [32]. They argue that blockchain brings new options of transparency which affects traders behaviour, especially 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.* [15] 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* which extend commit and reveal; we discuss them further in Section 5.2.

Double-spending attacks in Bitcoin have also been studied [12,29] where 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. We think of this as a type of front-running.

In the cryptographic literature, front-running attacks are modelled by allowing a 'rushing' adversary to interact with the protocol. 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.,* [23] and Hawk [30].

3 Cases of Front-running in DApps

To find example DApps to study, we use the top 25 DApps based on recent user activity from `DAppradar.com` and sample it in the first week of September 2018[4]. 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 4). However, user activity is arguably the best objective criteria for which data is readily available, the list captures our intuition about which DApps are significant, and it is at least better than 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.

DApp Category	Names (Ranking)
Exchanges	IDEX (1) **ForkDelta, EtherDelta** (2) **Bancor** (7) The Token Store (13) LocalEthereum (14) Kyber (22) **0x Protocol** (23)
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) 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. We discuss the DApps that are in bold.

3.1 Markets and Exchanges

The first category of DApp in Table 1 is exchanges. Exchanges such as EtherDelta[5], 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 [47] uses *Relayers* which act as the order book holders and could front-run the orders they relay. One method of price manipulation on financial markets is to flood the market with orders and cancel them when there are filling orders –taker's griefing– [19], to prevent such attacks, the user needs to send a transaction to cancel each of his orders. The reason for canceling order could be the unprofitability of the order due to price change. 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 is illustrated in Figure 1.

Designing truly decentralized exchanges, where the order book is implemented directly on a public blockchain, is being pursued by a number of projects [21]. These designs generally face the following attack (illustrated in Figure 2). An adversary can monitor the network with a full node for pending *buy* (or *sell*) transactions which could increase (or *decrease*) the future price of the asset. A

[4] List of decentralized applications `https://DAppradar.com/DApps`
[5] Also known as ForkDelta for the UI `https://forkdelta.app/`

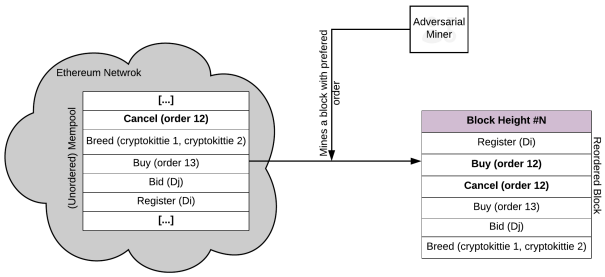

Fig. 1. The adversarial miner can monitor the Ethereum mempool for decentral exchange cancel orders and upon seeing the cancellation transaction, he puts his buy order prior to the cancel transaction. Doing so, the miner can profit from the underlying trade and also get the gas included in the cancel transaction.

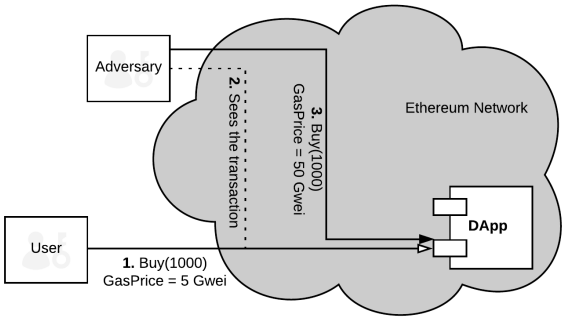

Fig. 2. The Adversary upon seeing the Buy order sends his own buy order with higher gas to front-run the first order.

front-runner can send a competing order with a higher gas price and hope to have her transactions mined ahead of the original pending transaction. The benefits of doing this vary. If the original bid is a large market order (*i.e.,* it will execute at any price), the adversary can front-run a pair of limit orders that will bid near the best offer price and offer at a higher price. If these execute ahead of the market order, the front-runner profits by scalping the price of the shares. If the adversary has pre-existing offers likely to be reached by the market order, she could front-run a cancelation and a new offer at a higher price. A simpler attack would be to gauge the demand for trades at a given price by the number of pending orders and to front-run at the same price in hopes that the market demand is the result of the accurate new market information. 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.

Bancor is another 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* [24]. 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 [42]. Researchers have shown and implemented a proof of concept code to front-run Bancor as a non-miner user [13].

3.2 Crypto-Collectibles Games

The second category of DApp in Table 1 is crypto-collectables. Consider Cryptokitties[6], 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* [22]. The market cap of cryptokitties peaked at more than 6 million dollars in the first few months of their launch, however, it has declined since. ERC-721 are similar enough to ERC-20 tokens that they can be listed on exchanges that are ERC-20 compatible. To buy a kitty, the user sends the following bid transaction: `bid(uint256 _tokenId)`. It contains the kitty's ID which can be found either on their website or on the ERC721 token smart contract. This is similar to open auctions and markets on section 3.1, bid value and the object bidding on is visible to the network and any user could easily front-run by sending the same transaction with higher gasPrice to replace the initial bidder. The front-runner can see there is an interest in a specific kitty and by buying it and auctioning it at a higher price can profit from the transaction.

3.3 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

[6] Cryptokitties website `https://www.cryptokitties.co/`

unique access to an unpredictable data stream to harvest for randomness. Any candidate (such as block headers) source of randomness 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[7]) 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[8] equivalent to $2.1M USD at the time. Blockchain forensics indicate a sophisticated winning strategy to front-run new ticket purchases [10] that would reset the counter. The winner appears to have started by deploying some 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.

3.4 Name Services.

The final category in Table 1 is name services, which are primarily aimed at displacing 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 [37]), domains purchases are transactions and network monitors can front-run other users attempting to register domains. This parallels front-running attacks seen in regular (non-blockchain) domain registration [4]. Ethereum Name Service (ENS)[9] 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 the bids, but not the bid amounts. The most common way of getting a domain is to call `startAuctionsAndBid()` in ENS auction smart contract, which leaks the hash of the domain name and the initial bid amount in the auction. Users are allowed to bid for 48 hours before the 48-hour reveal phase begins, in which bidders must send a transaction to reveal their bids for a specific domain in the auction. It is possible for a user to front-run other bids with the same bid amount by revealing first. The auction model is similar to crypto-collectibles auctions in 3.2.

4 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 appear to exceed $75B USD in the first half of 2018 [48].

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

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

[9] `https://ens.domains/`

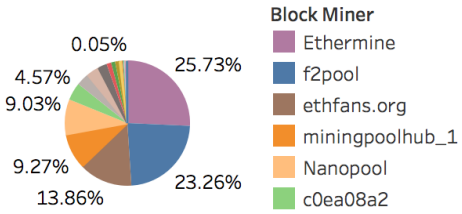

Fig. 3. The percentage of Ethereum blocks mined between block 3903900 and 3908029, this is the time frame in which Status.im ICO was running. This percentage roughly shows the hashing power ratio each miner had at that time.

At the DApp level, tokens are offered in short-term sales that see high transaction activity while the sale is on-going and then ICO activity tapers off to occasional owner transfers. When we collected the top 25 most active DApps on `DAppRadar.com`, no significant ICOs were in the initial sale stage. Despite the ICO category falls through our sampling method, we identify it as a major category of DApp and study it here.

4.1 *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 [39].

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 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 any transactions with higher than 50 *GWei* gas price (another mitigation against front-running).

4.2 Data Collection and Analysis

According to 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

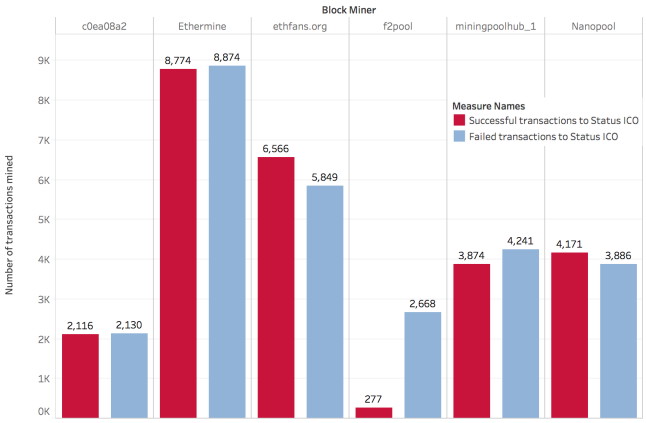

Fig. 4. This chart shows the miners behaviour on the time window that Status.im ICO was running. It is clear that the number of successful transactions mined by F2Pool do not follow the random homogeneous pattern of the rest of the network.

smart contract from these new address, without broadcasting the transactions to the network. 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[10] for the full node, a python script for extracting, transforming and loading Ethereum blocks, named `ethereum-etl` [35] and Google BigQuery.[11] Using this software stack, we were able to isolate transactions within the Status.im ICO. We used data analysis tool `Tableau`.[12] A copy of this dataset and the initial findings can be found in our Github repository.[13]

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

[10] Official Go implementation`https://github.com/ethereum/go-ethereum`
[11] `https://cloud.google.com/bigquery/`
[12] `https://www.tableau.com/`
[13] `github.com/` Removed for anonymity.

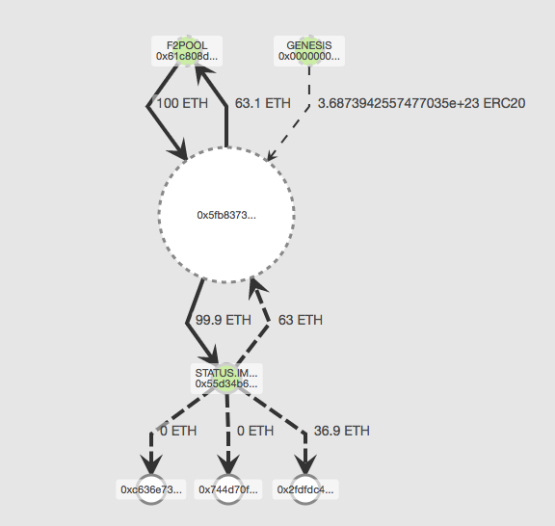

Fig. 5. F2Pool prior to Status.im ICO deposited 100 Ether in multiple new Ethereum addresses. On the time of the ICO they sent these deposits to Status ICO smart contract and prioritized mining of these transactions in their mining pool, this was to overcome the dynamic ceiling algorithm of the token sale smart contract. Later on they sent the refunded Ether back to their own address. [Graph made using Blockseer.com]

By tracing the transactions from these 30 addresses, we found explicit interference by the F2Pool[14] 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 back to these addresses, these funds were sent back to F2Pool main address a few days after and the tokens were aggregated later on 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 and gain monetary profit.

5 How to stop Front-running?

There are two main approaches to mitigating front-running on blockchains: one is to design a blockchain that is front-run resistant, and the other to design the application logic in a way that front-running is not profitable. This can be done in addition to any legal remedies that might be enforceable (See Appendix C). We note, but consider out-of-scope, approaches that introduce trusted par-

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

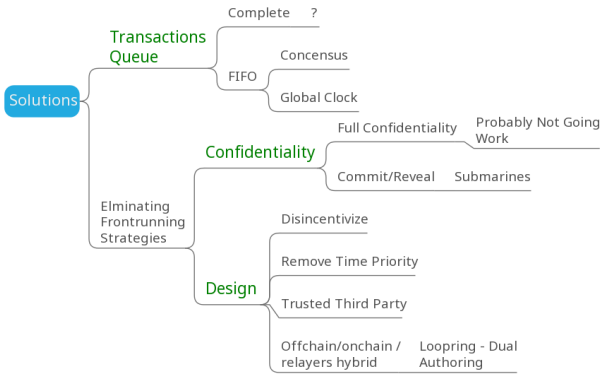

Fig. 6. *Solutions Mindmap*

ties to centralize time-sensitive functionalities (*e.g.,* exchanges *EtherDelta* and *0xProject* use off-chain order books [47,46]).

In this section, we describe the potential solutions using which, one can design and implement a decentralized application that is resistant to front-running. Note that, the existing technical difficulties make these solutions hard to achieve as it requires designers to test edge case scenarios.

5.1 Transaction Queue

Design Decision #1: Fixed Transaction Ordering. da Silva *et al.* [20] proposes an algorithm, known as Fixed Transaction Ordering and Admission (FTOA), in the consensus protocol that enforces the order of the transactions in the mined blocks, using logical timestamps in a Byzantine setting, to eliminate transaction reordering by miners. This is a theoretical solution and is not trivial to implement in the current Byzantine consensus protocols. An implementation of fixed transaction ordering is Canonical Transaction Ordering Rule (CTOR) introduced in November 2018 fork of Bitcoin cash, which indicates an order in which blocks are valid based on the order of the transactions in them; coinbase first, then ascending lexicographic order based on transaction hash [43]. This solution can potentially mitigate the re-ordering attack by miners, however it does not prevent it fully.

5.2 Confidentiality

Design Decision #2:Privacy-Preserving Blockchains. Privacy-preserving blockchains strive (to some extent [36,28]) to keep all details of transactions private, including participants and amounts. As an example, ZCash [25] uses two distinct types of addresses, transparent 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. While this solution works for simple transactions, building similar shields for DApps is subject to on-going research [30]. With the possibility of having private smart contracts in such a setting, it could be feasible to achieve a front-run resistance blockchain, however, the functionality of the smart contracts could be limited.

Design Decision #1: Commit / Reveal. A commitment scheme is a cryptographic primitive that enables one to commit to a value (*e.g.,* a statement, document, data, *etc.*) while keeping it a secret, and then open it (and only it) at a later time of the committer's choosing [14]. A common approach is to submit the hash value of the data to be committed, potentially with a random nonce (for predictable data), to a smart contract and later reveal the original data and nonce which can be verified to hash to the commitment. See Figure 9 in Appendix D.

An early use of this scheme is Namecoin, a Bitcoin-forked blockchain for name services [27]. In Namecoin, a user sends a commit transaction which commits to the name hash, similar to a sealed bid. Once the transaction is confirmed and the grace period is over, the user sends a reveal transaction revealing the bid and also the details of the requested domain [27]. Hence, the user is able to hide information from the adversarial parties in the system and prevent front-running attacks.

In the case of decentralized exchanges, users can use commit/reveal for bids, where commitments are placed in an ordered queue and executed in a later round after a reveal. This mitigation seems to require further research. One trader strategy is to flood orders at the commitment stage, but then only selectively and adaptively choose which to reveal. Collateralization is another challenge, where revealed bids may simply be intentions to execute but additional transactions are required to move the money or tokens; a transaction that can also be aborted early. One mitigation to early aborts is having users post a fidelity bond that can be automatically dispensed if they fail to fully execute committed transactions (as used in [34]). Commit/reveal is also seen in on-chain voting [7]. Using sealed votes, voters can assure their votes are hidden until a later date, however, their participation and weight of the vote are publicly known.

The participatory factor is another issue for the commitment scheme. For anyone watching the DApp address, there will be a direct transaction from the user to the DApp address, revealing the fact 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 the ENS commit-reveal process, if one forgets to reveal in the reveal phase, the committed funds are burnt and inaccessible thereafter. Also, this scheme means two transactions are required for a functionality which makes it more expensive.

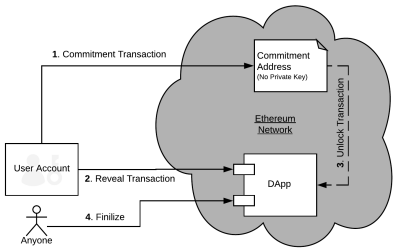

Fig. 7. Submarine Send [9]. User generates an *Unlock* transaction from which the commitment address is retrieved. By funding the *commitment address*, user is committed to the unlock transaction. After the commitment transaction, in the reveal phase, user sends the *reveal transaction* to the DApp and 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 #2: Submarine Commitments. Submarine Commitments [15,16] extend basic commit/reveal functionality with additional properties. In particular, they hide the DApp being invoked during the commit phase and also ensure that revealed transactions that send funds are fully collateralized. With submarine send, it is possible to hide sender, receiver, value, and data, so 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 could fill an entire paper, however in short, using the way Ethereum transactions are constructed and ECDSA ECRecover functionality, it is possible to generate one outgoing transaction from an address of which no private key exists. By constructing a transaction from that address to the DApp and funding that address we fulfill the commitment phase. By revealing the details and broadcasting the constructed transaction we reveal the commitment [9]. See Figure 7. However, this could be designed in other ways to suit the need of the DApp. Drawbacks of this scheme are similar to commit/reveal, with the addition of two more transactions to finalize a transaction, which adds cost and complexity to the user experience.

5.3 Design

Design Decision #3: Use-Case Specific Solutions. There is no perfect generic solution to this issue as of now, however, 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 not to rely on the time for the orders to be executed. For example when designing a decentralized exchange, one can use a *the call market* design instead of a

time-sensitive order book [18] to side-step and disincentivize front-running by miners. In a call market design, the arrival time of orders does not matter as they are executed in batches.

- *Disincentivizing Front-running Actors:* The call market solution also pivots profitable gains that front-running miners stand to gain into fees that they collect [18], removing the financial incentive to front-run. To the same end, Malinova [32] proposes a design for a decentralized exchange which uses economic models in their trading market design to make front-running expensive and more costly than profitable. They split big investments into smaller ones using intermediaries –liquidity demander– and liquidity providers to achieve this.
- *Trusted party:* It is possible to use a central trusted party, to mitigate the risks of front-running on the DApp design. As an example, LocalEthereum, an escrow-based exchange, facilitates the website as the matchmaking service for the exchange orders, and uses challenge random values from the website to be included in the signature of the transaction (See Code 1.1 in Appendix B), hence removing any possibility of replay attack or the possibility of any other address modifying key information in the order. This method prevents network observers to be able to make a new transaction with the defined values as the third party will be the final entity approving the transaction.

Design Decision #3: Dual Authoring. This scheme is proposed to solve the front-running issue within Loopring – a decentralized exchange protocol proposed by Wang *et al.* [45]. In the first version of this protocol, orders are grouped in *ring orders* once they are sent by the users. After the required signatures are provided by the *ring miner*, he sends the *submitRing transaction* to the Loopring Smart Contract (LPSC) for verification and settlement (see Figure **??**). While this transaction sits unconfirmed in the mempool, any front-runner can create a copy of this transaction with higher gasPrice and his address instead of the ring miner's address. Using the proposed *dual authoring* solution in the new version of Loopring, regeneration of the *submitRing transaction* is infeasible as the users have to sign the orders using their secondary private keys (auth-keys), which are only known to the ring miner and not other users (see Figure **??**).

6 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. Dev diary i: A walkthrough of plcr voting in solidity. adChain, 2017. Accessed: 2018-08-28.
8. The status network, a strategy towards mass adoption of ethereum. Status Team, 2017. Accessed: 2018-06-10.
9. Libsubmarine: Temporarily hide transactions on ethereum (cheaply!). LibSubmarine Team, 2018. Accessed: 2018-08-28.
10. Anonymous. How the first winner of fomo3d won the jackpot? `https://winnerfomo3d.home.blog/`, 2018. Accessed: 2018-09-09.
11. 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.
12. 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.
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. G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.
15. 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.
16. 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.
17. S. Buti, B. Rindi, and I. M. Werner. Diving into dark pools. 2011.
18. 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.
19. ConsenSys-Diligence. Security review of 0x smart contracts. *URL https://github.com/ConsenSys/0x-review/blob/master/report/3$_g$eneral$_f$indings.md*, 2017.
20. P. M. da Silva, M. Matos, and J. Barreto. Fixed transaction ordering and admission in blockchains.

21. 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.
22. 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.
23. J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015.
24. E. Hertzog, G. Benartzi, and G. Benartzi. Bancor protocol. 2017.
25. D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification. Technical report, Technical report, 2016–1.10. Zerocoin Electric Coin Company, 2016.
26. W. Kaal and M. Dell'Erba. Initial coin offerings: Emerging practices, risk factors, and red flags. 2017.
27. 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.
28. G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. An empirical analysis of anonymity in zcash. *arXiv preprint arXiv:1805.03180*, 2018.
29. 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.
30. 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.
31. J. Li and W. Mann. Initial coin offering and platform building. 2018.
32. K. Malinova and A. Park. Market design with blockchain technology. 2017.
33. J. W. Markham. Front-running-insider trading under the commodity exchange act. *Cath. UL Rev.*, 38:69, 1988.
34. 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.
35. 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.
36. A. Miller, M. Möser, K. Lee, and A. Narayanan. An empirical analysis of linkability in the monero blockchain. *arXiv preprint*, 1704, 2017.
37. S. Moosavi and J. Clark. Ghazal: toward truly authoritative web certificates using ethereum.
38. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
39. 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.
40. R. Radner and A. Schotter. The sealed-bid mechanism: An experimental study. *Journal of Economic Theory*, 48(1):179–220, 1989.
41. D. Siegel. Understanding the dao attack. *Web. http://www. coindesk. com/understanding-dao-hack-journalists*, 2016.
42. E. G. Sirer and P. Daian. Bancor is flawed. `http://hackingdistributed.com/2017/06/19/bancor-is-flawed/`, 2017. (Accessed on 06/14/2018).

43. J. Vermorel, A. Schet, S. Chancellor, and T. van der Wansem. Canonical transaction ordering for bitcoin. *URL https://blog.vermorel.com/pdf/canonical-tx-ordering-2018-06-12.pdf*, 2018.
44. D. Wang, J. Zhou, A. Wang, and M. Finestone. Lexisnexis. 2015. Accessed: 2018-09-12.
45. 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.
46. 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).
47. W. Warren and A. Bandeali. 0x: An open protocol for decentralized exchange on the ethereum blockchain. *URl: https://github. com/0xProject/whitepaper*, 2017.
48. 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.
49. H. Zhu. Do dark pools harm price discovery? *The Review of Financial Studies*, 27(3):747–789, 2014.

7 Appendix

A CryptoKitty

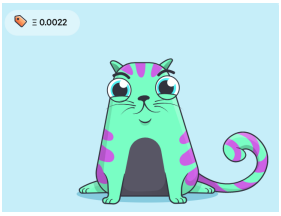

Fig. 8. Cryptokitty Number 842912

Figure 8 shows an example of a Cryptokitty.

B LocalEthereum

```
1   function createEscrow(bytes16 _tradeID, address _seller, address _buyer,
        uint256 _value, uint16 _fee,
2         uint32 _paymentWindowInSeconds, uint32 _expiry, uint8 _v, bytes32
              _r, bytes32 _s)
3       payable external {
4       bytes32 _tradeHash = keccak256(abi.encodePacked(_tradeID, _seller,
            _buyer, _value, _fee));
5   ...
```

```
6         // A signature (v, r and s) must come from localethereum to open an
              escrow
7         bytes32 _invitationHash = keccak256(abi.encodePacked(
8             _tradeHash,
9             _paymentWindowInSeconds,
10            _expiry
11        ));
```

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 [44]. 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 [49,17] and sealed bids [40] 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

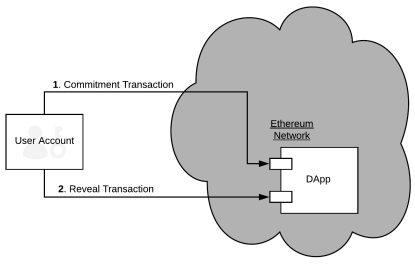

Fig. 9. Commit and Reveal. User sends a commitment transaction with the hash of the data, After the commitment period is over, user sends her reveal transaction to the DApp revealing the information that matches the commitment.

Figure 9 illustrates the commit/reveal approach.