The paper on blockchain front-running

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 blockchain's decentralized and transparent nature. In this paper, we consider instances of front-running across the top 25 decentral applications (DApps) deployed on Ethereum. Additionally, we carry out detailed analysis of Status.im initial coin offering (ICO) and show evidence of abnormal miner's behaviour indicative of front-running token purchases. 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 frontrunning 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. This paper is a series of case studies of DApps deployed on Ethereum (a popular blockchain supporting DApps) from each category of the top 25 most active.[1] We do case studies on decentralized exchanges (*e.g.,* Bancor), crypto-collectables (*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 [40], 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 [30,47]. ICOs started to experiment with different fair capping methods, such as reverse dutch auction and dynamic

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

ceilings [25]. We show empirical evidence of a miner purchasing tokens ahead of other users in Status.im. Finally, proposals to eliminate or mitigate frontrunning from DApps are scattered across forums, proposed standards (called EIPs in Ethereum) and academic papers; we systemize them 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. The problem can occur in 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 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 executes. 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 both 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

[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

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 [5, 6].

2.3 Background on Blockchain Front-running

In one sense, blockchain technology (which was introduced via Bitcoin in 2008 [37]) strives to disintermediate certain central parties that would 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 a unconfirmed transactions and broadcast a reactionary transaction that might be confirmed ahead of the original transaction ...

that is well-connected front-run other transactions in the network. For regular users to front-run others on the blockchain, they need to be well connected to other nodes on the network. Doing so, they are able to listen to the network and monitor all transactions that are broadcast. On the Ethereum blockchain, users have to pay for the computations in 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 per unit of gas, gasPrice, due to limited space in the blocks. Therefore, any regular users who run a full-node Ethereum client can front-run pending transactions by sending similar transaction with higher gas price. 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.

2.4 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 [12].They also propose a cryptographic approach,

[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).

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.* [17] 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 been studied [13, 28] 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 artefacts 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. Literature like the Bitcoin Backbone [] and Hawk [] capture this.

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 sampled it in 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.

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

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

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.

stored on a central server they control and shown on 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 [46] uses *Relayers* which act as the order book holders and are not prone to cheating. One main issue with the relay design is *cancellation grief*. To prevent denial of service attacks, the user needs to send a transaction to cancel her order. In this case, an adversarial actor many see a pending cancellation transaction, and send 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 illustrated in Figure 1.

Designing truly decentralized exchanges, where the orderbook is implemented directly on a public blockchain, is being pursued by a number of projects [cite github page]. 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 front-runner can send a competing orders 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 guage the demand for trades at a given

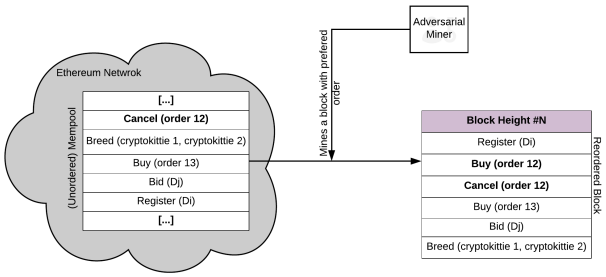

Fig. 1: The adversarial miner can monitor the Ethereum mempool for decentralized exchange cancel orders and upon seeing the cancellation transaction, he puts his buy order prior to the cancel transaction. Doing so, the miner can profit from the underlying trade and also get the gas included in the cancel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

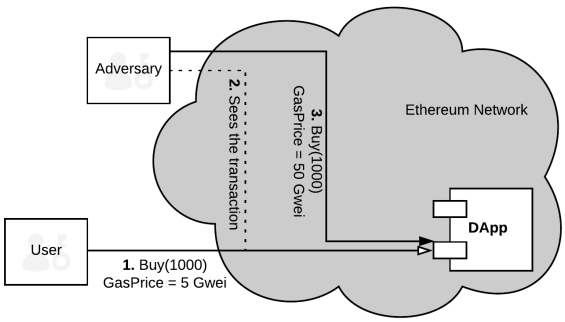

Fig. 2: The Adversary up on seeing the Buy order, sends his own buy order with higher gas to front-run the first order.

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 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 Ethereum-based application that allows users to exchange their tokens without any counter-party risk. This protocol aims to solve the cryptocurrency liquidity issue by introducing *Smart Tokens* [23]. Smart tokens are ERC20-compatible and can be bought or sold through a DApp-based dealer that is always available and implements a market scoring rule to manage its prices. Doing so, 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 Ethereum blockchain (unlike other existing decentralized exchanges), these transactions are all visible to the public for some time before being included within a block. This leaves this blockchain-based decentralized exchange vulnerable to the blockchain race condition attack as attackers are given enough time to front-run other transactions and gain favorable profits [41]. Researchers have shown and implemented a proof of concept code to front-run Bancor as a regular non-miner user [15].

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 cryptokitty (see Appendix A) is a cartoon kitten with a set of unique features to distinguish it from other cryptokitties, some features more rare 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 cryptokitty tokens peaked at more than 6 million dollars in the first few months their launch, however, has declined since. ERC-721 are similar enough to ERC-20 tokens that they can be listed on exchanges setup for ERC-20. To buy a cryptokittie, the user sends the following bid transaction: `bid(uint256 _tokenId)`. It contains the cryptokittty's ID which can be found either on their website or on the ERC721 token smart contract. This is similar to open auctions, bid value and the object bidding on is visible to the network and any user could be easily front-run by sending the same transaction with higher gasPrice to replace the initial bidder.

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 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

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

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

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

forensics indicate a sophisticated winning strategy to frontrun new ticket purchases [11] 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 [36]), 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. Although sealed auction method is used in ENS implementation, the normal workflow uses `startAuctionsAndBid()` with the deposit equal or higher than the bid, this information leakage could be used to front-run the transaction with the same bid amount or slightly higher to outbid the initial sealed bid. By doing so the front-runner invalidates the original bid and hence registers the domain name himself.

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 [47]. 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 tappers 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 will 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, resulting in close to 300,000 Ether of funds. In order to prevent few large investments to buy up all the tokens and limit the

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

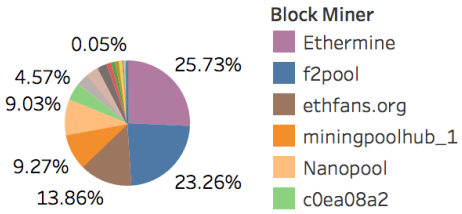

Fig. 3: The percentage of Ethereum blocks mined between block 3903900 and 3908029, this is the time frame in which Status.im ICO was running. This percentage roughly shows the hashing power ratio each miner had at that time.

amount of Ether deposited in an investment, they used a 'fair' token distribution method they called a *Dynamic Ceiling* as an attempt to increase the opportunity for smaller investors. They implemented multiple caps or 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 [38].

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 frontrunning).

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 token sale opened, F2Pool contsructed 31 transactions to the ICO token sale smart contract from the addresses these new address, without broadcasting these 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 querying, 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` [34] and Google BigQuery.[11] Using this software stack, we were

[10] Official Go implementation`https://github.com/ethereum/go-ethereum`
[11] `https://cloud.google.com/bigquery/`

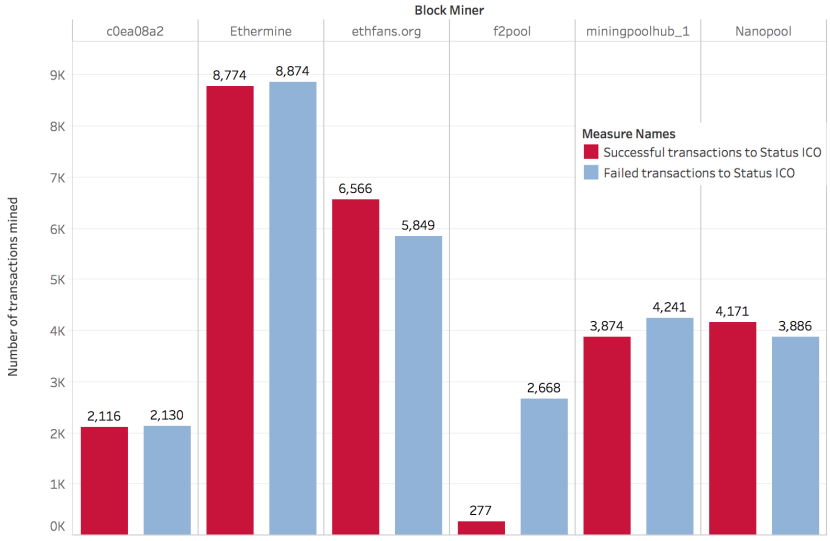

Fig. 4: This chart shows the miners behaviour on the time window that Status.im ICO was running. It is clear that the number of successful transactions mined by F2Pool do not follow the random homogeneous pattern of the rest of the network.

able to isolate transactions within the Status.im ICO sale. 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.

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. Although this incident does not involve

[12] `https://www.tableau.com/`
[13] `github.com/` Removed for anonymity.
[14] F2Pool Ethereum address was identified by their mining reward deposit address `https://etherscan.io/address/0x61c808d82a3ac53231750dadc13c777b59310bd9`

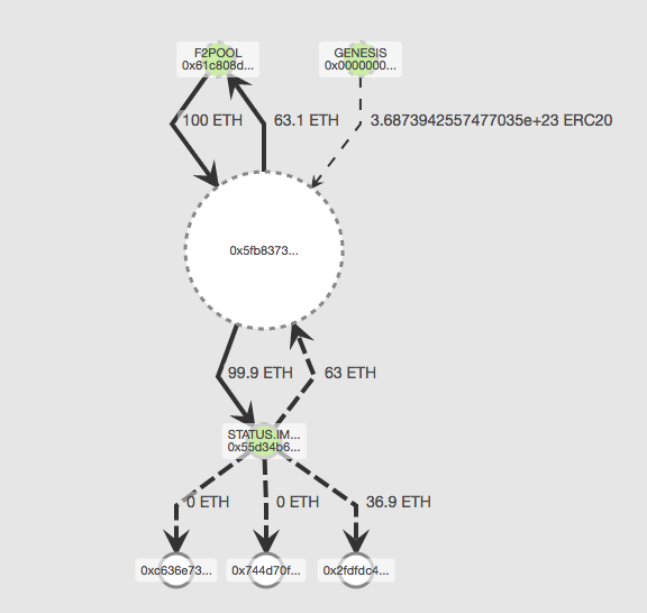

Fig. 5: F2Pool prior to Status.im ICO deposited 100 Ether in multiple new Ethereum addresses. On the time of the ICO they sent these deposits to Status ICO smart contract and prioritized mining of these transactions in their mining pool, this was to overcome the dynamic ceiling algorithm of the token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

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. We note, but consider out-of-scope, approaches that introduce trusted parties to centralize time-sensitive functionalities (*e.g.,* exchanges *EtherDelta* and *0xProject* use off-chain order books [45, 46]).

5.1 Front-run Resistant Blockchain

There are technical difficulties to achieve such solutions as there are unknown factors within such network designs and requires testing every corner and edge case scenarios. In this section, we describe the potential solutions using which,

one can design and implement a decentral application that is resistant to front-running.

Design Decision #1: Fixed Transaction Ordering
One possible solution to fix current blockchains in regards to transaction re-ordering by miners has been proposed by da Silva *et al.* [21]. This solution consists of an algorithm, known as Fixed Transaction Ordering and Admission (FTOA) algorithm, in the consensus protocol that enforces the order of the transactions in the mined blocks.

Design Decision #2:Privacy-Preserving Blockchain
Market participants are concerned about revealing information on their past trades, purchases, income, or liquidity needs. Privacy is a basic human right, that being said, there are limitations to the functionality of a system designed to preserve full privacy. However other than past transactions, real-time transactions would leak information about the sender or the intention of the order that could lead to front-running attacks. Privacy-preserving blockchains try to solve the initial issue by keeping all the details of the transactions private (in some extent [27, 35]). As an example, ZCash [24], uses two distinct type of addresses, transparent addresses and shielded addresses. Transparent addresses work similar to Bitcoin transactions, fully transparent about the sender and receiver addresses, the amount and included data. However shielded addresses are private and do not leak any information, this is great for privacy but not good for smart contracts capabilities. Smart contract functionality requires verification by the miners which require to know the input, functionality, and output of smart contracts to be able to determine if the transaction is valid. Albeit by using cryptographic primitives such as zero-knowledge proofs this issue could also be resolved [29].

Design Decision #3: Dual Authoring
Loopring is an open protocol proposed by Wang *et al.* to build decentralized exchanges [44]. This protocol operates a hybrid model with off-chain order management and communication while only the order matching takes place on the chain. In this protocol, the orders are grouped in *ring orders* immediately after they are sent by the users. Next, the required signatures are provided by a miner and owner of the order, see Figure 6. Eventually, the ring miner sends the *submitRing transaction* to the Loopring Smart Contract (LPSC) for verification and settlement. While the *submitRing transaction* sits unconfirmed in the mempool, any front-runner can create a copy of this transaction with his address instead of the ring miner's address and then sign the hash of the ring using his own private key. By paying a higher gasPrice, block miners choose the front-runner's transaction instead of the original *submitRing transaction*. To solve this issue, the new version of Loopring protocol uses the *dual-authoring* scheme. This solution sets two levels of authorization for orders - one for settlement, and one for

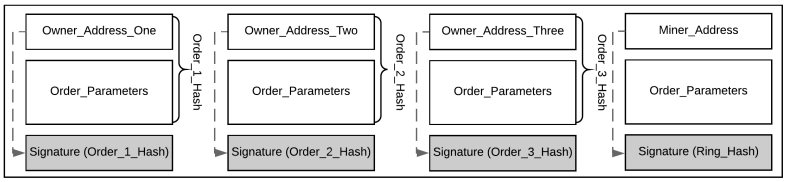

Fig. 6: *submitRing* Transaction in the old version of the Loopring protocol. Any user can regenerate this transaction by replacing the miner's address with his address and signing the hash of the ring using his private key.

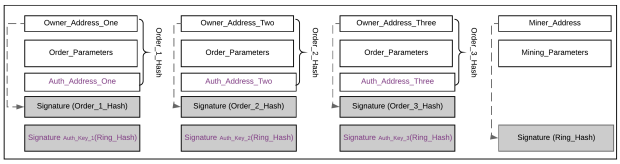

Fig. 7: *submitRing* Transaction in the new version of the Loopring protocol. Other users cannnot regenerate this transaction as they do not access auth-keys and hence are not able to create the signatures.

order matching. As it can be seen in Figure 7, each owner of an order generates a pair of public key (auth-address) and private key (auth-key), which are different from his address and its corresponding private key. These auth-keys are used to sign the hash of the ring and will further prevent anyone from regenerating the same *submitRing transaction*. The reason is that the auth-keys are only known to the miner of the ring as they are not part of the on-chain transaction. Doing so, it is assured that (i) the orders are not modified (because they are signed by the owner), (ii) no other miner can mine the ring (because the miner's address is signed by the initial miner), and no user can front-run the *submitRing transaction* (because other users do not have access to the auth-keys and hence are not able to generate new transactions with auth-keys signatures on them).

5.2 Application Design to Prevent Front-running

Another method to prevent front-running activities is to design the applications in such a way that there is no information leakage to the malicious parties to enable them to front-run other users. Followings discuss a few techniques which can be used by the DApp developers while designing the systems to mitigate front-running issues.

Design Decision #1: Commitment Scheme

The commitment scheme is a cryptographic primitive that enables one to commit to a value –binding– (*e.g.,* statement, document, data, *etc.*) while keeping it

a secret and reveal the committed value on a later time [16]. The commitment scheme is a robust method to prevent information leakage from the sensitive transactions. This could be done by simply submitting the hash value of the committed data to a smart contract and later on revealing the values linked to the committed hash, see Figure 10. In the case of decentral exchanges, the user can send a commit transaction which will be cryptic to network participants but will act as a placeholder in the queue for the user, after the transaction is mined, the user sends reveal transaction revealing the order details which will be executed in a fair order. This method is not perfect, depending on the DApp design, either the order is collateralized, that means the commitment transaction includes the funds required for fulfilling the order, which will leak some information about the order, or is not collateralized and opens up the possibility of user never revealing the commitment, which will be vulnerable to Denial-of-Service attacks. Another use of such a commitment scheme is in decentralized naming systems, such as Namecoin, Ethereum Name Service (ENS). In this case, 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 [26]. Using this scheme, one is able to hide information from the adversarial parties in the system and prevent them from front-running. A similar approach applies to on-chain voting as well [7], using sealed votes, voters can make sure their votes are hidden until a later date, however, their participation and weight of the vote are publicly known. Other than the collateralization issue for the commit and reveal scheme, another obvious issue is the participatory factor. For anyone watching the DApp address, there will be a direct transaction from the user to the DApp address, It will be obvious that a specific address has participated in the auction or the DApp using a commit and reveal scheme, but the details are hidden through the scheme. Both these issues, albeit hiding the details of the order and preventing direct front-running, leak information to other participants which could lead to more sophisticated front-running attacks depending on the application design. On the other note, these method has some drawbacks, the user experience is not smooth and one might forget to reveal the committed transaction. In 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 to run.

Cite Princeton paper on Namecoin for frontrunning mitigation using commit/reveal.

Design Decision #2: Submarine Commitments

Submarine Commitments [17, 18] are similar to the commitment schemes with some key differences such as hiding participatory factor and being fully collateralized. With submarine send, it is possible to hide sender, receiver, value, and data, the commitment transaction is identical to a transaction to a newly gen-

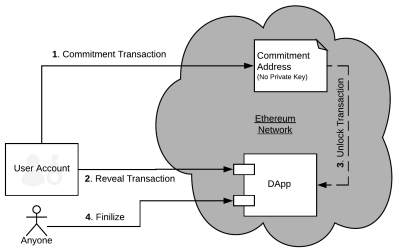

Fig. 8: Submarine Send [10]. User generates an *Unlock* transaction from which the commitment address is retrieved. By funding the *commitment address*, user is committed to the unlock transaction. After the commitment transaction, in the reveal phase, user sends the *reveal transaction* to the DApp 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.

erated address. Submarine sends could be a solution for sealed-bid auctions on the blockchain to hide the existence of the bids for other participants.

The details of how Submarine commitments work is outside the scope of this paper, however in 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 and by revealing the details and broadcasting the constructed transaction we reveal the commitment [10], see Figure 8. However, this could be designed in other ways to suit the need of the DApp. Drawbacks of this scheme are similar to 5.2, with an addition of two more transactions to finalize the functionality, which results in more cost and complicated user experience.

Design Decision #3: Logic 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 decentral exchange, one can use *the call market* design instead of a time-sensitive order book. In a call market design, the arrival time of the orders does not matter as they are executed in batches [20].

Disincentivizing Front-running Actors: This solution, which is economic rather than technical, is to disincentives the miners by paying them the fees. If the application is designed in a way that the fees of the orders are sent to the miner of the block, he would have less incentive to front-run the orders as he already gains enough financial benefit from acting fairly and executing the transactions on their correct order. For example, Malinova [32] proposes a design for a decentral

exchange which uses economic models in their trading market design to make front-running expensive and more costly than profitable.

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

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. Cryptokitties. Cryptokitties team, 2018. Accessed: 2018-08-31.
10. Libsubmarine: Temporarily hide transactions on ethereum (cheaply!). LibSubmarine Team, 2018. Accessed: 2018-08-28.
11. Anonymous. How the first winner of fomo3d won the jackpot? https://winnerfomo3d.home.blog/, 2018. Accessed: 2018-09-09.
12. 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.
13. 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.
14. I. Bentov, L. Breidenbach, P. Daian, A. Juels, Y. Li, and X. Zhao. The cost of decentralization in 0x and etherdelta. http://hackingdistributed.com/2017/08/13/cost-of-decent/, 2017. (Accessed on 08/14/2018).
15. 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).
16. G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of Computer and System Sciences*, 37(2):156–189, 1988.
17. 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.
18. 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.
19. S. Buti, B. Rindi, and I. M. Werner. Diving into dark pools. 2011.
20. 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.

21. P. M. da Silva, M. Matos, and J. Barreto. Fixed transaction ordering and admission in blockchains.
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. E. Hertzog, G. Benartzi, and G. Benartzi. Bancor protocol. 2017.
24. D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification. Technical report, Technical report, 2016–1.10. Zerocoin Electric Coin Company, 2016.
25. W. Kaal and M. Dell'Erba. Initial coin offerings: Emerging practices, risk factors, and red flags. 2017.
26. 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.
27. G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. An empirical analysis of anonymity in zcash. *arXiv preprint arXiv:1805.03180*, 2018.
28. 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.
29. 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.
30. J. Li and W. Mann. Initial coin offering and platform building. 2018.
31. L. Luu, D.-H. Chu, H. Olickel, P. Saxena, and A. Hobor. Making smart contracts smarter. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, pages 254–269. ACM, 2016.
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. 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.
35. A. Miller, M. Möser, K. Lee, and A. Narayanan. An empirical analysis of linkability in the monero blockchain. *arXiv preprint*, 1704, 2017.
36. S. Moosavi and J. Clark. Ghazal: toward truly authoritative web certificates using ethereum.
37. S. Nakamoto. Bitcoin: A peer-to-peer electronic cash system. 2008.
38. 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.
39. R. Radner and A. Schotter. The sealed-bid mechanism: An experimental study. *Journal of Economic Theory*, 48(1):179–220, 1989.
40. D. Siegel. Understanding the dao attack. *Web. http://www. coindesk. com/understanding-dao-hack-journalists*, 2016.
41. E. G. Sirer and P. Daian. Bancor is flawed. `http://hackingdistributed.com/2017/06/19/bancor-is-flawed/`, 2017. (Accessed on 06/14/2018).
42. 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.
43. D. Wang, J. Zhou, A. Wang, and M. Finestone. Lexisnexis. 2015. Accessed: 2018-09-12.

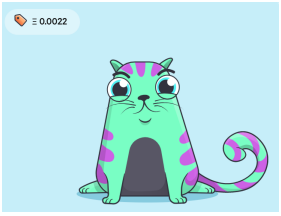

Fig. 9: Cryptokitty Number 842912

44. 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.
45. 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).
46. W. Warren and A. Bandeali. 0x: An open protocol for decentralized exchange on the ethereum blockchain. *URl: https://github. com/0xProject/whitepaper*, 2017.
47. 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.
48. H. Zhu. Do dark pools harm price discovery? *The Review of Financial Studies*, 27(3):747–789, 2014.

A CryptoKitty

Figure 9 shows an example of a Cryptokitty.

B LocalEthereum

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 [43]. 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 [19,48] and sealed bids [39] 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.

```
    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.

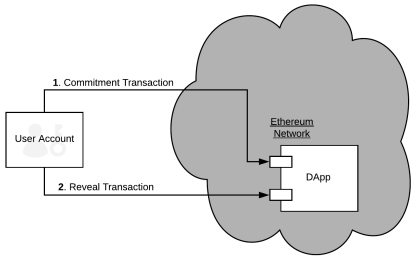

Fig. 10: Commit and Reveal. User sends a commitment transaction with the hash of the data, After the commitment period is over, user sends her reveal transaction to the DApp revealing the information that matches the commitment.

D Commit-and-Reveal

Figure 10 illustrates the commit/reveal approach.