# New Technology, Old Crimes

Shayan Eskandari

A Thesis in
The Concordia Institute for Information Systems Engineering (CIISE)

Presented in Partial Fulfillment of the Requirements
For the Degree of
Doctor of Philosophy
(Information and Systems Engineering)
at
Concordia University
Montréal, Québec, Canada

October 2023

# Concordia University
School of Graduate Studies

This is to certify that the thesis prepared

By: **Shayan Eskandari**

Entitled: **New Technology, Old Crimes**

and submitted in partial fulfillment of the requirements for the degree of

**Doctor of Philosophy (Information and Systems Engineering)**

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Chair
*Walter Lucia*

_____ External Examiner
*Kaiwen Zhang (ETS)*

_____ Examiner
*Amr Youssef*

_____ Examiner
*M. Mannan*

_____ Examiner
*Carol Fung*

_____ Supervisor
*Jeremy Clark*

Approved by _____
*Zachary Patterson, Graduate Program Director (CIISE)*

01 Sept 2023 _____
*Mourad Debbabi, Dean (GCS)*

# Abstract

Name: **Shayan Eskandari**

Title: **New Technology, Old Crimes**

Hello. No more than 250 words.

# Acknowledgments

Hello. Lorem ipsum dolor sit amet, consectetur adipiscing elit, sed do eiusmod tempor incididunt ut labore et dolore magna aliqua. Diam donec adipiscing tristique risus nec feugiat in fermentum posuere. Et netus et malesuada fames ac turpis. Nullam non nisi est sit. Felis eget velit aliquet sagittis id. Mauris commodo quis imperdiet massa tincidunt. Tellus molestie nunc non blandit massa enim nec. Facilisis mauris sit amet massa. Et molestie ac feugiat sed. Metus vulputate eu scelerisque felis imperdiet proin.

# Contents

# List of Figures

# List of Tables

# Chapter 1

# Introduction

The impact of blockchain technology on the world is undeniable. It has been a decade since the first blockchain application, Bitcoin, was introduced to the world. Since then, the technology has been adopted by many industries and has been the subject of many research studies. The technology has been used in many applications, from digital cash (*e.g.,* Bitcoin [157]), prediction markets [50], and decentralized governance [2]. Ethereum, one of the most popular blockchain platform that supports smart contracts, has been the most used platform for developing decentralized applications (DApps). Ethereum has also been used in many applications, from decentralized exchanges (*e.g.,* Bancor [105]), crypto-collectibles (*e.g.,* CryptoKitties [199]), gambling services (*e.g.,* Fomo3D [**?**]), and decentralized governance (*e.g.,* DAO).

Even though this technology is new and not fully matured, there are billions of dollars worth of assets stored and transacted on the blockchain, specifically Ethereum blockchain. A perception that crypocurrencies are criminal money has been formed

in the public opinion. This perception is not without merit, as there are many cases of fraud and theft in the blockchain space. However, the blockchain technology is not inherently criminal. In fact, the technology has the potential to reduce the crime rate by providing a transparent and immutable ledger.

The main theme of this dissertation is to analyze a few different types of attacks on blockchain networks and applications, and provide solutions to mitigate some of the attacks. Many of these attacks we study are not new and have been studied in the traditional financial systems. However, the decentralized nature of the blockchain applications makes the attacks feasible by many actors and potentially more dangerous.

## 1.1 Contributions

## 1.2 Outline

The rest of the dissertation is organized as follows: In Chapter 2, we offer a concise introduction to the essential concepts necessary for comprehending this dissertation. In Chapter **??**, we study the cryptojacking concept, its fast paced growth, and the ethical questions surrounding this use of the blockchain technology. In Chapter **??**, we study the frontrunning attacks on blockchain applications, provide a taxonomy of these attacks, and analyze the common mitigation methods. In Chapter **??**, we study the oracle implementations on the blockchain, and the attacks on blockchain

applications that use oracles. In Chapter **??**, we study the auditing of blockchain-based assets and the systematic challenges a company might have to properly audit their crypto-assets. Lastly, in Chapter **??**, we provide some concluding remarks and future research prospects.

# Chapter 2

# Background

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 work on all DApps but front-running is not necessarily beneficial, depending on the DApp's internal logic and/or as any mitigation it might implement. Therefore, DApps need to be studied individually or in categories. In this report, we draw from a scattered body of knowledge regarding front-running attacks on blockchain applications and the proposed solutions, with a series of case

studies of DApps deployed on Ethereum (a popular blockchain supporting DApps). We do case studies on decentralized exchanges (*e.g.,* Bancor), crypto-collectibles (*e.g.,* CryptoKitties), gambling services (*e.g.,* Fomo3D), and decentralized name services (*e.g.,* Ethereum Name Service). We also study initial coin offerings (ICOs). Finally, we provide a categorization of techniques to eliminate or mitigate front-running including transaction sequencing, cryptographic techniques like commit/reveal, and redesigning the functioning of the DApp to provide the same utility while removing time dependencies.

## 2.1 Preliminaries

Public blockchains are the most promising underlying technology for many applications. They are decentralized, transparent, and immutable. However, they are also slow, expensive, and have limited functionality. They can and will replace many intermediary entities we know of today. However as we will dive deeper in this subject, blockchain, as a technology is in its infancy. The public aspect of these blockchains changes many assumptions developers and system designers have about the data flow within their applications. In this section, we will go through the essential concepts of blockchain technology and how they are related to the front-running attacks. We give a background on the front-running defitions as they are used in the traditional financial systems and how they are generalized to the blockchain applications.

In order to understand how front-running attacks are feasible and potentially

dangerous in the blockchain applications, some concepts should be discussed.

### 2.1.1 Blockchain

Blockchain, is an online decentral ledger. This technology has the potential of many interesting applications, from digital cash (*e.g.,* Bitcoin [157]), prediction markets [50], and decentral governance [2]. Bitcoin, started in 2009, was the first application of blockchain technology and since then the concept of decentral ledger has grown to many other applications that just a ledger holding transaction data.

#### Ethereum

Ethereum [230] is a blockchain that attracted most of the developers compared to other blockchains. It extended on the features of its earlier cousin, Bitcoin, by adding functionalities such as smart contracts. Smart contracts are applications living on the blockchain that can immutably execute their verified code. Ethereum uses a Turing-complete virtual machine and enables programs to live and be executed on the blockchain. This is in contrast to Bitcoin that uses a UTXO[1] model and mainly supports value transfers. Bitcoin has a scripting language that can extend the functionality of the transactions to some extent, however, with Ethereum Turing-complete language the possibilities are limitless.

[1]Unspend Transaction Output

### 2.1.2 Smart Contracts

Smart contracts are small codebases (applications) that live on a blockchain. The technical details of smart contracts are not necessary to understand for this report. In short, smart contracts can be seen as blackbox applications that get inputs from a user and follow the code flow to the output, which can trigger monetary transactions. Smart contracts, mostly has been used for tokenized economy, except some technical limitation, the functionalities are limitless. From unstoppable gambling games to complete voting and payroll systems.

However, as noted earlier, everything on a blockchain is compromised of transactions and blocks. The order of the transactions in each block indicates the order of events in the Ethereum blockchain. Given that miners, and recently entities named *block builders*, are in control of the order, it is possible for these entities to reorder the transaction in a block, or even not include a transaction in a block for higher financial gain from the new order. This is the basics of blockchain front-running that we discuss in the next chapters.

### 2.1.3 Ethereum Network

The peer to peer aspect of Ethereum network enables the possibility for a full decentralized network. As the vision of Ethereum is a world computer that everyone can use, the network is designed to be accessible to everyone. This means that anyone can run a full node and be part of the network. The network is designed to be trustless, meaning that the nodes do not need to trust each other. The network is

also permissionless, meaning that anyone can join the network and be part of the consensus. This is in contrast to the traditional financial systems that are centralized and permissioned, meaning that only a few entities are in control of the network and the information flow.

### mempool

When a user sends a transaction to the Ethereum network, one node will validate the transaction and propagate it through the network. While this transaction is still not included in the block, it is stored in the memory of all the nodes, also known as *mempool*. The mempool is a list of the transactions that are not yet included in the blockchain, however the order of the transactions are different for each node, as they have received the transactions in different order. This opens up the block builder to the possibility of reordering the transactions in the block they build for their own financial gain.

# Chapter 3

# Browser-based Cryptojacking

*This chapter is based on the paper "A first look at browser-based Cryptojacking" published in the IEEE Security & Privacy on the Blockchain (IEEE S&B) 2018 co-located with EuroS&PW at University College London (UCL) [81]. This paper was supervised by Jeremy Clark and co-authored by Andreas Leoutsarakos, Troy Mursch.*

## 3.1 Introduction

Bitcoin [157] emerged almost a decade ago as an open source project, which mushroomed into a cryptocurrency sector collectively capitalized at over $500 billion USD[1]. Every day, people new to the concept of cryptocurrencies look for a quick and simple way to acquire some crypto-wealth. In the early days of Bitcoin, users on their personal computers could effortlessly acquire the currency through mining—a process

[1]Coinmarketcap - Global Charts - Accessed: 2017-12-14 `https://coinmarketcap.com/`

Bitcoin uses to incentivize nodes to verify transactions as they are recoded in the blockchain. However, a second wave of mining technology saw users augmenting the CPU power of their computers with GPUs. Other groups of people deployed snippets of JavaScript code on websites that recruited their visitor's CPU power, often unknowingly, to mine for them as part of a bigger mining network (*i.e.,* a mining pool). However, both approaches quickly became infeasible as the computing power required to mine bitcoins grew exponentially to over 12 petahashes[2]. This was due to the emergence of application-specific integrated circuits (ASICs) and collective mining pools, which continue the third wave of mining to this day [159].

As the years passed and a few key cryptocurrencies emerged as the market leaders, the concept of browser mining largely became forgotten. Today, the most common way for the average person to acquire cryptocurrencies is to purchase them. It came as a surprise to many when stories began to circulate on popular media outlets this year about websites mining cryptocurrencies through browsers again. Figure 3.1 shows how the searches for "browser mining" have changed since Bitcoin was launched. Websites like The Pirate Bay [87] experimented with browser mining as a way to add a new revenue stream, while others like Showtime.com [210] claimed they had the code injected after they were discovered.

This paper tells the story behind the rejuvenation of browser-based mining. It is centred on *cryptojacking* (also known as *coinjacking* and *drive-by mining*), a term coined to refer to the invisible use of a vulnerable user's computational resources to

[2]Bitcoin hash rate - Accessed: 2017-11-20 `https://blockchain.info/charts/hash-rate`

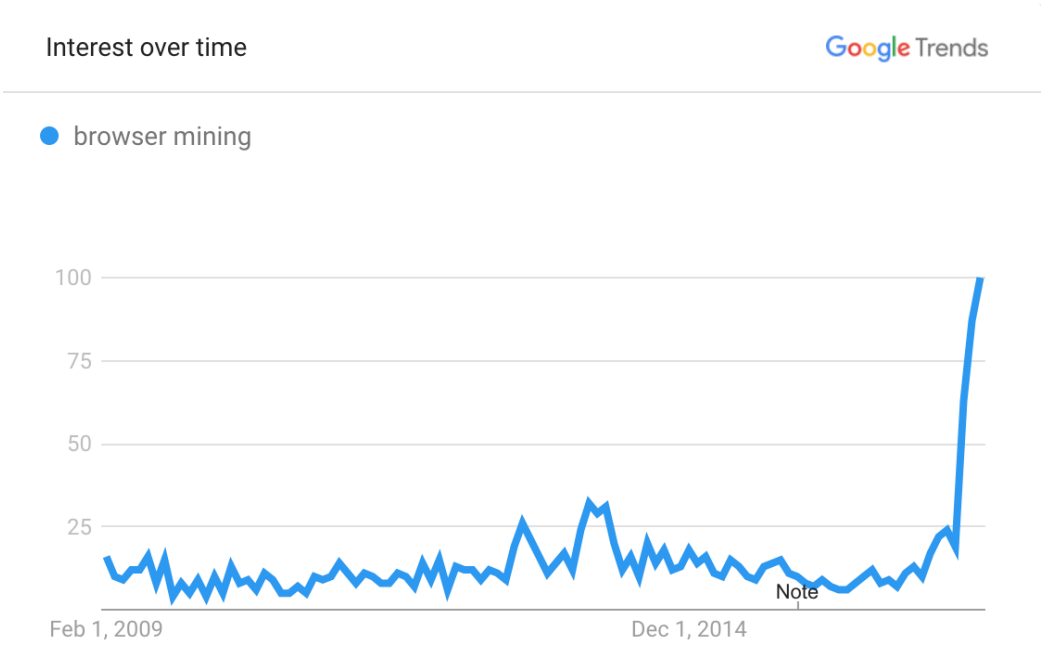



Figure 3.1: Search interest for "browser mining" over time. Search interest seems to have piqued during price surges, which culminated with Bitcoin crossing $1000 USD for the first time in December 2013. Soon after Bitcoin's first major crash searches consistently waned until a recent large spike, which is more than 4 times the lifetime average. The waning period before the recent surge could be attributed to the advent of ASIC usage for Bitcoin mining, and the surge is likely due to the revival of browser mining for non-Bitcoin currencies that have gained a sizeable market capitalization.

mine cyptocurrencies. Technically in-browser mining is a subset of cryptojacking, although most uses of the term apply to browser-based mining. In this case, mining happens within the client browser when the user visits the website. We have also seen the term cryptojacking applied to malware that mines cryptocurrencies, or in the situation where malware renders a machine as an unwitting participant in a botnet, and the botnet is rented for the purposes of mining crypcurrencies (*cf.* [107]). The resource consumption of in-browser cryptojacking can noticeably degrade a computer's performance.

## 3.2 Preliminaries and Related Work

### 3.2.1 Browser-based Mining

#### Early days

The idea of in-browser mining started in the early days of Bitcoin. Bitcoin Plus[3] is one example of a discussion on replacing ads with Bitcoin browser miners[4]. It was also argued that browser-based mining provides greater scalability and decentralization as the barrier to entry is lowered to any unmodified computer with an internet connection. Soon after there was a rise in Bitcoin JavaScript miners such as JSMiner (2011)[5] and MineCrunch (2014)[6]. MineCrunch's visibility was increased by campaigns and the active online presence of its developers. Based on the developer claims, MineCrunch was well optimized for Javascript, but still worked 1.5x slower than native applications for CPU mining (*e.g.,* CPUMiner[7]). Although CPU mining became uncompetitive with GPU and ASIC-based mining, it remained a sandbox for botnet admins to experiment with the thousands of CPUs at their disposal. Botnet mining has been studied in the literature [107, 232], as well as covert mining within enterprises and cloud environments [198].

In addition to unprofitability, browser-based mining faced legal challenges. In May

[3]Bitcoin For the Uninitiated: Now, A Browser-Based Mining Client May 19th, 2011 `https://www.themarysue.com/browser-based-bitcoin-mining/`

[4]BitCoin browser mining as a replacement for ads `https://www.reddit.com/r/Bitcoin/comments/ieaew/bitcoin_browser_mining_as_a_replacement_for_ads/`

[5]A JavaScript Bitcoin miner `https://github.com/jwhitehorn/jsMiner`

[6]MineCrunch, web(JS) miner with integration feature `https://cryptocurrencytalk.com/topic/24618-minecrunch-web-js-miner-with-integration-feature/`

[7]CPU miner for Litecoin and Bitcoin - `https://github.com/pooler/cpuminer`

2015, the New Jersey Attorney General's office reached a settlement with the developers of "Tidbit", a browser-based Bitcoin miner. Terms of the settlement included ceasing operations of Tidbit. Then acting Attorney General John J. Hoffman stated "No website should tap into a person's computer processing power without clearly notifying the person and giving them the chance to opt out." [164].

**From one CPU to ASICS and mining pools**

The first Bitcoin block mined on a GPU happened on July 18th, 2010 by a user named ArtForz [227], by using a private mining code that he developed himself. It was not until mid-2011 that others started implementing and releasing open source GPU-based mining tools. These tools greatly increased mining efficiency due to the hashing power of a GPU and the massive parallelizing possible with multiple GPUs (also known as mining rigs). The move from software to hardware followed shortly after. First, programmable FPGA chips resulting in custom-built circuits specifically for mining[8]. Then by mid-2012, companies started selling ASICs designed specifically for Bitcoin mining. After delay of about a year in delivering ASIC products, Bitcoin mining started transitioning from GPUs to ASICs where it remains today. Consequently, the hashing power of the Bitcoin network increased and the mining difficulty followed. To illustrate the change, consider a desktop PC CPU mining at 10 MH/s: on expectation, it will take 425 years before mining a single block [107].

In parallel to the evolving technology, collective action emerged through the use of

[8]Custom FPGA Board for Sale! (August 18, 2011) `https://bitcointalk.org/index.php?topic=37904.0`

mining pools. A mining pool is a collective of individual miners. Participants receive a slice of work for mining the current block on behalf of the pool. If a member of the pool mines the block, the block reward is split amongst the participants of the pool *pro rata* according to their computational effort [182]. As an aside, a very elegant protocol for reporting 'near-solutions' to the pool enables participants to prove, without trust, the level of effort they are contributing to the pool at all times. In general, a mining pools cannot amplify earnings, they only change their shape. An income stream from a pool is a steady trickle, while solo-mining results in sporadic dumps of income. The first Bitcoin block found on a mining pool was on December 16, 2010 that was a beta implementation of a pool operated by a user named *slush*.

### 3.2.2 Monero

Launched in April 2014, Monero [151] is a cryptocurrency alternative to Bitcoin. It purportedly offers increased privacy by obfuscating the participants in a transaction, as well as the amounts. This is in contrast to more popular cryptocurrencies like Bitcoin and Ethereum, where a pseudonymous-but-complete transaction graph can be constructed from the public blockchain. Recent research has shown Monero's obfuscation techniques are less effective than originally claimed [147, 124]. Since regulation on exchanging between cryptocurrencies is lighter than exchanging cryptocurrencies for fiat money, and such services are not geographically bound, obtaining Monero for Bitcoin and vice versa is efficient and enables Monero to be used as a short-term medium of exchange for Bitcoin holders. This approach (and Monero's acceptance)

is particularly popular on so-called dark web markets; markets that do not ban illicit goods and services.

A second characteristic that distinguishes Monero from Bitcoin is in the mining algorithm it uses. Monero still employs proof-of-work, specifically an algorithm called CryptoNight [62]. However the computational puzzle is designed to be *memory-hard*: it requires the storage of a large set of bytes and then requires frequent reads and writes from this memory. Such puzzles are optimized for CPUs with low-latency memory-on-chip, and not as well suited for circuits like FPGAs and ASICs. CryptoNight requires approximately 2MB per instance, which fits in the L3 cache of modern processors. Over the course of the next few years, these L3 cache sizes should become mainstream and allow more CPUs, and thus users, to participate in Monero's ecosystem. It has also been shown that ASICs cannot handle more than 1MB of internal memory, which is less than the size of memory required to calculate a new block. GPUs are also at a disadvantage since GDDR5 memory, which are used in modern GPUs and considered one of the fastest types of memory, is notably slower than L3 cache [216].

### 3.2.3 Coinhive

Monero built on its early success and continued to gain in popularity over the years, which caught the attention of some developers who decided to revisit the idea of browser mining. See Table 3.1 for a timeline of events. One of the earliest efforts appeared in September 2017 and was called Coinhive [53]. Soon after, a competitor

| | |
|---|---|
| **2014-04-18** | Monero Cryptocurrency released. |
| **2017-09-14** | Coinhive Miner launched. |
| **2017-09-17** | ThePirateBay caught using coinhive [203]. |
| **2017-09-21** | Adblockers started to block coinhive scripts. |
| **2017-09-24** | Showtime caught running coinhive [210]. |
| **2017-09-25** | Coinhive clones started to appear. |
| **2017-10-13** | PolitiFact website compromised [225]. |
| **2017-10-16** | Coinhive launched authedmine - authorized mining. |
| **2017-11-23** | LiveHelpNow Hack incident [130]. |
| **2018-01-25** | Cryptojacking code found on Youtube ads [145]. |
| **2018-02-11** | UK Information Commissioner's Office incident [208]. |

Table 3.1: Timeline of Monero and in-browser mining reports

named Crypto-Loot[9] emerged. Both websites provided APIs[10] to developers for implementing browser mining on their websites that used their visitors' CPU resources to mine Monero. A portion of mined Monero would go back to the API developer, and the rest would be kept by the website. Not long after their early success, several copycats appeared such as Coin-Have and PPoi [68] to take part in the reborn practice. It even inspired a new coin specifically designed for browser mining named JSECoin,[11] which has yet to find an audience. These developments took place over the course of a few weeks, which signalled the renewed success of browser mining. However, Coinhive's approach as a legitimate group set it apart from its peers and established itself as the leader in the space. They also launched separate services such as proof-of-work CAPTCHAs and short-links, which could be used to prevent spam while mining Monero [53].

## 3.3 Threat Model

In-browser mining is considered as an abuse unless user's consent is granted. The attack surface to abuse users' browsers through cryptojacking is broad, and there are multiple vectors where various entities can inject mining scripts in the website's codebase. We summarize those here.

[9] Crypto-Loot - A web Browser Miner — Traffic Miner — CoinHive Alternative `https://crypto-loot.com/`

[10] Application Programming Interface

[11] JSEcoin's Website Cryptocurrency Mining `https://jsecoin.com/`

### 3.3.1 Webmaster initiated

A website administrator can add a mining script to her webpage, with or without informing users. Website owners may do this to monetize their sites, especially when they have been blacklisted or blocked by standard advertising platforms. In one example, a researcher found Coinhive on a large Russian website offering child pornography to users [184]. Revenue estimates, based on the website traffic data available, were roughly $10,000 a month after converting the value of XMR mined to USD.

### 3.3.2 Third-party services

Many websites serve active Javascript from third parties within their own webpages. This could be ads from an ad network, accessibility tools or tracking and analytics services. Third parties with these privileges can inject cryptojacking scripts into the sites that use them, either intentionally or as a result of a breach. The first two incidents, Coinhive was injected into the websites of Movistar[12] and Globovision[13] using Google Tag Manager[14]. Movistar stated that Coinhive was not put on their website by a hacker, but instead was due to *"internal error"* while they were conducting "production tests". No statement was provided by Globovision on why the cryptojacking scripts appeared on their site on November 15, 2017 [206]. Another high-profile cryptojacking case involving a Google platform occurred in January 2018 when Trend

[12]Movistar is a major telecommunications brand owned by Telefonica, operating in Spain and in many Hispanic American countries `https://www.movistar.com/`

[13]Globovision is a 24-hour television news network in Venezuela and Latin America `http://globovision.com/`

[14]Google Tag Manager is a tag management system created by Google to manage JavaScript and HTML tags used for tracking and analytics on websites

Micro researchers found advertisements containing Coinhive miner script were shown to YouTube users in Japan, France, Taiwan, Italy, and Spain for nearly a week [145]. Similar to Showtime, YouTube inherently has a high average visit duration as a video streaming site and thus is prime target for cryptojacking operations.

### 3.3.3 Browser extensions

Cryptojacking was not limited to websites in 2017. The Chrome extension *Archive Poster* remained on the Chrome Web Store for days while silently cryptojacking an unknown portion of their 100,000+ users. After multiple user reports, followed by multiple news media outlets covering the issue, the extension was removed [92]. Similar cryptojacking extensions has been identified on less popular Mozilla Firefox add-ons as well.

### 3.3.4 Breaches

If an attacker is able to breach principle servers, websites, extensions, or the scripting services they use, they can inject cryptojacking scripts that will impact the site's users without the site's knowledge or consent. For example, a researcher found a malicious modification to webchat system LiveHelpNow's SDK; it resulted in unsolicited mining across nearly 1500 websites using their chat support service [130] such as retail store chains Crucial and Everlast websites. In another example, Coinhive was found on the political fact-checking website PolitiFact[15] A compromised JavaScript library was

[15]PolitiFact: Fact-checking US politics `https://politifact.com/`

found to be injecting the cryptojacking code. The malicious code remained on the site for at least four hours before it was removed [225]. PolitiFact executive director stated, "Hackers were able to install their script on the fact-checking website after discovering a misconfigured cloud-computing server" [220].

Another recent example of such incident is a breach in a website plugin called *Browsealoud*[16] led to injection of cryptojacking scripts in some United Kingdom governmental websites such as *Information Commissioner's Office, UK NHS services, Manchester City Council* and around 4200 other websites [208]. Within the same month, cryptojacking script was seen on *Tesla* and *LA Times* websites through poorly secured cloud configuration [158].

### 3.3.5 Man-in-the-middle

A user's web traffic is often routed through intermediaries that may have plaintext access to content. For example, internet service providers or free public wireless routers can inject cryptojacking scripts into non-HTTPS traffic. Advertisement code injection has been seen in practice before [209] and there have been assertions of similar injections of browser mining scripts at certain Starbucks free Wi-Fi hotspots in Argentina[17].

[16] An accessibility tool to read the content aloud in multiple languages `https://www.texthelp.com/en-gb/products/browsealoud/`

[17] `https://twitter.com/imnoah/status/936948776119537665`

## 3.4 Measurements

### 3.4.1 Prevalence of Coinhive and alternatives

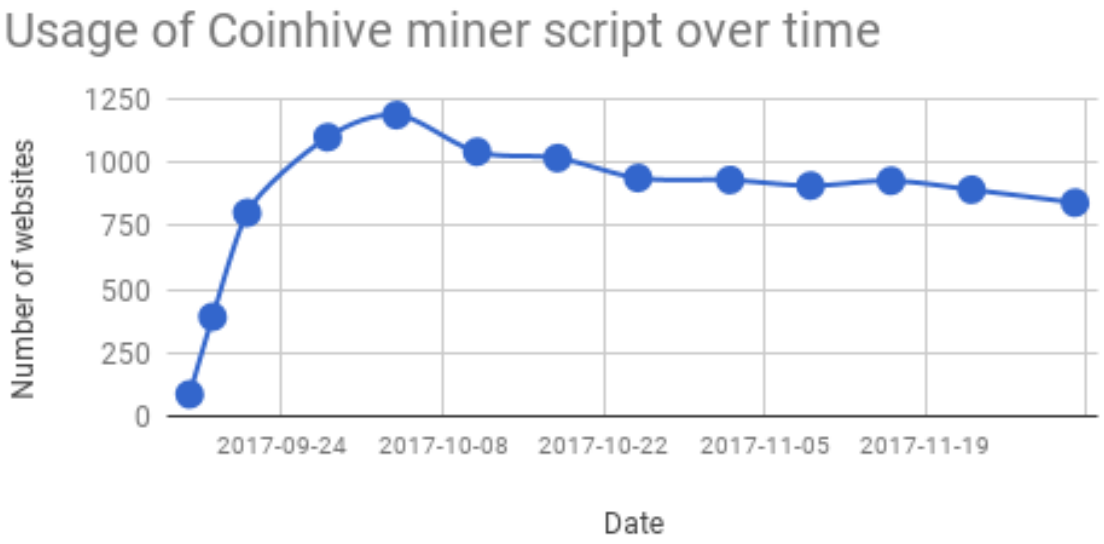



Figure 3.2: The number of instances of the Coinhive miner scripts found using the query in Figure 3.3 in top one million websites over a three month period beginning with the release of Coinhive in September 2017.

Based on the fact that Coinhive is the dominant website offering in-browser mining (see Figure 3.4), we first focus on measuring the prevalence of Coinhive scripts deployed on internet sites. We use the censys.io BigQuery dataset [74] for the top million sites indexed by Zmap[18]. We simply look for the `coinhive.min.js` script within the body of the website page. The query we use is in Figure 3.3 and the results over a two month period are provided in Figure 3.2. These findings are corroborated by another search engine, PublicWWW[19], which indexes the source code of publicly

[18] `https://zmap.io`
[19] Search Engine for Source Code `https://publicwww.com/`

```
SELECT domain, tags, p80.http_www.get.headers.content_language, p80.http
FROM censys-io.domain_public.20171123
WHERE STRPOS(p80.http.get.body, coinhive.min.js) > 0 or STRPOS(p80.http_
```

Figure 3.3: A BigQuery SQL query to find websites that embed the Coinhive script using a dataset of the top one million sites from censys.io.

available websites. Using PublicWWW's dataset, over 30,000 websites were found to have the `coinhive.min.js` library [155]. As seen from our data in Figure 3.2, the adoption of this script was substantial in the first days of its release. However, progress slowed down at the same time as ad-blockers and organizations started to block Coinhive's website. The initial purpose of this service, as claimed by Coinhive, was to replace ads and cover server costs for webmasters. As the service did not require that websites receive user consent before running the miner code, it started to be used maliciously in users' browsers. This type of usage resulted in Coinhive being included in some company's top-10 most wanted malware list [47].

This type of measurement will become less accurate moving forward. Cryptojacking services are evolving to use obfuscated JavaScript and randomized URLs to evade detection[20]. An example of these methods can be found in the cryptojacking service provider called Minr. In this case, the script is automatically obfuscated for users implementing the code. In addition, the domain names used by Minr frequently change to circumvent blocklists and anti-malware software.

Coinhive has begun to be blocked by enterprises. One example is shown in Figure 3.7. This blocking seems to have sent Coinhive operators to lesser known alternatives with the same or similar functionality. We used the same methodology on

[20] `https://twitter.com/bad_packets/status/940333744035999744`

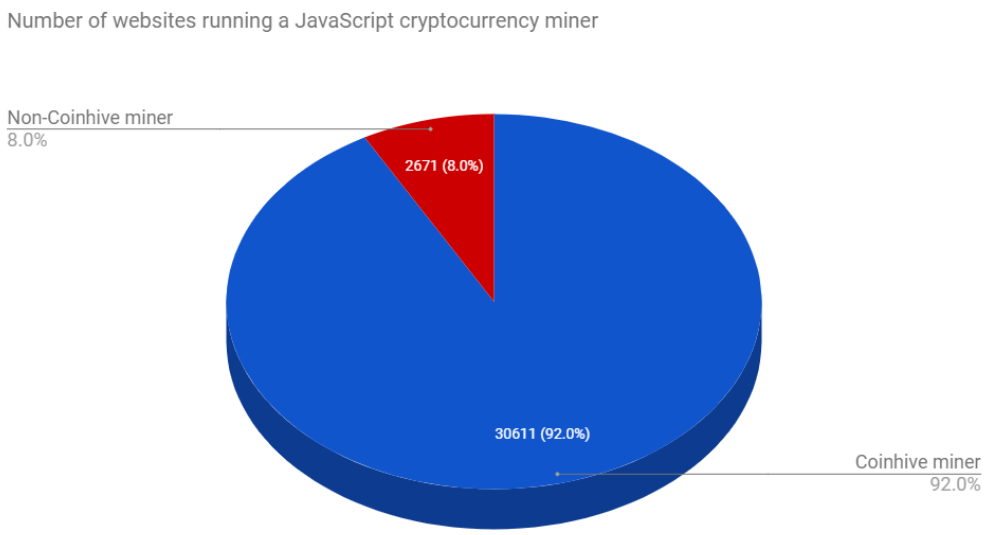



Figure 3.4: Share of websites using a Javascript cryptocurrency miner, details in Table 3.2

PublicWWW dataset to find the usage of Coinhive and its alternatives on the internet. Table 3.2 shows the keywords used to identify these services. The result can be found on Figure 3.4 and Figure 3.5.

Coinhive has also reacted by focusing on adding methods to enforce asking for user consent and legitimizing the use of cryptojacking. It introduced another domain and service called *Authedmine*, which requires user's consent to start mining in the browser. This service did not get the same attention as the original service, but it did inspire discussions regarding the ethics of such services, which is discussed in Section 3.6. Using the same methodology, censys.io was used to measure the prevalence of AuthedMine and show the results in Figure 3.8.

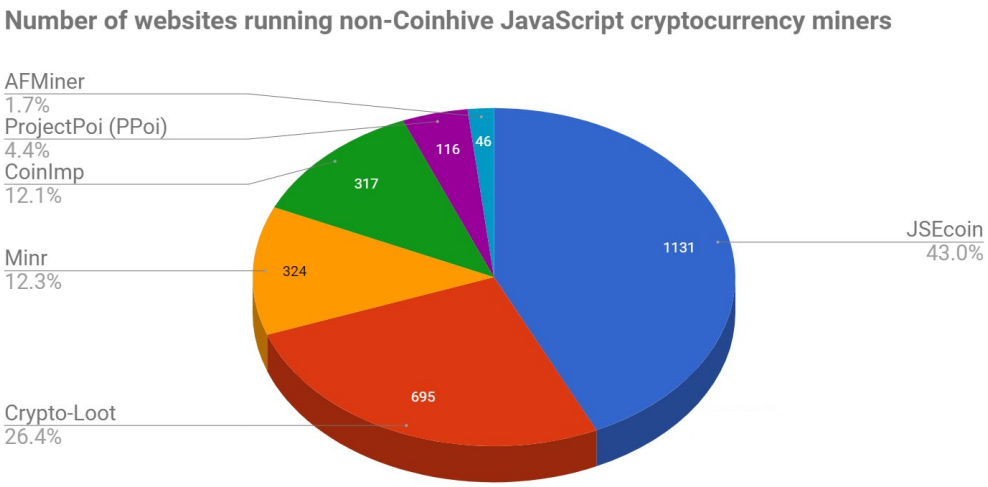



Figure 3.5: Share of websites using a Coinhive alternative, details in Table 3.2

### 3.4.2 Client impact

Most cryptojacking scripts discovered were configured to use around 25% of user's CPU, which can be justified as it will be under the threshold of attracting the user's attention, and it could be argued as fair-usage of their hardware. During the first few days, however, there were some reports of 100% CPU usage while visiting websites containing these scripts [204], which can be characterized as malicious. By default, the Coinhive JavaScript library will use all available CPU resources. The user implementing the script must include a throttle value to reduce the client-side CPU usage during mining operations. We show an example in Figure 3.9.

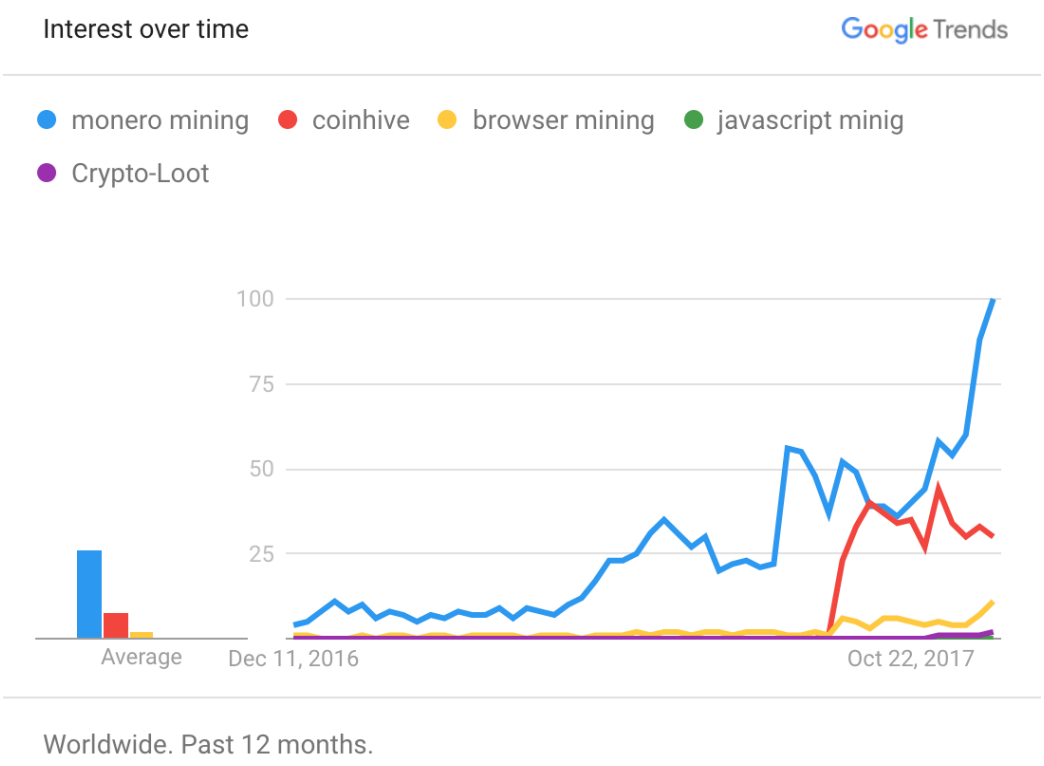



Figure 3.6: Google Trend over last 12 months: there has been more interest in Coinhive than the broader, related search term "Browser mining". Comparing to other services offering Monero browser mining API, Coinhive had the advantage of being the first to offer the service.

### 3.4.3 Profitability

Coinhive developers estimate a monthly revenue of about 0.3 XMR (about $101 USD) for a website with 10-20 active miners [53]. We sought to validate this estimation with a real world data set provided to us[21]. One of the biggest Coinhive campaign operators is a domain parking service. It runs Coinhive on over 11 000 parked websites. While visits to parked domains are considerably shorter than an average website, the data spans a period of three months and gives some insight into the profitability of cryptojacking. During the experimental period of about 3 months, they accumulated

[21] In collaboration and with thanks to Faraz Fallahi `https://github.com/fffaraz`

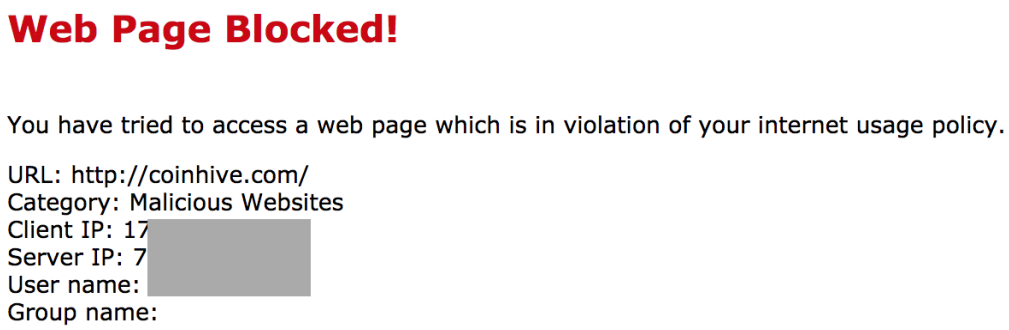



Figure 3.7: Concordia university has categorized the coinhive.com website as malicious and has blocked it.

105 580 user sessions for an average of 24 seconds per session. For the period examined, the revenue was 0.02417 XMR (Monero's currency) which at the time of writing is valued at $7.69 USD. Further detail is provide in Figures 3.10 and 3.11. While an A/B test was not setup to determine how much traditional web advertising would have brought in, freely available web calculator tools suggest we might expect an order or two of magnitude greater for comparable traffic.

## 3.5 Mitigations

We discuss the ethics of cryptojacking in the next section, but in the case of cryptojacking without user consent, it is seems natural to us to presuppose users want to be protected. Protection might take a few forms, which we outline here.

### 3.5.1 Obtaining consent

Cryptojacking tools might attempt to legitimize the practice by first obtaining user consent on a service provider level. An example of this is the Authedmine service

| Website | Results | Query Parameter |
|---|---|---|
| Coinhive | 30611 | ‘coinhive.min.js‘ |
| JSEcoin | 1131 | ‘load.jsecoin.com‘ |
| Crypto-Loot | 695 | ‘CryptoLoot.Anonymous‘ |
| Minr | 324 | ‘minr.pw‘,‘st.kjli.fi‘, ‘abc.pema.cl‘,‘metrika.ron.si‘, ‘cdn.rove.cl‘,‘host.d-ns.ga‘, ‘static.hk.rs‘,‘hallaert.online‘, ‘cnt.statistic.date‘,‘cdn.static-cnt.bid‘ |
| CoinImp | 317 | ‘www.coinimp.com/scripts/min.js‘, ‘www.hashing.win‘ |
| ProjectPoi (PPoi) | 116 | ‘projectpoi.min‘ |
| AFMiner | 46 | ‘afminer.com/code/miner.php‘ |
| Papoto | 42 | ‘papoto.com/lib/papoto.js‘ |

Table 3.2: Cryptojacking data was gathered by totalling the number of websites which had the following libraries in their source codes, indexed by PublicWWW by 12/24/2017. Figure 3.4 and Figure 3.5 are visualizations of these result.

from Coinhive discussed previously. Malicious sites might also opt for a service like Authedmine if it is whitelisted on its users‘ networks and then attempt to circumvent the consent process. For example, consent that requires a click from the user has been shown in some circumstances to be vulnerable to clickjacking attacks [185].

While cryptojacking is nowhere near the prevalence of tracking cookies, eventually it might grow into a regulatory issue where governmental bodies could use legislative approaches to obtain consent, similar to the provisions many countries now use for cookies (including honouring the ‘do not track’ HTTP header and obtaining click-based consent).

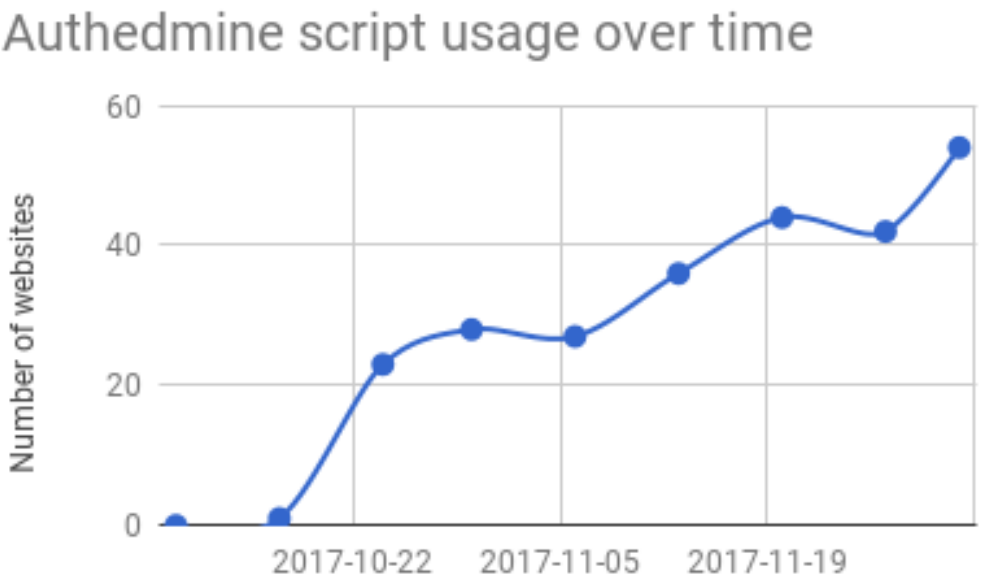



Figure 3.8: Usage of AuthedMine Miner scripts in top one million websites since its introduction

### 3.5.2 Browser-level mitigation

Browser developers have begun discussion of intervening in cryptojacking[22]. Potential mitigations include: throttling clientside scripting, warning users when clientside scripting consumes excessive resources, and blocking the sources of known cryptojacking scripts. Determining appropriate for thresholds for client-side processing that are high enough to allow legitimate applications and low enough to deter cryptojacking is an open research problem, as would be the wording of any notifications to the user that would lead the user to make an informed decision about allowing or not allowing resource consumption (*cf.* SSL/TLS warnings [194, 193, 10]). Browsers such as Opera, have taken a stance against cryptojacking scripts and blocked them via

[22]'Please consider intervention for high cpu usage js' `https://bugs.chromium.org/p/chromium/issues/detail?id=766068`

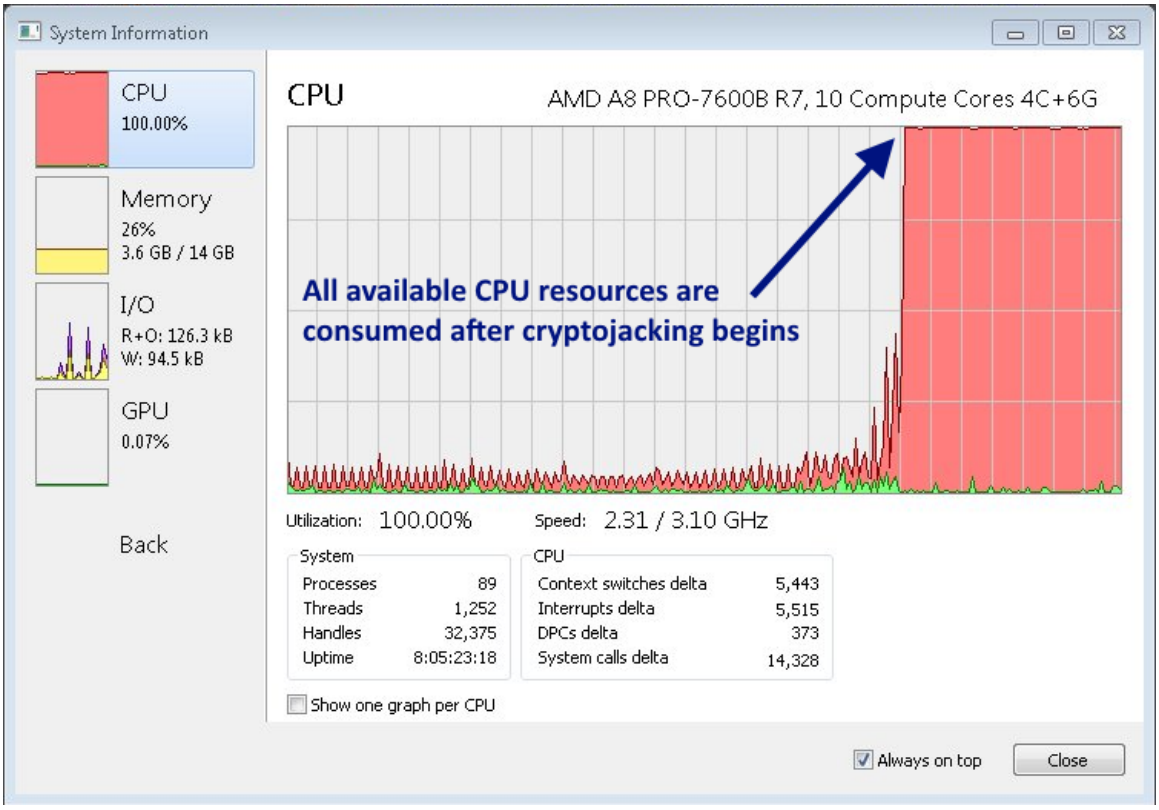



Figure 3.9: Comparison of CPU usage of browser without and with browser mining enabled.

their "NoCoin" blacklist [165]. It is too early to determine the effectiveness of using a blacklist to block such activities.

It is worth noting that some browsers might actually take the exact opposite approach and promote (consensual) in-browser mining, as it enables a form of monetizing websites independent of both (1) ad networks and the user tracking that accompanies the current ad model, and (2) users maintaining some form of credits or currencies for making micropayment to websites they use(*e.g.,* Brave Browser [23]). Browser mining has been shown to not be as efficient as native mining applications today. Therefore, optimizations on how browsers pass system calls to the operating

[23] `https://brave.com`

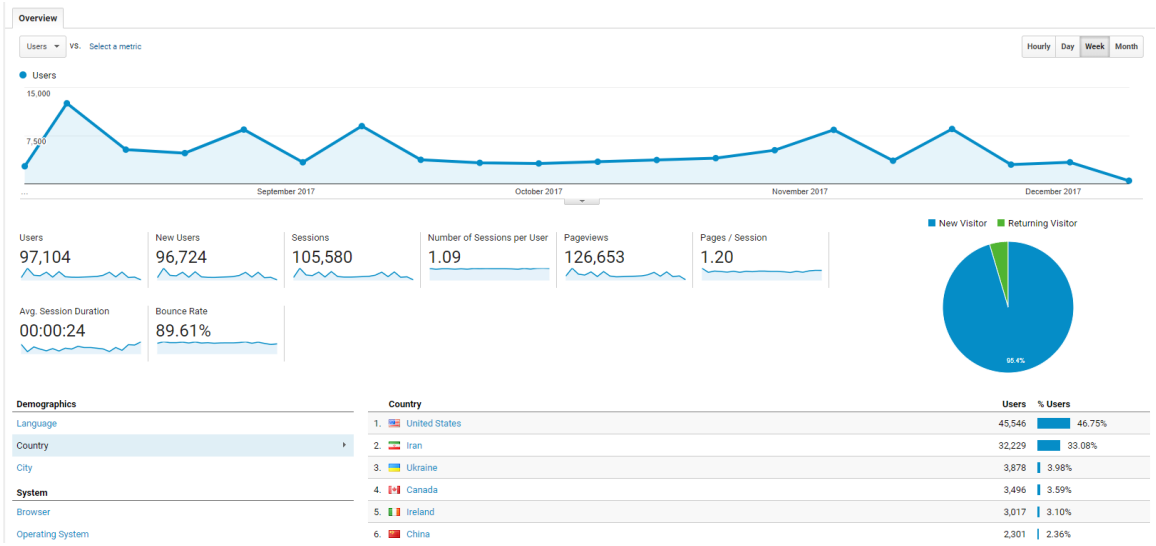



Figure 3.10: Google Analytics dashboard showing the number of visitors to a domain parking service of 11 000 domains.

system can be made, or there can even be browsers designed specifically to support efficient browser mining.

## 3.6 Discussion

While cryptojacking might be relatively new, it fits the pattern of various other technologies deployed on the web that raise ethical questions. In thinking about it, we distinguish a few cases: (1) the use of cryptojacking on a breached website, (2) the use of cryptojacking by the website owner with an attempt at obtaining user consent, and (3) the use of cryptojacking by the website owner without obtaining user consent. We would argue that (1) is clearly unethical; invariant to one's views on the ethics of hacking, we cannot see a justification for a breach that profits the adversary without any external benefits to anyone else.

The second case, cryptojacking after gaining user consent, is controversial primarily because it is unclear if users understand what they are consenting to, what they receive in return (some examples might include the elimination of ads, premium features, paywalled content, or higher definition video streams), and whether it is a fair exchange. To understand the zeitgeist, consider a recent poll conducted by Bleeping Computer that found: "many users said they are OK with websites mining Monero in the background if they don't see ads anymore" [28]. Coinhive released AuthedMine in recognition of the importance to many of user consent. ThePirateBay.org [22] ran cryptojacking scripts while users searched for torrent files without notice in their Privacy Policy, nor any visible warning on any part of the website that informed their users of this activity. This resulted in a backlash against the website, which responded with the following statement, "Do you want ads or do you want to give away a few of your CPU cycles every time you visit the site?" [204]. While the admins admitted to their testing of browser mining, their notice came after it was discovered and they ultimately removed the code. In both auction-based and keyword-based online advertisement, the advertiser pays the advertisement publisher to distribute the advertisement and the advertisement publisher pays a portion of the revenues to the website owner whom the advertisement was shown on her website [120]. However with in-browser mining as a replacement monetization strategy, a more direct compensation is established with less intermediaries which could benefit users and sites alike.

The potential harm to users of cryptojacking is higher energy bills, along with

accelerated device degradation, slower system performance, and a poor web experience [190, 203]. While consent may be obtained from the user, it is unclear if the user's mental model of how they are paying can be made clear to them. On the other hand, the privacy disclosures users make in the traditional advertising model are also intangible; it is doubtful users understand what they are consenting to when they, for example, consent through a banner [86] to the use of tracking cookies; and many websites waste computational resources without consequence through buggy scripting and unnecessary libraries. In short, the ethics are not clear-cut and should be debated.

One webservice prone to cryptojacking is video streaming—the longer a user is engaged on a website, the more income can be earned through browser mining. Showtime.com [205] and UFC.com [207] are two popular streaming sites that were asserted by researchers to have deployed Coinhive. Showtime has declined to comment on how or why Coinhive was implemented on their website. Speculation has been raised that it was injected via a third-party analytic tool, New Relic, due to Coinhive being found inside the New Relic code block within showtime's website source code. However a New Relic representative denied these claims in a statement to The Register, "It appears [Coinhive scripts] were added to the website by [Showtime's] developers." [205]. In a statement released by the UFC, they denied the presence of the code stating, "[they] did not find any reference to the mentioned Coinhive JavaScript [code]"[24].

The third case is the use of cryptojacking without user consent. Moor, in "What

[24] `https://twitter.com/bad_packets/status/928044219222048769`

is Computer Ethics?" [152] introduces the concept of an *invisible factor* for invisible computer operations in society. Based on his definitions, we would classify cryptojacking that does not gain user consent as *invisible abuse*: the intentional use of the invisible operations of a computer to engage in unethical conduct. Here the cryptojacker is earning money from unaware users that are being charged on their electricity bill. As discussed before, we already have court cases against such activities [164] and regulations for activities such as online user tracking [86], which indicates the need to start discussions and regulation on in-browser mining to fill in this policy vacuum as well.

## 3.7 Acknowledgements

J. Clark thanks NSERC and FRQNT for partial funding of this research.

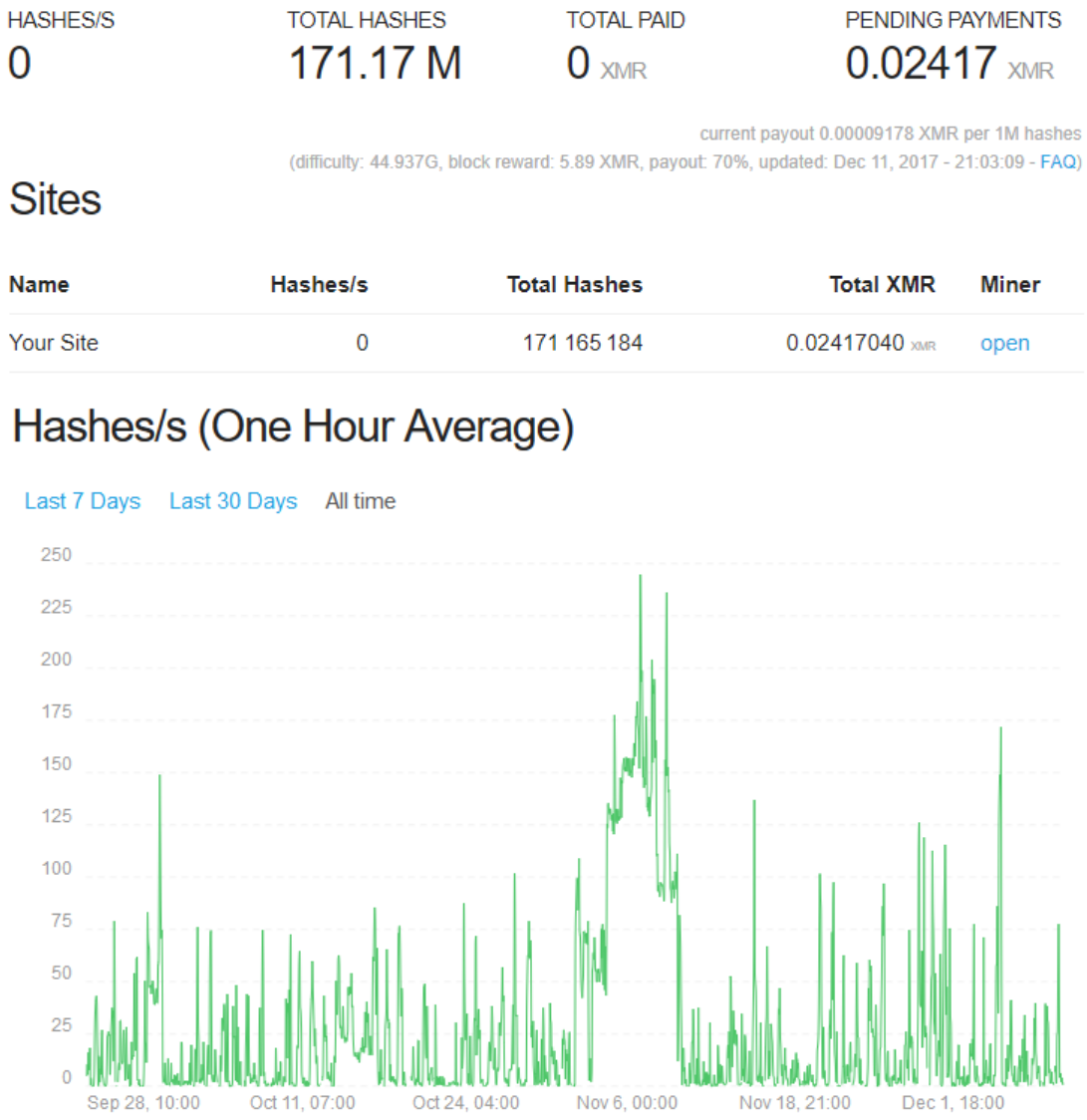



Figure 3.11: Coinhive dashboard showing the earnings of a domain parking service that runs Coinhive on 11 000 domains. Over the course of about 3 months, the operator earned 0.02417 XMR (currently $7.69 USD).

# Chapter 4

# Transparent Dishonesty: front-running attacks on Blockchains

*This chapter is based on the paper "SoK: Transparent Dishonesty: front-running attacks on Blockchain" published at 3rd Workshop on Trusted Smart Contracts In Association with Financial Cryptography (FC) in February 2019 [82]. This paper was supervised by Jeremy Clark and co-authored by Seyedehmahsa Moosavi.*

## 4.1 Background

In this chapter, we briefly describe what front-running is in the traditional markets and how it is related to other concepts regarding its legality and ethics. Additionally, we discuss the background of front-running on the blockchain applications and how the advancement and complexity of the technology has increased the attack vector to applications that previously were not vulnerable to information flow attacks. Finally, we discuss the related literature in this area.

## 4.2 Traditional Front-running

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

Cases of front-running are sometimes difficult to distinguish from related concepts

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

## 4.3 Literature on Traditional Front-running

Front-running originates on the Chicago Board Options Exchange (*CBoE*) [138]. The Securities Exchange Commission *(SEC)* in 1977 defined it as: "The practice of effecting an options transaction based upon non-public information regarding an impending block transaction[1] in the underlying stock, in order to obtain a profit when the options market adjusts to the price at which the block trades. [9]" Self-regulating exchanges (*e.g., CBoE*) and the *SEC* spent the ensuing years planning how to detect and outlaw front-running practices [138]. 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

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

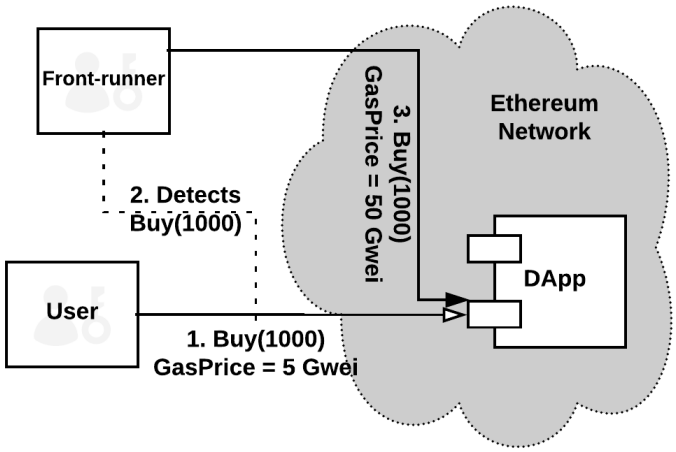
knowledge.[2]" The *CBoE* tried to educate their members on existing rules, however, differences in opinion regarding the unfairness of front-running activities, insufficient exchange rules and lack of a precise definition in this area resulted in no action [9] until the SEC began the regulation. We refer the reader interested in further details on this early regulatory history to Markham [138]. The first front-running policies applied only to certain option markets. In 2002, the rule was expanded to cover all security futures [3]. In 2012, it was expanded further with the new amendment, FINRA Rule 5270, to cover trading in options, derivatives, or other financial instruments overlying a security with only a few exceptions [6, 5]. Similar issues have been seen with domain names [4, 75] as well.

## 4.4 Background on Blockchain Front-running

Blockchain technology (introduced via Bitcoin in 2008 [157]) strives to disintermediate central parties that participate in a transaction. However, blockchains also introduce new participants in the process of relaying and finalizing transactions. Miners are in the best position to conduct these attacks as they hold fine-grained control over the exact set of transactions that will execute and in what order and can mix in their own (late) transactions without broadcasting them. Miners do however have to commit to what their own transactions will be before beginning the proof of work required to solve a block.

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

Figure 4.1: The front-runner upon spotting the profitable transaction *Buy(1000)* sends his own transaction with higher gas price to bribe the miners to prioritize his transaction over initial transaction.




Any user monitoring network transactions (*e.g.,* running a full node) can see unconfirmed transactions. On the Ethereum blockchain, users have to pay for the computations in a small amount of Ether called gas [1]. The price that users pay for transactions, gasPrice, can increase or decrease how quickly miners will execute them and include them within the blocks they mine. A profit-motivated miner who sees identical transactions with different transaction fees will prioritize the transaction that pays a higher gas price due to limited space in the blocks. This has been called a gas auction [108]. Therefore, any regular user who runs a full-node Ethereum client can front-run pending transactions by sending adaptive transactions with a higher gas price (see Figure 4.1).

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

receive them at all [100, 137].

## 4.5 Literature on Blockchain Front-running

Front-running is related to two well-studied concepts: double-spending and rushing adversaries [122]. Double-spending attacks in Bitcoin are related to front-running [20, 116]. In this attack, a user broadcasts a transaction and is able to obtain some off-blockchain good or service before the transaction has actually been (fully) confirmed. The user can then broadcast a competing transaction that sends the same unspent coins to herself, perhaps using higher transaction fees, arrangements with miners or artifacts of the network topology to have the second transaction confirmed instead of the first. This can be considered a form of self-front-running. In the cryptographic literature, front-running attacks are modeled by allowing a so called 'rushing' adversary to interact with the protocol [23]. In particular, ideal functionalities of blockchains (such as those used in simulation-based proofs) need to capture this adversarial capability, assuming the real blockchain does not address front-running. See *e.g.,* Bitcoin backbone [94] and Hawk [122].

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 [18]. They also propose a cryptographic approach, similar to commit and reveal (see Section 4.9.2) to prevent front-running.

Furthermore, Daian *et al.*, looks at the economically motivated front-running at-

tacks by introducing the notion of *Maximum Extractable Value (MEV)* [65]. MEV[3] is defined by actions that facilitates transaction reordering or front-running attacks to profit from the transactions in the mempool. Additionally, they run economical analysis on the MEV occurrences in the wild.

## 4.6 A Taxonomy of Front-running Attacks

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

In the first type of attack, a *displacement attack*, it is not important to the adversary for Alice's function call to run after Mallory runs her function. Alice's can be orphaned or run with no meaningful effect. Examples of displacement include: Alice trying to register a domain name and Mallory registering it first [114]; Alice trying to submit a bug to receive a bounty and Mallory stealing it and submitting it first [38]; and Alice trying to submit a bid in an auction and Mallory copying it.

In an *insertion attack*, after Mallory runs her function, the state of the contract is changed and she needs Alice's original function to run on this modified state. For

[3]Originally, MEV was centered around Miner Extractable Value. However, with the subsequent development of block builders, the scope of MEV broadened to encompass all network participants.

example, if Alice places a purchase order on a blockchain asset at a higher price than the best offer, Mallory will insert two transactions: she will purchase at the best offer price and then offer the same asset for sale at Alice's slightly higher purchase price. If Alice's transaction is then run after, Mallory will profit on the price difference without having to hold the asset.

In a *suppression attack*, after Mallory runs her function, she tries to delay Alice from running her function. After the delay, she is indifferent to whether Alice's function runs or not. We only observe this attack pattern in one DApp and the details are quite specific to it, so we defer discussion until Section 4.7.3.

Each of these attacks have two variants, *asymmetric* and *bulk*. In some cases, Alice and Mallory are performing different operations. For example, Alice is trying to cancel an offer, and Mallory is trying to fulfill it first. We call this *asymmetric displacement*. In other cases, Mallory is trying to run a large set of functions: for example Alice and others are trying to buy a limited set of shares offered by a firm on a blockchain. We call this *bulk displacement*.

Further more, Torres *et al.* [213] expands on this taxonomy and conducts an analysis of the profitability of each type of frontrunning attacks in the wild. In their research they show that suppression attack has the highest reward but also highest risk, while the displacement attack has no risk as it does not depend on the victims transaction. Insertion attack is the most popular type of attacks mainly on the decentralized exchanges.

Table 4.1: Top 25 DApps based on recent user activity from `DAppRadar.com` on September 4th, 2018. The DApps that are in bold are discussed in this paper.

| DApp Category | Names | Rank |
|---|---|---|
| Exchanges | IDEX | 1 |
| | **ForkDelta, EtherDelta** | 2 |
| | **Bancor** | 7 |
| | The Token Store | 13 |
| | LocalEthereum | 14 |
| | Kyber | 22 |
| | **0x Protocol** | 23 |
| Crypto-Collectible Games (ERC-721 [77]) | **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 |

## 4.7 Cases of Front-running in DApps

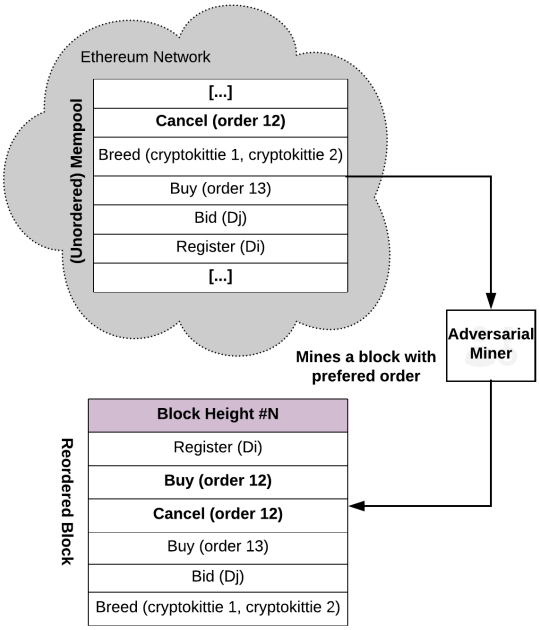
To find example DApps to study, we used the top 25 DApps based on recent user activity from `DAppradar.com` in September 2018.[4] User activity is admittedly an imperfect metric for finding the 'most significant' DApps: significant DApps might be lower volume overall or for extended periods of time (*e.g.,* ICOs, which we remedy

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

by studying independently in Section 4.8). However, user activity is an objective criteria, data on it is available, and the list captures our intuition about which DApps are significant. It suffices for a first study in this area, and is preferable over an ad hoc approach. Using the dataset, we categorized the top 25 applications into 4 principal use cases. The details are given in Table 4.1.

### 4.7.1 Markets and Exchanges

Figure 4.2: The adversarial miner monitors the Ethereum mempool for decentralized exchange transactions. Upon spotting a profitable cancellation transaction, he puts his buy order prior to the cancel transaction in the block he mines. Doing so, the miner can profit from the underlying trade and also get the gas included in the cancel transaction.




The first category of DApp in Table 4.1 are financial exchanges for trading ether and Ethereum-based tokens. Exchanges such as EtherDelta[5], purport to implement a decentralized exchange, however, their order books are stored on a central server

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

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 [224] uses *Relayers* which act as the order book holders and could front-run the orders they relay.

As seen in traditional financial markets, one method to manipulate the spot price of an asset, is to flood the market with orders and cancel them when there are filling orders ("taker's griefing" [57]). Placing an order in a partially centralized exchange is free, but to prevent taker's griefing attacks, the user needs to send an Ethereum transaction to cancel each of his orders. Cancelling orders is most important when prices change faster than order execution. In this case, when an adversarial actor sees a pending cancellation transaction, he sends a fill order transaction with higher gasPrice to get in front of the cancellation order and take the order before it is canceled (this is known as *cancellation grief*). This attack follows the asymmetric displacement template and is illustrated in Figure 4.2.

Designing truly decentralized exchanges, where the order book is implemented directly on a public blockchain, is being pursued by a number of projects [71]. These designs are generally vulnerable to front-running attacks following a displacement or insertion template. For example, a front-running full node or miner might gauge the demand for trades at a given price by the number of pending orders, and try to displace them at the same price assuming the demand is the result of the accurate new information about the asset. Alternatively, the front-runner might observe a large market order (*i.e.,* it will execute at any price). The adversary can try to insert

a pair of limit orders that will bid near the best offer price and offer at a higher price. If the pair executes ahead of the market order, the front-runner profits by scalping the price of the shares. Finally, if adversary has pre-existing offers likely to be reached by the market order, she could insert cancelations and new offers at a higher price.

Bancor is an exchange DApp that allows users to exchange their tokens without any counter-party risk. The protocol aims to solve the cryptocurrency liquidity issue by introducing *Smart Tokens* [105]. 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 favorable profits by buying before the order and fill the original order with slightly higher price. Researchers have shown and implemented a proof of concept code to front-run Bancor as a non-miner user [30].

### 4.7.2 Crypto-Collectibles Games

The second category of DApp in Table 4.1 is crypto-collectables. Consider Cryptokitties [199], the most active DApp in this category and third most active overall. Each kitty 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* [77]. Kitties are generally bought and sold on-chain through auction smart contracts. See Sections 4.7.1 and 4.7.4 for more details on auction-based front-running attacks.

Specific to Cryptokitties protocol, they can breed and give birth. When cryptokitties breed, the smart contract sets from which future block the pregnancy of the cat can be completed. Anyone can complete the pregnancy by calling `giveBirth()` after the birthing block and they will receive a reward in ether[6]. Even though front-running these calls would not affect the protocol workflow, but this displacement attack could result in financial profit for front-runners [177, 121].

### 4.7.3 Gambling

The third category of DApp in Table 4.1 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 [171]. Any candidate

[6]As there are no automated function calls in Ethereum, this incentive model –known as *Action Callback* [173]– is used to encourage users to call these functions.

source of randomness (such as block headers) is accessible to all DApp functions and can also be manipulated to an extent by miners.

Fomo3D is an example of a game style (known as Exit Scam[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 displace any new ticket purchases [16] that would reset the counter. The winner appears to have started by deploying many high gas consumption DApps unrelated to the game. When the timer of the game reached about 3 minutes, the winner bought 1 ticket and then sent multiple high gasPrice transactions to her own DApps. These transactions congested the network and bribed miners to prioritize them ahead of any new ticket purchases in Fomo3D. Recall this basic form of bribery is called a *Gas Auction*; See related work [141, 32] for more sophisticated bribery contracts.

We classify this in the unique category of a suppression attack in our taxonomy (see Section 4.6). At first glance, it seemed like an extreme version of an asymmetric/bulk displacement attack on any new ticket purchase transactions. However the

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

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

key difference is that the front-runner does not care at all about the execution of her transactions—if miners mined empty blocks for three minutes, that would also be acceptable. Thus, bulk displacement[9] is simply a means-to-an-end and not the actual end goal of the adversary.

### 4.7.4 Name Services

The final category in Table 4.1 is name services, which are primarily aimed at disintermediating central parties involved in web domain registration (*e.g.,* ICAAN and registrars) and resolution (*e.g.,* DNS). For simple name services (such as some academic work like Ghazal [153]), domains purchases are transactions and front-runners can displace other users attempting to register domains. This parallels front-running attacks seen in regular (non-blockchain) domain registration [4]. Ethereum Name Service (ENS) [112] is the most active naming service on Ethereum. Instead of allowing new `.eth` domain names to be purchased directly, they are put up for a sealed bid auction which seals the domain name in a bid, but not the bid amount. Most implementations use the more user friendly but less confidential method for starting and bidding on a domain name: `startAuctionsAndBid()`. This method leaks the hash of the domain and the initial bid amount in the auction. Original names can be guessed from the hashes (*e.g.,* rainbow tables, used in ENS Twitter bot[10]) or people can bid on domains even though they do not know what they are because of speculation on its value.

[9] Also known as Block Stuffing Attack [191]

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

Users are allowed to bid for 3 days before the 2-day reveal phase begins (see 4.9.2), in which all bidders (winners and losers) must send a transaction to reveal their bids for a specific domain or sacrifice their bid amount . Also note that if two bidders bid the same price, the first to reveal wins it [70]. Using the leaked information, the domain squatter can win the auction with the same price of the original bidder by revealing it first. This is similar to front-running as it relies on inserting an action before the user, however we do not consider this specific action as front-running attack.

## 4.8 Cases of Front-running in ICOs

Initial coin offerings (ICOs) have changed how blockchain firms raise capital. More than 3000 ICOs have been held on Ethereum, and the market capitalization of these tokens appears to exceed $75B USD in the first half of 2018 [235]. At the DApp level, tokens are offered in short-term sales that see high transaction activity while the sale is on-going and then the activity tapers off to occasional owner transfers. When we collected the top 25 most active DApps on `DAppRadar.com`, no significant ICOs were being sold. The ICO category slips through our sampling method, but we identify it as a major category of DApp and study it here.

### 4.8.1 *Status.im* ICO

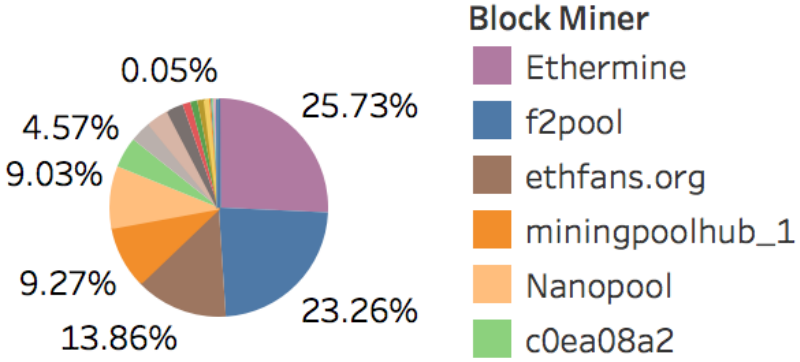
To deal with demand, ICOs cap sales in a variety of ways to mitigate front-running attacks. In June 2017, *Status.im* [200] started its ICO and reached the predefined cap within 16 hours, collecting close to 300,000 Ether. In order to prevent wealthy investors purchasing all the tokens and limit the 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 [170].

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

### 4.8.2 Data Collection and Analysis

According to the analysis we carried out, we discovered that the F2Pool—an Ethereum mining pool that had around 23% of the mining hash rate at the time (Figure 4.3)—

Figure 4.3: 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.




sent 100 Ether to 30 new Ethereum addresses before the Status.im ICO started. When the ICO opened, F2Pool constructed 31 transactions to the ICO smart contract from their addresses, without broadcasting the transactions to the network[11]. 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[12] for the full node, a python script for extracting, transforming and loading Ethereum blocks, named `ethereum-etl` [143] and Google BigQuery.[13] Using this software stack, we were able to isolate transactions within the Status.im ICO. We used data analysis tool

[11]Note that we do not have an authoritative copy of the mempool over time, however, the probability of these transactions being broadcasted to the network and exclusively get mined by the same pool as the sender is low.

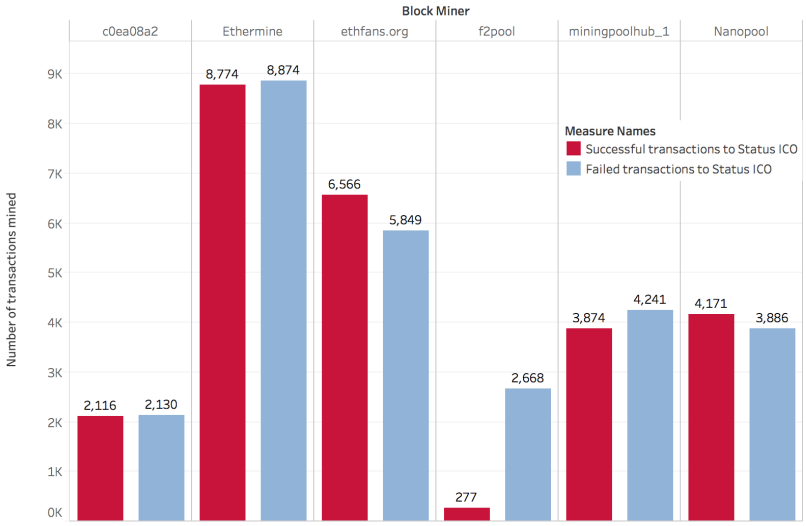
[12]Official Go implementation`https://github.com/ethereum/go-ethereum`.

[13]`https://cloud.google.com/bigquery/`

Tableau.[14] A copy of this dataset and the initial findings can be found in our Github repository[15].

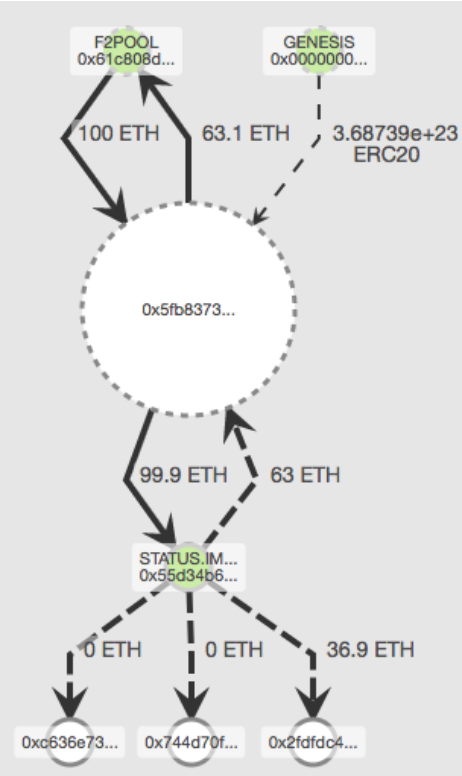
As shown in Figure 4.4, most of the top miners in the mentioned time frame, have mined almost the same number of failed and successful transactions which were directed toward Status.im token sale, however F2Pool's transactions indicate their successful transactions were equivalent to 10% of the failed 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 rejected and threw an exception and by *"successful transaction"* we mean the transactions that went through and received tokens from the smart contract.

Figure 4.4: This chart shows the miners behaviour on the time frame that Status.im ICO was running. It is clear that the number of successful transactions mined by F2Pool do not follow the random homogeneous pattern of the rest of the network.




[14] https://www.tableau.com/
[15] http://bit.ly/madibaFrontrunning

Figure 4.5: Prior to *Status.im* ICO *F2Pool* deposited 100 Ether in multiple new Ethereum addresses. On the time of the ICO, transactions sent from these addresses to *Status* ICO smart contract were prioritized in their mining pool, resulting in purchasing *ERC20* tokens. This method was used to overcome the dynamic ceiling algorithm of the ICO smart contract. Later on they sent the refunded Ether back to their own address.[17]




By tracing the transactions from these 30 addresses, we found explicit interference by F2Pool[18] in this scenario. As shown in Figure 4.5, the funds deposited by F2Pool in these addresses were sent to *Status.im* ICO and mined by F2Pool themselves, where the dynamic ceiling algorithm refunded a portion of the deposited funds. A few days after these funds were sent back to F2Pool main address and the tokens were aggregated later in one single address. Although this incident does not involve transaction

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

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

reordering in the blocks, it shows how miners can modify their mining software to behave in a certain way to front-run other transactions by *bulk displacement* to gain monetary profit.

## 4.9 Key Mitigations

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

We classify the mitigations into four main categories. In the first category, the blockchain removes the miner's ability to arbitrarily order transactions and tries to enforce some ordering, or queue, for the transactions. In the second category, cryptographic techniques are used to limit the visibility of transactions, giving the potential front-running less information to base their strategy on. In the third category, DApps are designed from the bottom-up to remove the importance of transaction ordering or time in their operations. We also note that for DApps that are legally well-formed (*e.g.,* with identified parties and a clear jurisdiction), front-running attacks can violate laws, which is its own deterrent. The idea for the forth category is to embrace the front-running opportunities and design methods to share the profit with more actors than just the block builders. We discuss each of these categories in more detail

below.

**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 [7]. As mentioned in section 4.3, 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 [239, 42] and sealed bids [179] 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.

### 4.9.1 Transaction Sequencing

Ethereum miners store pending transactions in pools and draw from them when forming blocks. As the term 'pool' implies, there is no intrinsic order to how transactions are drawn and miners are free to sequence them arbitrarily.[19] The vanilla Go-Ethereum (geth) implementation prioritizes transactions based on their gas price

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

and nonce [8]. Because no rule is enforced, miners can sequence transactions in advantageous ways. A number of proposals attempt to thwart this attack by enforcing a rule about how to sequence transactions.

First-in-first-out (FIFO) is generally not trivial to implement on a distributed network because transactions can reach different nodes in a different order. Kelker *et al.* proposes a solution this problem by adding a property to the consensus protocol – transaction-order-fairness– [118, 117]. Furthermore, A trusted third party can be used to assign sequential numbers to transactions (and sign them), but this is contrary to blockchain's core innovation of distributed trust. Nonetheless, most of the scaling layer two solutions use centralize sequencers and some exchanges do centralize time-sensitive functionalities (*e.g., EtherDelta* and *0xProject*) in off-chain order books [224, 223].

One alternative is to sequence transactions pseudorandomly. This can be seen in proposals like Canonical Transaction Ordering Rule (CTOR) by Bitcoin Cash ABC [218] which adds transactions in lexicographical order according to their hash [111]. Note that Bitcoin does not have a front-running problem for standard transactions. While this could be used by Ethereum to make front-running statistically difficult, the protection is marginal at best and might even exacerbate attacks. A front-runner can construct multiple equivalent transactions, with slightly different values, until she finds a candidate that positions her transaction a desirable location in the resulting sequence. She broadcasts only this transaction and now miners that include her transaction will position it in front of transactions they heard about much earlier.

Finally, transactions themselves could enforce order. For example, they could specify the current state of the contract as the only state to execute on. This transaction chaining only prevents certain types of front-running; *i.e.,* it prevents insertion attacks but not displacement attacks (recall our taxonomy in Section 4.6). As transaction chaining only allows one state-changing transaction per state, at most one of a set of concurrent transactions can be confirmed; a drawback for active DApps.

Da Silva *et al.* [63] 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. Additionally, Kelkar *et al.* [118] proposes a solution by introducing an additional property to the Byzantine consensus protocol, transaction order-fairness, and they analyze the assumptions necessary to realize fair ordering. Another solution is to create a centralized sequencer that orders that transactions, this is a common solution for many scaling Layer two solutions on Ethereum. This solution is not ideal to decentralized applications as it requires a trusted third party to order the transactions. In some cases such as Fomo3D, the fair ordering of the transaction can increase the likely hood of the hacker winning the game by using the similar suppression attack as described in 4.7.3.

### 4.9.2 Confidentiality

**Privacy-Preserving Blockchains.** All transaction details in Bitcoin are made public and participant identities are only lightly protected. A number of techniques increase confidentiality [39, 139] and anonymity [146, 163, 188] for cryptocurrencies. A current research direction is extending these protections to DApps [228, 181]. It is tempting to think that a confidential DApp would not permit front-running, as the front-runner would not know the details of the transaction she is front-running. However, there are some nuances here to explore.

A DApp interaction includes the following components: (1) the code of the DApp, (2) the current state of the DApp, (3) the name of the function being invoked, (4) the parameters supplied to the function, (5) the address of the contract the function is being invoked on, and (6) the identity of the sender. Confidentiality applied to a DApp could mean different levels of protection for each of these. For front-running, function calls (3,4) are the most important, however, function calls could be inferred from state changes (2). Hawk [122] and Ekiden [49] are examples of (2,3,4)-confidentiality (with limitations we are glossing over).

The applicability of privacy-preserving blockchains needs to be evaluated on a case-by-case basis. For example, one method used by traditional financial exchanges in dealing with front-running from high frequency traders is a dark pool: essentially a (2,3,4)-confidential order book maintained by a trusted party. A DApp could disintermediate this trusted party. Users whose balances are affected by changes in the contract's state would need to be able to learn this information. Further, if the con-

tract addresses are known (*i.e.,* no 5-confidentiality), front-runners can know about the traffic pattern of calls to contracts which could be sufficient grounds for attack; for example, if each asset on an exchange has its own market contract, this leaks trade volume information. As a contrasting example, consider again decentralized domain registration: hiding state changes (2-confidentiality) defeats the entire purpose of the DApp, and protecting function calls is ineffective with a public state change since the state itself reveals the domain being registered.

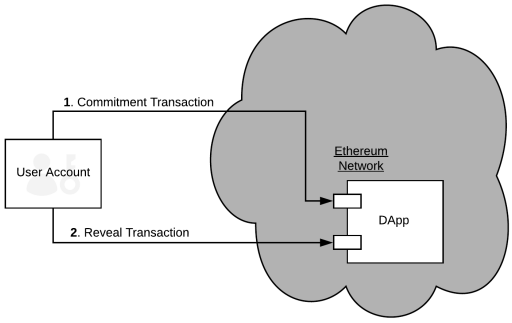
Privacy-preserving blockchains strive (to some extent [148, 115]) to keep all details of transactions private, including participants and amounts. As an example, ZCash [106] uses two distinct types of addresses, transparent and shielded addresses. Transparent addresses work similar to Bitcoin transactions, 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. 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.

**Commit/Reveal.** While confidentiality appears insufficient for solving domain name front-running alone, a hybrid approach of sequencing and confidentiality can be effective and is, in fact, an example of an older cryptographic trick known as commit/reveal. The essence of the approach is to protect the function call (*e.g.,* (3,4)-

or (4)-confidentiality) until the function is enqueued in a sequence of functions to be executed. Once the sequence is established, the confidentiality is lifted and the function can only be executed in the order it was enqueued (or, generally speaking, not at all).

Recall that a commitment scheme enables one to commit to a digital value (*e.g.,* a statement, transaction, data, *etc.*) while keeping it a secret (*hiding*), and then open it (and only it: *binding*) at a later time of the committer's choosing [33]. A common approach (conjectured to be hiding) is to submit the cryptographic hash of the value with a random nonce (for low entropy data) to a smart contract, and later reveal the original value and nonce which can be verified by the contract to correctly hash to the commitment (see Figure 4.6).

Figure 4.6: 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.



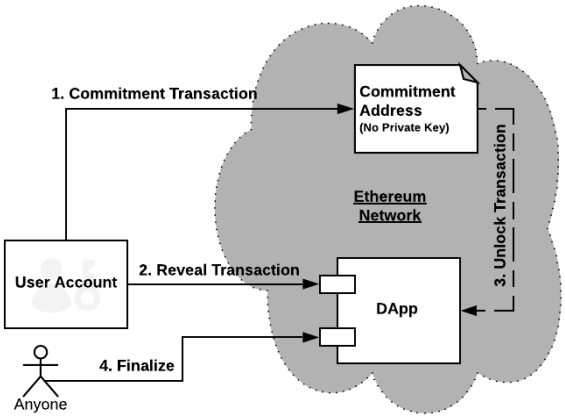

An early application of this scheme to blockchain is Namecoin, a Bitcoin-forked DApp for name services [114]. In Namecoin, a user sends a commit transaction which registers a new hidden domain name, similar to a sealed bid. Once this first

transaction is confirmed, a time delay begins. After the delay, a second transaction reveals the details of the requested domain. This prevents front-running if the reveal transaction is confirmed faster than an adversarial node or miner can redo the entire process.

Commit/reveal is a two-round protocol, and aborting after the first round (early aborts) could be an issue for this (along with most multi-round cryptographic protocols). For example, in a financial exchange where the number of other orders might be in a predictable interval, an adversary can spray the sequence (*i.e.,* a price-time priority queue) with multiple committed transactions and no intention of executing them all. She then only reveal the ones that result in an advantageous trade.[20] There are other ways of aborting; if payments are required but not collateralized, the aborting party can ensure that payment is not available for transfer. One mitigation to early aborts that blockchain is uniquely positioned to make is having users post a fidelity bond of a certain amount of cryptocurrency that can be automatically dispensed if they fail to fully execute committed transactions (this is used in multi-round blockchain voting [142]). Finally, we note that any multiple round protocol will have usability challenges: users must be aware that participating in the first round is not sufficient for completing their intention.

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

Figure 4.7: Submarine Send [37]. User generates an *Unlock* transaction from which the commitment address is retrieved using ECDSA ECRecover. 1. by funding the *commitment address*,user is committed to the transaction. 2. User sends the *reveal transaction* to the DApp, revealing the nature of the commitment transaction. 3. She broadcasts the *unlock transaction* to unlock the funds in the commitment address. 4. After the *"Auction"* is over, anyone can call *Finalize* function to finalize the process.




**Enhanced Commit/Reveal.** Submarine Commitments [37, 36] extend the confidentiality of the commit and reveal, so that the commitment transaction is identical to a transaction to a newly generated Ethereum address. They initially hide the contract address being invoked, providing (3,4,5)-confidentiality during the commit phase; and they ensure that if a revealed transaction sent funds, the funds were fully collateralized at commit time and are available to the receiving smart contract. See Figure 4.7.

### 4.9.3 Design Practices

The final main category of mitigation is to assume front-running is unpreventable and to thus responsively redesign the functionality of the DApp to remove any benefit from it. For example, when designing a decentralized exchange, one can use a call market design instead of a time-sensitive order book [50] to side-step and disincentivize front-

running. In a call market design, the arrival time of orders does not matter as they are executed in batches[21] . The call market solution pivots profitable gains that front-running miners stand to gain into fees that they collect [50], removing the financial incentive to front-run.

In the finance literature, Malinova and Park discuss front-running mitigations for blockchain-based trading platforms [135]. Instead of studying DApps, they develop an economic model where transactions, asset holdings, and traders' identities have greater transparency than in standard economic models—transparency they argue that could be accomplished by blockchain technology. However, in their model, they assume entities can interact directly over private channels to arrange trades. They define front-running in the context of private offers, where parties might adjust their position before accepting or countering a received offer. This model is quite different than the DApp-based model we study here.

Another example in the design of ERC20 standard [219] is the allowance functionality. *approve()* function in the specification allows a second entity to be able to spend N tokens from the sender's balance. In order to change the allowance, sender must send a transaction to set the new allowance value. Using the insertion attack, attacker could front-run the new allowance transaction and spend the old value before the new value is set [180, 110], and then additionally spend the new amount at a later time. Solutions such as *decreaseApproval()/increaseApproval()* were added in updated implementations.

[21] Also known as batch auctions [221]

Baum *et al.*, expands on many of the mitigation approaches described in this section by modeling the front-running mitigation methods in decentralized finance and discuss further research directions and challenges [21]. Additionally, Heimbach *et al.* systemizes the transaction reordering manipulation mitigation schemes and analyze the full impact on the blockchain [102] and they determine that, at present, no strategy entirely satisfies all the requirements of the blockchain ecosystem.

### 4.9.4 Embracing Front-running.

This approach has been initially advocated by a group called Flashbots as to remove the negative externalities posed by Maximal Extractable Value (MEV) [89]. MEV is the profit that miners can make by reordering transactions in a block. They propose a new type of block building mechanism, called *Flashbots Bundle*. A bundle is a set of transactions that are bundled together and submitted to the *relayer*. The bundle is not broadcasted to the network and is only visible to the relayer, in which is later shared with block proposer and builder. The block builder can then decide to include the bundle in the block or not. This method is not a solution to front-running, but it is a way to share the profit with more actors than just the block builders. Any entity can create a transaction bundle which includes their own transactions and the transactions they are front-running in the preferred order that is profitable for them. They submit this bundle to a relayer in which a portion of the profit is offered to the block builder to include the bundle in their block. This solution offers more actors to participate in MEV and share the profit –democratizing MEV–. Even though

this solution does not solve the issue, it changes the dynamic of the transaction propagation and block building, and potentially reduces the network congestion load and gas price for the overall network [140].

## 4.10 Concluding Remarks

Front-running is a pervasive issue in Ethereum DApps. DApp developers don't necessarily have the mindset to design DApps with front-running in mind. This is an attempt to bring forward the subject and increase awareness of these type of attacks. 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.

### 4.10.1 Ethics and Legality

The underlying technology of Ethereum is not designed to prevent front-running attacks. However, many researchers and industry leaders have been discussing the issue and possible solutions for a while. As we discussed in 4.3, in traditional markets, the idea of front-running orders took more than 3 decades to fully understand and implement legal solutions to prevent it. When it comes to blockchain, the technology is still new and people are still figuring out all the possible attacks and how to stop them. This research does not dive deep into the ethical problems of MEV, but it does bring up a big question: in a system where people are paid to act honestly, MEV

lets them seek extra profits and gives them similar incentives. For example, MEV can affect many users by changing the order price after a user sends a transaction but before it goes through. In traditional rules, this is seen as a kind of market manipulation or insider trading and is illegal. So, it's important to think about how these actions and rules apply in the blockchain world.

# Chapter 5

# Oracles; from the ground truth to market manipulation

*This chapter is based on the paper "Oracles; from the ground truth to market manipulation" published at 3rd ACM Conference on Advances in Financial Technologies [83]. This paper was supervised by Jeremy Clark and co-authored by Mehdi Salehi, Wanyun Catherine Gu.*

## 5.1 Background

With billions of dollars at stake, decentralized networks are prone to attacks. It is essential that the smart contracts, which govern how systems are run on these networks, are executed correctly. Public blockchains, like Ethereum, ensure the correct execution of smart contract code by taking the consensus of a large, open network

of nodes operating the Ethereum software. For consensus to form, many nodes need to make decisions based on the exact same input data. Hypothetically, if a decision requires nodes to fetch data or use a service provider outside of the blockchain, there can be no guarantee that every node in a global network has the same access and view of this external source. For this reason, blockchains only execute on internal sources: data and code provided in a current transaction, or past data and code already stored on the blockchain.

Many potential decentralized applications seem very natural until the designer hits the 'oracle problem' and realizes an interface to the external world is required. An oracle is a solution to this problem. It is a service that feeds off-chain data into on-chain storage. The trust model of oracles vary—some data comes with cryptographic certification while other data is assumed to be true based on trusting the oracle, or a set of oracles. Oracle-supplied data cannot easily be changed or removed once finalized on-chain, allowing disputes over data accuracy to be based on a public record. Leveraging this immutability is one approach to incentivizing oracles to post truthful information.

We aim to construct in this report a systematization of knowledge (SoK) of implementation choices for oracles, facilitated by breaking down the operation of an oracle into a set of modules. For each module, we explore potential system vulnerabilities and discuss attack vectors. We also aim to categorize all the significant oracle proposals of different projects within a taxonomy we propose. The goal of this research is to help the reader better understand the system design for oracles across different

use cases and implementations.

## 5.2 Preliminaries

Ethereum [230] is a prominent public blockchain with the largest developer headcount. While oracles are applicable to any blockchain, we will adopt Ethereum as a concrete example of a blockchain for the purposes of explaining each concept in this proposal. Ethereum is inspired by Bitcoin but adds a verbose language for programming *smart contracts* that execute on the `Ethereum Virtual Machine (EVM)`. All transactions and executions are verified by a decentralized network of nodes. Solidity is the main high-level programming language used by developers for developing smart contracts and decentralized applications (DApps). Smart contracts are small code bases that live on a blockchain. In short, smart contracts can be seen as blackbox applications that get inputs from a user and follow the code flow to the output, which can update the state of the contract and trigger monetary transactions.

### 5.2.1 The Oracle Problem.

Smart contracts cannot access external resources (*e.g.,* a website or an online database) to fetch data that resides outside of the blockchain (*e.g.,* a price quote of an asset). External data needs to be relayed to smart contracts with an oracle. An *oracle* is a bridge or gateway that connects the off-chain real world knowledge and the on-chain blockchain network. The 'oracle problem' [45] describes the limitation with which the

types of applications that can execute solely within a fully decentralized, adversarial environment like Ethereum. Generally speaking, a public blockchain environment is chosen to avoid dependencies on a single (or a small set) of trusted parties. One of the first oracle implementations used a smart contract in the form of a database (*i.e.,* mapping[1]) and was updated by a trusted entity known as the `owner`. More modern oracle updating methods use consensus protocol with multiple data feeds or polling techniques based on the 'wisdom of the crowd'. The data reported by an oracle will always introduce a time lag from the data source and more complex polling methods generally imply longer latency.

### 5.2.2 Trusted Third Parties.

A natural question for smart contract developers to ask is: if you trust the oracle, why not just have it compute everything? How is trusting the oracle, different from trusting a third party to provide the information? There are a few answers to this question: (1) there may be benefits to minimizing the centralized trust (*i.e.,* to just providing data instead of full execution), (2) there are widely trusted organizations and institutes—convincing one to operate an oracle service is a much lower technical ask than convincing one to operate a complete platform, and (3) if a data source becomes untrustworthy, it may require less effort to switch oracles than to redeploy the system. These points are not meant to be exhaustive, but rather to illustrate that there are many reasons why a developer may choose to implement their own oracle

[1] A Solidity `mapping` is simply a key-value database stored on a smart contract.

solution or distribute trust by using an oracle network instead of one trusted third party.

### 5.2.3 Methodology.

We found papers and other resources by examining the proceedings of top ranked security, cryptography, and blockchain venues; attending blockchain-focused community events; and leveraging our expertise and experience. Our inputs include academic papers, industry whitepapers, blog and social media posts, and talks at industry conferences on blockchain technology, Ethereum, and decentralized finance (DeFi).

### 5.2.4 Oracle Use-Cases.

Oracles have been proposed for a wide variety of applications. Based on our reading, most of the use-cases fall into one of the main categories below.

- **Stablecoins** [51, 149, 168, 97, 134] and **synthetic assets** [186] require the exchange rate between the asset they are price-targeting and the price of an on-chain source of collateral.

- **Derivatives** [80, 26, 196] and **prediction markets** [50, 169] require external prices or event outcomes to settle on-chain contracts.

- **Provenance systems** [183, 211] require tracking information of real world assets like gold, diamonds, mechanical parts, and shipments.

- **Identity** [119, 136] and other on-chain reputation systems require knowledge of governmental records to establish identities.

- **Randomness** [44] can only be produced deterministically on a blockchain. In order to use any non-deterministic random number, an external oracle is needed to feed the randomness into the smart contract. **Lotteries** [174] and **games** [85] are examples. Additionally, cryptographic tools like verifiable random functions (VRF) [144, 95] and verifiable delay functions (VDFs) [40, 31] can mitigate, respectively, any predictability or manipulability in the randomness.

- **Decentralized exchanges** can use prices from an external oracles to set parameters. On-chain market makers [105] uses such prices to minimize the deviation from the external market prices and tailor the pricing function. Additionally, some use oracles to provide sufficient liquidity near the mid-market price for more efficient automated market making [72, 233, 178].

- **Dynamic non-fungible tokens (NFTs)** [29] are crypto-collectables that can be minted, burned, or updated based on external data. For example, sports trading cards which depends on the real-time performance of a player.

## 5.3 Modular Work Flow

For our main contribution, we deconstruct how an oracle operates into several modules that generally operate sequentially (but in some solutions, certain steps are skipped)

and then we study each module one-by-one. An overview of the work flow is as follows:

5.3.1 **Ground Truth**: The goal of the oracle system is to relay the ground truth (*i.e.,* the real true data) to the requester of the data.

5.3.2 **Data Sources**: Data Sources are entities that store or measure a representation of the ground truth. There are a diverse set of data sources: databases, hardware sensors, humans, other smart contracts, *etc.*

5.3.3 **Data Feeders**: Data feeders report off-chain data sources to an on-chain oracle system. In order to incentivize truthful data reporting, an oracle system can introduce a mechanism to select data feeders from a collection of available data providers. The incentive mechanism can be collateral-based, such as staking, or reputation-based to find a reliable set of data feeders for each round of selection.

5.3.4 **Selection of Data Feeders**: The process of determining which data feeders should be used in an oracle system can be categorized into two main types: centralized and decentralized selection.

5.3.5 **Aggregation**: When data is submitted by multiple data feeders, the final representation of the data is an aggregation of each data feeder's input. The aggregation method can be random selection or algorithmic rule-based, such as using weighted average (the mean) or majority opinion. The design of the aggregation method is one of the most important aspects of an oracle system, as

intentional manipulation or unintentional errors during the aggregation process can result in untruthful data reporting by the oracle system.

5.3.6 **Dispute Phase**: Some oracle designs allow for a dispute phase as a counter-measure to oracle manipulation. The dispute phase might correct submitted data or punish untruthful data feeders. The dispute phase might also introduce further latency.

The steps above are visualized in Figure 5.1. Next we dive deeper into the modular workflow by trying to further define each module. As appropriate, we also discuss feasible attacks on the modules and possible mitigation measures.

### 5.3.1 Ground Truth

While not a module itself, ground truth is the initial input to an oracle system. Oracle designers cannot solve basic philosophical questions like *what is truth?* However it has to be understood (i) what the data actually represents and (ii) if it is reliable. Data is sometimes sensitive to small details. Consider a volatility statistic for a financial asset: basics like which volatility measure is being used over what precise time period are obvious, but smaller things like the tick size of the market generating the prices could be relevant [88]. When data is aggregated from multiple sources, minor differences in what is being represented (called *semantic heterogeneity*) can lead to deviations between values [133, 231, 98].

While oracle systems will attempt to solve the issue of malicious participants who

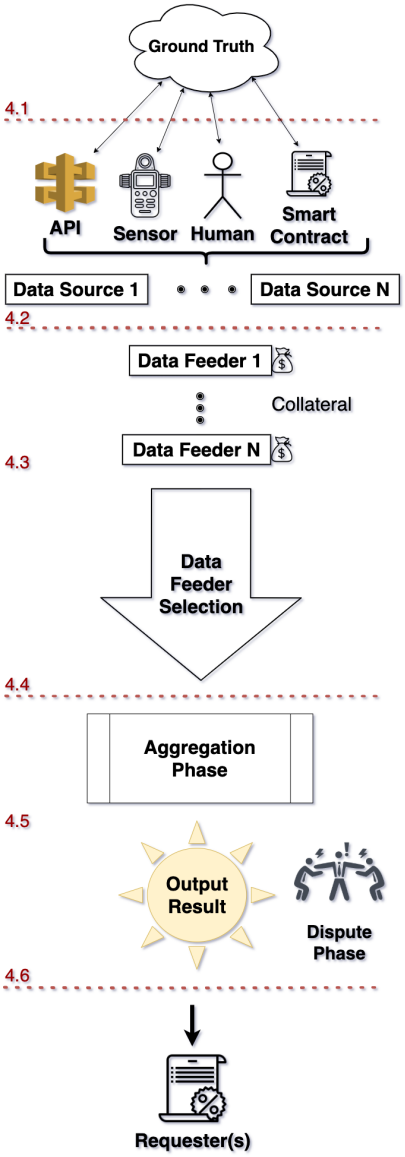



Figure 5.1: A visualization of our oracle workflow

mis-report the ground truth, it does not address the fundamental question of whether the ground truth itself is reliable. Some philosophers argue truth is observed, and observations require a 'web of beliefs' that is subject to error (for its consequences in security, see [104]). Reliability is judged by the assumptions made about the data source, described next.

### 5.3.2 Data Sources

Data Sources are defined here as passive entities that store and measure the representation of the ground truth. Common types of data sources include `databases`, `sensors`, `humans`, `smart contracts`, or a combination of them. Depending on how data sources gather and retrieve the ground truth, different attack types arrise. Using a hybrid of data sources (if possible) could reduce the reliability on a single point of input. We describe each common type and their security considerations.

**Humans.**

A human may provide the requested data, either by direct observation or by indirectly relaying data from another data source. Humans are prone to errors which is the main risk of this data source. Human errors include how the data is retrieved, how the data collector interprets the truth, and if data is relayed from a reliable source. Researchers have categorized human errors into the following three types (from least to most probable): very simple tasks, routine tasks, and complicated non-routine tasks [131]. An example for each category is, respectably, reading Bitcoin's exchange rate from an unverified source, inputting the data into the system, and configuring the oracle system.

Humans may also act maliciously and deliberately report wrong data when they perceive it will benefit them. As we will see in further modules, a robust oracle system will use incentives and disputes to promote truthful statements.

**Sensors.**

Sensors are electronic devices that collect raw data from the outside world and make it available to other devices. The data source may use more than one sensor to obtain the desired data. One example from traditional finance is the weather derivative, first introduced by the Chicago Mercantile Exchange (CME) [154]. These instruments use weather data provided by trusted institutions, such as the National Climate Data Center,[2] which collects weather data through a network of sensors.

Provenance is a highly cited application of blockchain, where products are tagged and traced through out the supply chain, including transportation, for management and/or certification [211, 150, 234]. The tags could be visual (barcodes) or electronic (RFID). A host of attacks on RFID have been proposed outside of blockchain oracles [14]. Blockchain technology does not solve some important trust issues: ensuring the proper tag is affixed to the proper product, each product has one tag, each tag is affixed to only one product, and tags cannot be transferred between products. This is called the *stapling* problem [183].

Sensors can produce noisy data or malfunction. The hardware of a sensor can also be modified when remote or physical access is unauthenticated (or weakly authenticated as many sensors are constrained devices). Probably the highest profile sensor attack (outside of blockchain) is Stuxnet [127]—malware that manipulated the vibration sensors, the valve control sensors, and the rotor speed sensors of Iran's nuclear centrifuges, causing the system to quietly fail [126].

[2] `https://www.ncdc.noaa.gov/`

**Databases and application programming interfaces (APIs).**

The most common mechanism used by software to fetch data is to use an API to obtain the data directly from a centralized database. A database is a set of tables that collect system events, while the API is an interface with the database. For example, a financial exchange keeps track of information in a database about every trade that has been executed. A data source that needs the daily traded volume of an asset could use the appropriate API of the exchange's database to extract the data from the related table in the database.

An active attacker can attack the system from two points. Modifying the data at rest in the database, or modifying the data in transit before and after the API call.

**Smart Contracts.**

Smart contract could be used as a data source similar to a database. Decentralized finance (DeFi) applications on Ethereum include decentralized exchange services like Uniswap [11], or other oracles that operate on-chain. For instance API3 oracle [24] uses other on-chain oracles, called *dAPIs*, as their data source. These oracles are whitelisted through voting by API3 token holders.

*Automated Market Makers* (AMMs) [222] are an on-chain alternative to centralized exchanges. Liquidity providers collateralize the contract with an equally valued volume of two types of cryptoassets. A mathematical rule governs how many assets of the one type are needed to purchase assets of the other. A well-known example of such mathematical rule is the *Constant Function Market Makers* (CFMM) to cal-

culate the exchange rates of tokens in a single trade [189]. The idea behind AMM was first raised by Hanson's logarithmic market scoring rule (LMSR) for prediction markets [99]. A class of DeFi projects (*e.g.,* Uniswap [11, 12] and Balancer [19]) uses CFMM to automate their market-making process. One of the utilizations of AMM is the ability to measure the price of an asset in a fully decentralized way, which addresses the *pricing oracle problem* [15].

One potential attack vector to the auto price discovery mechanism in an AMM is to manipulate prices provided by an algorithm, since the algorithmic rules used by an AMM is written in the smart contract and therefore how prices are quoted by the AMM can be calculated in advance. One real case example on *bZx* is described in Section 5.4.3. In addition to market manipulation, *smart contract vulnerabilities* [17, 48] could possibly be used to influence the data coming from the oracle, which we will discuss more in section 5.4.3.

### 5.3.3 Data Feeders

Data feeders are entities who gather and report the data from a data source (Section 5.3.2) to the oracle system. A common configuration consists of an *external feeder* which draws from off-chain data sources and deposit the data to an on-chain module. In case the data source is already on the blockchain, the data feeder step can be skipped.

It is not common to assume data feeders are fully honest, however a variety of threat models exist. Generally, this module will not attempt to determine if the data

has been falsified (the later sections data selection (Section 5.3.4), data aggregation (Section 5.3.5) and dispute phase (Section 5.3.6) modules will deal with this issue); rather it will consist of tunnelling the data through the feeder with some useful security provisions. We discuss most important security provisions to achieve data integrity, confidentiality, and non-repudiation on any specific data.

**Source Authentication.**

Data integrity can be enhanced by authenticating the source of the data and ensuring message integrity is preserved. It is sufficient to have the source sign the data, assuming the source's true signature verification (*i.e.,* public) key is known to the recipient of the data. This is most appropriate for sources like humans and sensors (although sensors may use a lightweight cryptographic alternative to expensive digital signatures [187]).

Databases, websites, and APIs typically support many cryptographic protocols, including the popular HTTPS (HTTP over SSL/TLS) which adds server authentication and message integrity to HTTP data [52]. However HTTPS alone is typically not sufficient, as the message integrity it provides can only be verified by a client that connects to the server and engages in an interactive handshake protocol. This client cannot, for example, produce a transcript of what occurred and show it to a third party (*e.g.,* a smart contract on Ethereum) as proof that the message was not modified. To turn HTTPS data into signed data (or something similar), a trusted third party can vouch that the data is as received. TLS notary [212] and DECO [237]

offer solutions that attest for the authenticity of HTTPS data. Town Crier [236] uses Trusted Execution Environments (TEE) like Intel SGX [59] to push the trust assumption onto TEE technology and, ultimately, the chip manufacturer.

**Confidentiality.**

For many smart contracts that rely on oracles, the final data is made transparent (*e.g.,* prices, weather, event outcomes). In a few cases, oracles feed data that is private (*e.g.,* identities, supply chain information) and the contract enforces an access structure of which entities under which circumstances can access it [136].

Confidentiality might also be temporary. Given the fact that information submitted to the mempool is public, there is a natural risk on the oracle system that a data feeder uses another data feeder's information to self-report to the system. This form of collusion between data feeders is called *mirroring attack [76]* in computer security literature. The data feeders are willing to freeload another data feeder's response to minimize their cost of data provision. They will also be confident that their data will not be an outlier and be penalized. To mitigate the risk of mirroring attacks, the oracle designer should consider mechanisms that ensure the confidentiality of the data sent by the data feeders. A popular technique to achieve confidentiality is to use a commitment scheme [33]. In a commitment scheme, each data feeder should send a commitment of the plain data as an encrypted message to the receiver. Later, the sender can reveal the original plain data and verify its authenticity using the commitment.

| *Category* | *Example* | No Trusted Third Party | Low latency | Resilient to Sybil Attacks | Resilient to Targetted DoS Attacks | Incentives are Endogenous |
|---|---|---|---|---|---|---|
| Centralized | `Maker V1 Oracle` | | • | • | | |
| Voting | `Maker V2 Oracle` | • | | | • | |
| Staking | `Chainlink`, `ASTRAEA` | • | • | ○ | • | • |

Table 5.1: Evaluation Framework on selection of data feeders. For details see Section 5.3.4

**Non-Repudiation.**

A non-repudiation mechanism assures that a party cannot deny the sender's proposal after being submitted to the system. Oracle systems might rely on cryptographic signature schemes to eliminate the risk of in-transit corruption and to create irrefutable evidence of the data being provided by a source, for use in the dispute phase (Section 5.3.6) as needed.

### 5.3.4 Selection of Data Feeders

In order to ensure correct data is fed into the system, the design must select legitimate data feeders and weed out less qualified and malicious participants. In a non-adversarial environment, the design might aggregate all the incoming data without any selection, skipping this step.

The earliest designs for oracle systems, such as Oraclizeit [25] and PriceGeth [80],

were designed using just one single data feeder; however, to improve data quality and the degree of decentralization, more complex oracle systems such as ChainLink [76] involves selecting qualified data feeders to aggregate an output that is expected to be more representative of the ground truth.

This process can be categorized into two main types: centralized and decentralized selection, with decentralized selection having multiple approaches through voting and staking. Centralized selection and decentralized selection through voting, create an allowlist of legitimate data feeders, in contrast to selecting based on the algorithmic criteria in decentralized selection through staking.

**Centralized (Allowlist) Selection.**

A centralized selection is a permissioned approach where a centralized entity selects a number of data feeders directly without the involvement of other participants in the network. A centralized selection is analogous to having an allowlist for authorized data feeds (*e.g.,* Maker Oracle V1 [134]). Compared to a decentralized approach, centralized selection is fast and direct. However the trust footprint on the central entity is large: it must solely select legitimate data feeders and also have high availability to update the allowlist as needed.

**Decentralized (Allowlist) Selection through Voting.**

By decentralizing the selection process, the goal is to distribute the trust from a single entity to a collective decentralized governance. Voting distributes trust and

provides a degree of robustness against entities failing to participate, however it adds latency and introduces the threat that an actor can accumulate voting rights to sway the vote [132], or even to do distrust and destroy the system (*e.g.,* Goldfinger attack [123]).

For instance, in Maker V2 oracle [134], the selection of the data feeders is done through a decentralized governance process [97]. MKR[3] token holders vote on the number of authorized data feeders and who these data feeders can be [54].

Note that sometimes voting processes can provide the illusion of decentralization while not being much different than a centralized process in practice. To illustrate, consider a project with a governance token, in which most tokens are held by a few individuals where the project leaders advocate for their preferences and there is no established venue for dissenting opinions. If voters only inform themselves from one source of information, that source becomes a de facto centralized decision maker.

**Decentralized Selection through Staking.**

Like voting, staking attempts to utilize a token to align the incentives of the participants with the current functioning of the system. Mechanically, it works different: data feeders post collateral against the data they provide. In the dispute phase 5.3.6, any malicious data feeders will be punished by losing a portion or all of their collateral (called *slashing*). Even without slashing, the collateral amount acts as a barrier to entry for participants and rate-limits participant.

[3]MakerDAO Governance Token

The stake can be both in token value and reputation of the data feeder. As an example, in Chainlink [76] protocol has a reputation contract that keeps track of the accuracy of data reporting of different feeders. The `ExplicitStaking` module in Chainlink 2.0 defines the number of Link tokens each oracle node must stake to become a data feeder, while the service agreement of the Chainlink oracle defines the circumstances in which a node's stake will be slashed [166]. Put together, the incentives for selected data feeders to act honestly are avoiding reputational loss, avoiding loss of stake and penalty fees, and maintaining good standing for future income. In terms of selection, the data selection module forms a leaderboard, based on collateral and reputation, to select the highest ranked data feeders from all available feeders.

Another approach, introduced by ASTRAEA [13], uses a combination of game theory and collateralization between different actors in the system (Voters and Certifiers) to achieve equilibrium on what the final data should be.

A staking-based selection module avoids a central trusted third party, but it can add latency for adding/remove data feeds and other adjustments. It is also open to sybil attacks by design, while working to ensure these attacks have a significant cost for the adversary.

One challenge for designing a staking mechanism is setting a high enough punishment (slashing) mechanism to thwart malicious actions. Projects like UMA [214], another smart contract oracle design, dynamically adjust staked collateral needed for each round to ensure that *Cost of Corruption* (CoC) is higher than the pro-

jected *Profit from Corruption* (PfC). Profit from Corruption is defined by the data requester, in which UMA contracts require higher collateral to finalize the data from the data feeders. It is also important that participants are incentivized to file correct disputes—ones that will ultimately lead to identifying misbehaviour. If disputes are filed on-chain, the disputer will have to pay gas costs that need to be ultimately reimbursed by the resolution process.

Decentralized selection is done by the holders of some scarce token, typically a governance token specific to the oracle service. The simplest decentralized mechanism to hold a vote amongst token holders, who are indirectly incentivized (we call this an *exogenous incentive*) cast informed votes since they hold a token tied to the success of the system (*e.g.,* TruthCoin [197]). In a staking system, token holders are directly incentivized (a *endogenous incentive*) to vote 'correctly' (this remains to be defined but assume for now it means they vote in a way that will not be disputed) by posting some amount of their tokens as a fidelity bond. Stakers stand to be rewarded with new tokens and/or penalized (collateral slashed) depending on the performance of the data feeders they vote for.

Additionally a protocol could introduce a random selection within the data feeders to decrease the chance of sybil attacks. As an example Band Protocol [175], choses a random validator from top 100 staked participants for their oracle system.

Another approach used by Tellor oracle [201] is a simple Proof of Work (PoW) algorithm for each round of data. The first 5 miners to submit their desired data alongside the solution to the mining puzzle are selected as the data feeders of the

round. The selection is based on the hash power of each data feeder and randomness nature of proof of work consensus.

**Evaluation Framework on the selection of data feeders**

To compare designs for data feeder selection, we provide an evaluation framework. The definition of each evaluation criteria (*i.e.,* column of the table) follows, specifying what it means to receive a full dot (•), partial dot (◦) or to not receive a dot.

*No Trusted Third Party. A selection process that is distributed or decentralized among several equally-powerful entities earns a full dot (•). A process that relies on a single entity for critical functions is not awarded a dot.*

The voting and staking processes are decentralized amongst multiple token holders (•). As the name implies, the centralized process uses a trusted third party (no dot).

*Low Latency. A selection process that can move from proposal to finality within a single transaction is awarded a full dot (•). A process that requires multiple rounds of communication or communication among several entities is not awarded a dot.*

The centralized process can make selection decisions unilaterally (•). The voting process involves a round of communication with all of the participants (no dot). The staking process draws feeders unilaterally from an established leaderboard (•).

*Resilient to Sybil Attacks. A selection process that only allows unique feeders to participate is awarded a full dot (•). The evaluation does not consider what specific method is used to determine entities are unique but assumes it works reasonably well (not strictly infallible). A process that is open to multiple fake feeders controlled by*

*the same adversary is awarded a partial dot (∘) if each additional feeder created by the adversary has a material financial cost. If there is no material cost to creating additional fake feeders, the process receives no dot.*

The centralized process manages an allowlist based on real world reputations. We assume this reasonably prevents sybils (•). The staking process admits sybils but deters them by requiring staked tokens for each, which is costly (∘). The voting process does not deter sybils from entering the election but relies instead on the voting process to not select them (no dot).

*Resilient to Targeted Denial of Service Attacks. A selection process that only halts when multiple entities to go offline or fail is awarded a full dot (•). If critical functionalities cannot be performed with the failure of a single entity, but the basic selection process can proceed, it is awarded a partial dot (∘). If the process can be fully halted by the failure of a single entity, it is awarded no dot.*

The voting and staking processes can proceed until enough honest participants fail that a dishonest majority remains (•). By contrast, a failure with the central entity in a centralized process can prevent critical functionalities, like updating the allowlist (∘).

*Incentives are Endogenous. Every selection process should have the ability to remove untruthful feeders. Some selection processes might go beyond this and incentivize feeders to provide truthful information. Processes are awarded a full dot (•) if the awards/punishments can be realized by the selection process itself. If the selection process relies only on external incentives (e.g., damage to reputation), it is awarded*

*no dot. The evaluation does not consider how information is determined to be truthful or not. Endogenous means the design is simpler but does not imply it is more secure (cf. [91]).*

The staking process requires feeders to post collateral that can be taken (*i.e.,* slashed) for malicious behavior (•). Centralized and voting processes do not use internal incentives (no dot).

### 5.3.5 Aggregation of Data Feeds

Aggregation is the process of synthesizing the selected data feeds into one single output. The quality of the output depends on the data feed selection (see Section 5.3.4) and the aggregation process used. To highlight the importance of designing an aggregation method correctly, consider the case of Synthetix, a trading platform [196] that used the *average* (or *mean*) of two data feeders as their aggregation method. An attacker leveraged this to manipulate one of the two feeders by inflating the real price by 1000x. Mean aggregation is highly sensitive to outlier data and the attack resulted in Synthetix's loss of several million dollars [195].

**Statistical Measures.**

The three core statistics for aggregation are mean, median and mode. Many oracle systems use the median as the aggregated output, by selecting the middle entry of a list of ordinal data inputs. Unlike the mean, the median is not skewed by outliers, although it assumes the inputs have an appropriate statistical distribution where

the median is a representative statistic for the underlying ground-truth value. For example, if we believe data from the feeders is normally distributed with possible outliers, the median is appropriate. However if we believe it is bi-modally distributed, then discretizing and computing the mode (most common value) of the data is more appropriate. The mode is useful for non-numeric data (and nominal numbers). An approximation to the mode is picking a data input at random, however access to randomness from a smart contract is a well-documented challenge [46, 40, 44]. Oracle projects like Chainlink do not prescribe a fixed aggregation method and let the data requesters select one.

To improve the quality of simple statistics such as the median and the mode, weights can be applied in the calculation. For instance, to mitigate manipulation of price data, one can choose to use *time-weighted average price* (TWAP) [215], or liquidity volume, or both [12]. Typically, the liquidity and trading volume of a market correlates with the quality of the price data. To illustrate, Uniswap V2 uses TWAP over several blocks (*e.g.,* mean price in the last 10 blocks) to reduce the possibility of market manipulation in a single block (*e.g.,* via flash loans [176]). In Uniswap V3, TWAP is optimized for more detailed queries including the liquidity volume and allowing users to compute the geometric mean TWAP [12].

**Stale Data.**

Some use cases require frequent updates to data, such as weather data and asset prices. Stale data can be seen as valid data and pass the selection criteria, but it

will reduce the aggregated data quality. Projects like Chainlink rank feeders based on historic timeliness. A naive approach ignores this issue and always uses the last submitted data of a data feeder even if the data feeder has not updated its price for some specific period. This approach is problematic if the underlying data is expected to change frequently. An example occurred on Black Thursday 2020 [226] to MakerDao when Maker's data feeders could not update their feeds because of very high network congestion. After a significant delay in time, feeds were updated. The price had shifted by a large amount and the reported data jumped, leading to sudden, massive liquidations that were not adequately auctioned off.

### 5.3.6 Dispute Phase

The dispute phase is used to safeguard the quality of the final output and give the stakeholders a chance to mitigate inclusion of wrong data. Dispute resolution can be an independent module after the aggregation phase or it can be implemented at any other oracle module (*e.g.,* at the end of every aggregation 5.3.5 or data feeder selection 5.3.4). Most oracle systems do dispute resolution internally, but market specialization has produced firms that provide outsourced dispute resolution as service (*e.g.,* Kleros [128]). To systemize the landscape, we first distinguish between systems that aim to detect (and remove) bad data providers and systems that vet the data itself. We then iterate how data is determined to be valid or invalid for the purposes of a dispute. Finally, we illustrate the consequences of a successful dispute: what happens to the disputed data and what happens to its provider.

### Provider-level and Data-level Vetting

Dispute resolution can be *provider-oriented* or data-oriented. Under a *provider-oriented* regime, the focus is on selecting honest data providers and using disputes to remove data providers from serving as oracles in the future. In the optimistic case that providers are honest, oracle data is available immediately, however if an honest provider is corrupted, it will have a window of opportunity to provide malicious data before being excluded. One illustration of a provider-oriented system is operating a centralized allowlist of data providers (*e.g.,* MakerDAO v2) where providers can be removed. Chainlink [76] strives to decentralize this functionality, where a reputation-based leaderboard replaces the allowlist.

In a data-oriented regime, the focus is vetting the data itself. This can result in a slower system as oracle data is staged for a dispute period before it is finalized, however it can also correct false data (not merely remove the corrupted data feeder from future submissions). One illustration of a data-oriented system is Tellor [201, 58], where data is staged for 24 hours before finalization. If it is disputed, a period of up to 7 days is implemented to resolve the dispute. It is also possible that a system allows the resolution itself to be further disputed with one or more additional rounds. In Augur [169] for instance, the dispute step may happen in one round (takes maximum 1 day) or may contain other rounds of disputes that can last more than 7 days.

### Determining the Truth

In the optimistic case, an oracle system will feed and finalize truthful data, while disputes enable recourse for incorrect data. However disputes also introduce the possibility of two types of errors.

| | No Disputes | Disputed |
|---|---|---|
| Data is correct | Correct | False Positive |
| Data is incorrect | False Negative | Correct |

Table 5.2: Dispute phase types of errors

Dispute resolution in oracle systems focus on false positives. Incentivizing the discovery of false positives is present in some staking-based systems, however false negatives are not otherwise dealt with. In order to resolve a false positive, correct data must be used as a reference but, of course, if correct data is available as a reference, then it could replace the entire oracle system. That leaves two reasons for why an oracle system might still exist: (a) the reference for correct data is too expensive to consult on a regular basis, or (b) there is no reference for correct data and it must be approximated.

If feeders are placed on an allowlist by a trusted party, disputes could be filed with the trusted party and manually verified. As far as we know, this is the only example of (a), although (a) is the basis for other blockchain-based dispute resolution protocols like optimistic roll-ups [113]. The rest of the truth discovery mechanisms are based on (b) approximating the truth.

A *statistical approach* is selecting, from a set of values proposed by different feeders, the median of the values (*e.g.,* appropriately distributed continuous numerical

data) or the mode (*e.g.,* non-continuous or non-numerical data). It is possible to augment this approach by having feeders *stake* collateral in some cryptocurrency (*e.g.,* a governance token for the oracle project), and this collateral is taken (*slashed*) from the feeder if their data deviates from the median by some threshold. If the amount slashed is payed, in part or in full, to the entity that filed and/or supported a dispute on the data, this incentivizes feeders to help reduce false negative errors in addition to false positives. One challenge is setting an acceptable threshold for slashing. A large threshold tolerates moderately incorrect data without punishment, while a small threshold could punish data feeders that are generally honest but faulty, slow, or reporting on highly volatile data.

If a governance token exists for the oracle project, a related approach is to introduce *voting* on disputed data by any token holder, and not limit the decision to just the feeders. In Augur [169] and ASTRAEA [13], disputers vote to change the tentative outcome because they believe that outcome is false. Voting occurs over a window of time which extends the time to resolve disputes. By comparison, statistical mechanisms can be applied automatically and nearly instantly after the data is aggregated. However voting incorporates human judgement which might produce better outcomes in nuanced situations.

One final truth discovery mechanism is *arbitrage* which is applicable in the narrow category of exchange rates between two on-chain tokens. This can be illustrated by the NEST oracle [160] where data feeders assert the correct exchange rate between two tokens by offering a minimum amount of both tokens at this rate (*e.g.,* 10 ETH

and 39,000 USDT for a rate of ETH/USDT = 3900). If the rate is incorrect, other participants will be given an arbitrage opportunity to buy/sell ETH at this rate, an action that can correct the price. This is very similar to drawing a price from an on-chain exchange, like Uniswap, and suffers from he same issue: an adversary can manipulate the oracle price by spending money. It is secure when the *Cost of Corruption* (CoC) is greater than the *Profit from Corruption* (PfC), however PfC can never be adequately accounted for because profit can come from extraneous (extra-Ethereum) factors [91]. The UMA [125] oracle system has data feeders provide their own PfC estimates for the data they provide.

**Consequences for Incorrect Data**

We now consider the consequences for disputed data that has been determined to be incorrect. In provider-oriented dispute resolution, incorrect data has consequences for the data feeder (see next subsection) but not the data itself. By the time the dispute is resolved, it is *too late* to change the data itself.

In data-oriented dispute resolution, data that has been deemed incorrect can either be *reverted* or *corrected*. Reversion means the outcome result will be annulled and the system should start from scratch to obtain new data, while corrected data will reflect a new undisputed value. The difference between the two is essentially in the complexity of the dispute resolution system. For reversion, a collective decision is taken to accept or reject data — a binary option that is known in advanced. By contrast, correcting data requires new data to be proposed and then a collective

decision to be made on all the proposals which is more complex but does not avoids rerunning the oracle workflow.

These differences also impact *finality*: when should oracle data be considered usable? Dispute periods, re-running the workflow, and allowing resolved disputes to be further disputed can all introduce delays. To illustrate, consider Augur [169] which implements a prediction market on binary events. Any observer with an objection to a tentative outcome can start a dispute round by staking REP (Augur's native token) on the opposite outcome. Dispute windows are 24 hours and then extended to 7 days for disputes on disputes. If the total staked amount exceeds 2.5% of all REP tokens, the market enters a 60-day settlement phase called a fork window when all REP holders are obliged to stake on the final outcome.

**Consequences for Data Feeder**

If data has been deemed incorrect through disputes or rejected for being an outlier, the feeder who provided the data might face consequences like being banned, slashed, or suffering reputational loss. It is also possible that there is *no consequence* for the feeder other than the data being discarded. For example, in a sensor network, results from faulty sensors could have their data filtered out but continue to contribute data in expectation that they will be repaired in the future.

In oracle designs based on allowlists, a feeder could be *banned* or temporarily suspended for providing incorrect data. For dispute resolution based on staking, a feed could suffer *economic loss* by having their stake taken from them. It is important to

reiterate that this economic loss does not necessarily outweigh the utility of attempting to corrupt oracle data. The profit from corruption depends on where the data is being used, which could be within larger system than the blockchain itself [91]. Finally, a feeder might suffer *reputational loss* for providing incorrect data. One can imagine this would be the case if, for example, the Associated Press misreported the outcome of the 2020 US Presidential election after announcing that it would serve as an oracle for this event on Ethereum.

Another illustration of these options is Chainlink, which maintains a decentralized analogue to a leaderboard where feeders are ranked according to the amount of LINK (Chainlink's token) they stake, as well as their past behavior in providing data that is timely and found to be correct. Data feeders with the outlier data will be punished by losing their collateralized LINK tokens and reducing their reputation score on the reputation registry. The lost of tokens is a direct cost, while the loss of reputation could impact their future revenue.

### 5.3.7 Classification of Current Oracle Projects

In Table 5.3, we present a classification of several oracle implementations using the modular framework described in this section. This table showcases a wide variety of approaches, as well as some specialization on specific modules (*e.g.,* TownCrier and Deco on data source and Kleros on dispute resolution). We caution that blockchain projects can change how they work very quickly, new projects will emerge, and current projects will be abandoned. Table 5.3 has a limited shelf-life of usefulness, however

| *Oracle* | Data Source | Data Feeder: Selection Mechanism | Data Feeder: Staking | Aggregation Mechanism | Dispute: Provider/Data Vetting | Dispute: Determining the Truth | Dispute: Consequences (Slash, Ban, Lo |
|---|---|---|---|---|---|---|---|
| ChainLink [76] | API | Reputation, Staking | • | Statistical Measure | P | Statistical Measure | S |
| UMA [214] | Human, API | FCFS† | • | × | D | Staking | S |
| Augur [169] | Human | Single Source⋆ | • | × | D | Voting | S |
| Uniswap [215] | Smart Contract | × | × | TWAP | × | × | × |
| MakerDAO V1 [134] | Human, API | Centralized Allowlist | × | Median | × | × | × |
| MakerDAO V2 [134] | Human, API | Decentralized Allowlist | × | Median | P | Voting | B |
| NEST [160] | Human | × | • | ×⋆⋆ | D | Arbitrage | L |
| Band protocol [175] | API | Random Selection | • | Statistical Measure | P | Staking | S |
| Tellor [58] | Human, API | PoW | • | Median | P | Staking | S B |
| ASTRAEA [13] TruthCoin [197] | Human | Staking | • | Mode | D | Voting | S |
| Provable [25] PriceGeth [80] | API | × | × | × | × | × | × |
| DIA Oracle [69] | API, Smart Contract | × | × | × | D | Staking | B |
| DECO [237] TownCrier [236] | HTTPS | × | × | × | × | × | × |
| API3 [24] \w Kleros [128] | Oracles | Decentralized Allowlist | • | Statistical Measure | P | Voting | S B |

Table 5.3: A classification of the existing oracle implementations using the modular framework described in Section 5.3.
• indicates the properties (columns) are implemented in the corresponding oracle (rows), and × indicates the property is not applicable.
† First Come First Serve ⋆*The Market Creator assigns the designated reporter* ⋆⋆ *The series of reported prices will be sent to requester without aggregation (See 5.3.6)*

the workflow itself (modules, sub-modules, and design choices) is based on general principles and intended to have long-lasting usefulness. We exclude modules described in section 5.4 from this table as the infrastructure and implementation can differ for

different use-cases of the oracle. While we can find some sort of implementation for most of the projects listed in table 5.3, it is hard to determine which one's are deployed in "production". Many have testnet or sidechain deployments, some are academic work and some have code but not clear if there are actual users.

## 5.4 Interacting with the blockchain

While the initial inputs to an oracle are generally *off-chain* (with the exception of pulling data from another smart contract) and the final output is by definition *on-chain*, the oracle designer will choose to implement the intermediary modules—data feeder selection, aggregation and dispute resolution—as either off-chain or on-chain. Generally, on-chain modules are preferred for transparency and immutability, while off-chain modules are preferred for lower costs and greater scalability.

To illustrate, Chainlink and NEST Protocols were ranked #5 and #7 respectively in gas usage among all DApps on Ethereum.[4] This ranking was achieved mainly because they implement all modules fully on-chain. Later, Chainlink implemented an off-chain reporting (OCR) protocol [35] with the goal of reducing the gas costs associated with on-chain transactions. This protocol uses digital signatures to authenticate feeders and a standard (*e.g.,* Byzantine fault tolerant [43]) consensus protocol between Chainlink nodes.

At some point, an oracle system must move on-chain and start interacting with the underlying blockchain. We assume for the purpose of illustration that Ethereum

[4]Based on Huobi DeFiLabs Insight on September 2020 [109]

is the blockchain being used. Data flow from an off-chain module to a smart contract involves the following three components which we detail in this section.

**5.4.1 Off-chain Infrastructure**: Assuming at least one module is off-chain, an infrastructure is required to monitor requests for oracle data from the blockchain, gather the data from the data sources, implement a communication network between data feeders, and create a final transaction to be sent to the blockchain infrastructure.

**5.4.2 Blockchain Infrastructure**: Off-chain infrastructure will pass the data as a transaction to blockchain nodes, which relay transactions and use a consensus algorithm agree on new blocks. The nodes run by miners are discussed in particular as they dictate the order of transactions in every block they mine.

**5.4.3 Smart Contracts**: The transaction triggers a state change in a smart contract on the blockchain, typically a contract owned by the oracle which is accessible from all other contracts. Alternatively, the oracle could write directly into a data consumer's contract (called a *callback*).

### 5.4.1 Off-chain Infrastructure

Depending on the oracle design, there can be different types of off-chain infrastructure. If financial data is pulled from Uniswap's oracle [215], there is no off-chain infrastructure needed because the oracle is already a fully on-chain oracle. For other applications, off-chain infrastructure could consist of a single server (*e.g.,* TownCrier [236])

or many nodes that intercommunicate through their own consensus protocol (*e.g.,* Chainlink OCR [35]). Availability and DOS-resistance [192] are core requirements of off-chain infrastructure, specially in oracle systems working with time-sensitive data and high update frequency. In this section we describe different possible components of the off-chain infrastructure.

### Monitoring the Blockchain

For oracles that are capable of returning a custom data request made on-chain (called *request-response oracles*), every data feeder needs to monitor the oracle's smart contract for data requests. The common implementation consists of a server subscribing to a blockchain node for specific events.

### Connection to Data Source

The data feeder requires a connection to the data source 5.3.2 to fetch the desired data. This connection can be an entry point for an adversary to manipulate the data however it is possible to mitigate this issue by integrating message authentication (recall source authentication in Section 5.3.3). Examples include relaying HTTPS data (*e.g.,* Provable [25] via TLSNotary [212]) or from trusted hardware enclaves (*e.g.,* TownCrier [236] via Intel SGX [59]). Vulnerabilities with the web-server or SGX itself [34] are still possible attack vectors.

### Data Feeders Network

In order to increase the scalability of the oracle network, multiple data feeders might aggregate their data off-chain (*e.g.,* Chainlink OCR [35]). In OCR, a leader is chosen from the participants to gather signed data points from other nodes. Once consensus is achieved on the aggregated set of data, the finalized data, accompanied by the signatures, is transmitted to the blockchain node. This reduces the costs as only one transaction is sent to the blockchain, while maintaining similar security as having each chainlink nodes send the data themselves.

Like any network system, availability is essential to the operation of the oracle. To illustrate, in December 2020, MakerDAO's oracle V2 had an outage due to a bug in their peer-to-peer data feeder network stack [93]. We do not summarize all the literature on peer-to-peer network attacks, but denial-of-service attacks [229] and sybil-attacks [73] are critical to mitigate to ensure the availability of the network and the oracle.

### Transaction Creation

In order to submit data to a blockchain, the data feeder is required to construct a valid blockchain transaction that includes the requested data. This transaction must be signed with the data feeder's private key to be validated and authenticated on-chain. The data feeders must protect the signing keys from theft and loss [79], as this key can be used to impersonate the oracle.

Transactions compete for inclusion in the next block by offering different levels

of transaction fees, known as the gas fee in Ethereum. In time-sensitive oracle applications, the relay must specify an appropriate amount of gas according to market conditions. For instance, on 'Black Thursday' in March 2020 [226], the Ethereum network was congested by high fee transactions and some oracles failed to adjust their price feed. To mitigate this problem, the module which is responsible for creating the final transaction must have a *dynamic gas* mechanism for situations where gas prices are rapidly climbing. In this case, pending transactions must be canceled, and new ones must be generated with higher gas price, which may take a few iterations to get in. Dynamic fees depend directly on the network state and require a connection to the blockchain node to estimate the adequate gas price.

In addition, the data feeder's sending address on the blockchain must have sufficient funds to be able to pay the estimated gas price. It is crucial for the availability of the oracle that the data feeders monitor their account balance as spam attacks might drain their reserves with high gas fees, as happened to nine Chainlink operators in September 2020 [60].

### 5.4.2 Blockchain Infrastructure

In this section, we discuss the blockchain infrastructure that is required by any entity interacting with the blockchain. While this infrastructure is not specific to oracles, we illustrate key points that can impact oracle availability.

### Blockchain Node

A blockchain node relays transactions to the other nodes in the network for inclusion in the blockchain. The node is responsible for storing, verifying, and syncing blockchain data. The availability of nodes is very important for the oracle system, as a blocked node cannot send transactions. Extensive research on network partitioning attacks apply to decentralized networks, with the main objective of surrounding an honest nodes with the malicious nodes [217, 161, 238, 101, 103]. This results in the node believing it is connected to the blockchain network when it is not.

### Block Creation

Transactions that have been circulated to the blockchain network are stored in each node's `mempool`. Mining nodes select transactions from their `mempool` according to their priorities (*e.g.,* by highest gas price as in Geth [84], while respecting nonces). Front-running attacks [82, 64] try to manipulate how miners sequence transactions. For example, someone might observe an unconfirmed oracle transaction in the mempool, craft a transaction that profits from knowing what the oracle data will be, and attempt to have this transaction confirmed before the oracle transaction itself (called an *insertion attack* [82]). This might be conducted by the miner themself. In this case, it is called *transaction reordering*, and the profit miners stand to make from doing this is termed *Miner Extractable Value (MEV)* [64]). Other nodes or users on the network who can act quickly and offer high fees can also conduct front-running attacks. Users might also attempt a *bulk displacement attack* [82] that fills the con-

secutive blocks completely to delay reported data from oracles. There could be a profit motive for this attack if the oracle data becomes expired, or if the data feeder's collateral is slashed and redistributed to the attacker.

Research on MEV (*e.g.,* Flashbots [90]) has shown the possibility of new type of attacks based on reordering the transactions, such that if there's a high profit for changing the order of some transactions in a (few) blocks, miner is incentivize to use his hash rate to perform a reorganization attack[5] [129], and profit from the execution of the newly ordered transactions. For instance, Uniswap uses the last price in a block to determine the average price (TWAP), in which a miner can add new trades while reordering the past trades with the goal of manipulating the price average to profit on other applications that uses Uniswap as price oracle.

### Consensus

The goal of the consensus algorithm used in the blockchain is to verify and append the next block of transactions to the blockchain. If the nodes do not come to agreement on a state change, a fork in the network happens with different nodes trying to finalize different forks of the blockchain. Given the network is decentralized, short-lived forks happens frequently in the network that generally are resolved within a few blocks [162]. All valid transactions in the abandoned fork will eventually be mined in the main chain, likely in a new order (called *reorganization* or a *reorg*).

A reorg opens the possibility of attacks by using known, unconfirmed, transactions

[5]Also referred to as *Time-bandit attacks* [64]

from the abandoned fork. To illustrate, consider Etheroll [85], an on-chain gambling game where users bet by sending a number that payouts if it is smaller than a random number determined by an oracle. To prevent front-running from the mempool, the Etheroll oracle would only respond when a bet was in a block. Despite this mitigation, in April 2020, the Etheroll team detected an ongoing front-running attack on their platform [78]. The attacker was betting rigorously and waiting for small forks—collected by Ethereum in *uncle blocks*—where the original bet and oracle's random number response were temporarily discarded by the reorg. The attacker would place a winning bet with a high fee to front-run the original bet and eventual inclusion of the oracle's transaction in the reorganized chain. A general principle of this attack is that even if oracle data bypasses the mempool and is incorporated directly by miners, front-running through reorgs is still possible.

There are two solutions to front-running through reorgs. The first is to delay the settlement of the bet by a few blocks to prevent issues caused by small reorganization forks. The second is to incorporate a hash of the request (*e.g., request-id*) in the response to prevents the request (*e.g.,* bet) from being swapped out once the response (*e.g.,* random number) is known.

Other consensus attacks [96, 27, 103] exist but are less related to oracles. We omit discussion of them.

### 5.4.3 Smart Contracts

Although oracles are usually designed to be the source of truth for on-chain smart contracts, some smart contracts can also be used as oracles by others even though they were not designed with the oracle use-case in mind. To expand this idea, oracles could be a *'an end in itself'*, which is to say they are designed specifically to be used as a source of truth. These oracles fetch the data from external sources(5.3.2) and make it available on-chain (*e.g.,* PriceGeth [80]).

By contrast, *a means to an end* oracle is a contract that produces useful data as a byproduct of what it is otherwise doing. Examples are on-chain markets and exchanges like Uniswap and other automated market makers (AMMs). The markets are designed for facilitating trades but provide pricing information (*price discovery*) that can be used by other contracts (*e.g.,* margin trading platforms) as their source of truth.

In this section we dive deeper in the relationship between the oracle's smart contract and the data consumer smart contract. We start by defining possible interaction models, and then discuss specific issues related to the oracle's contract and the consumer's contract.

#### Oracle Interaction Models

A distinction in the oracle design is whether the interaction between with the consumer's contract is implemented as a *feed*, a *request-response*, or the related *subscribe-response*.

A *Feed* is a smart contract system that publishes the data for others to use. It does not require any requests to fetch the data and using an interval to update the data on its smart contract (*e.g.,* Maker DAO Oracle [134]). From a technical aspect, in order to use a feed oracle, the data consumer smart contract only needs to query the oracle's smart contract and no additional transactions are needed.

The *Request-Response* model is similar to a client-server API request on traditional web development. The requester must send a request to the oracle's smart contract, which then is picked up by the off-chain module of the oracle to fetch the requested data from the data source. The data is then encapsulated in a transaction and sent back to the data requester smart contract through the oracle's smart contract. Due to the nature of this design, at least two transactions are needed to complete the work flow, one from the requester and another for the responder.

The *Subscribe-Response* model is similar to Request-Response with one main difference, the request does not need to be in a transaction. If there is pre-arranged agreement, the oracle will watch for emitted events from the requester smart contract and respond to the requests. Alternatively, the requester is allowed to read the feed through an off-chain agreement (*e.g.,* API3 [24]).

#### Oracle's Smart Contract

In the oracle designs that implement some of the modules on-chain, the oracle's smart contract could include data feeder selection (Section 5.3.4), aggregation (Section 5.3.5), and dispute resolution (Section 5.3.6). In addition to these modules, the

oracle's smart contract can be used as the data feed storage for other smart contracts to read from, or to authenticate the oracle's response on the consumer smart contract. In the *feed* model, the oracle's smart contract is where the consumer fetches the oracle data from. In the *Request-Response* model, the data consumer smart contract (defined below in Section 5.4.3) requires knowledge of the oracle's smart contract's address in advance, for the initial request and also verification of the oracle's response. For the rest of this section, we discussion potential attacks on the oracle's smart contract.

**Implementation Flaws** There are many known smart contract vulnerabilities that have been extensively discussed [56, 48] and possibly could affect the legitimacy of the oracle system.

In many DeFi projects, a common design pattern is to use on-chain markets, such as Uniswap, for the price oracle, however, these systems were not designed to be used as oracles and are prone to market manipulation. The end result is that currently, the most prevalent attack vector in DeFi is oracle manipulation [61]. To illustrate this attack, consider the lending (and margin trading) platform bZx. It fetched prices from KyberSwap, a decentralized exchange, to calculate the amount of collateral of one cryptoasset is needed to back the loan of a different asset. In one attack on bZx [167], the attacker used a *flash loan* to manipulate KyberSwap's sUSD/ETH exchange rate. The attacker then borrowed ETH with insufficient collateral because the bZx contract believed the collateralized sUSD was worth much more than it

actually was. When the attacker absconded with the borrowed ETH, forgoing its collateral, and then unwound its other positions and repaid the flash loan, it profited at bZx's expense. Arguably bZx (the data consumer) is the flawed contract but the ease in which KyberSwap (the oracle contract) could be manipulated was not well understood at the time either. In reaction, decentralized exchanges embraced their role as a price oracle and hardened themselves against price manipulation by using aggregation methods like the *Time-Weighted Average Price* (TWAP) (described in Section 5.3.5).

**Governance** In order to remove the centralization of control in many DeFi projects, a governance model is introduced that uses a native token for voting and staking. The governance model for an oracle could propose, vote, and finalize changes to system variables like the approved data feeders on the oracle's allowlist or various fees.

While a decentralized governance model removes the trust in a central entity, it does not remove the possibility of a wealthy entity (a *whale*) taking control of the system by accumulating (or borrowing [176]) enough tokens to pass their proposals. In addition, logical issues in the governance implementation could result in tricking the voters into approving a proposal that has malicious consequences [156].

As an example, in the MakerDAO platform, MKR token holders can vote to change parameters related to Maker's oracle module [134]. An attacker in October 2020, used a flash loan to borrow enough MKR tokens to pass a governance proposal, aimed to change the list of consumer smart contracts and obtain read access to the

Maker's oracle [132]. It could be more dangerous if the attacker planed to change the other parameters of the oracle such as *Whitelisted data feeders* or *bar* parameter : the sufficient number of data feeders for data feeder selection module. Potentially an attacker may pay a bribe to the MKR holders to buy their votes, or use a *Decentralized Autonomous Organization (DAO)* to pay for the votes without having ownership of the tokens [66].

### Data Consumer Smart Contract

The final point in the oracle workflow is the smart contract that needs the data for its business logic. Aside from any possible code vulnerabilities in this smart contract, there are common implementation patterns concerning the oracle workflow.

In the *feed* model, the data consumer smart contract relies on oracles to fetch the required data in order to function as intended. It is essential to use oracles with multiple data feeders and a proper aggregation methods. To illustrate the importance, consider the lending service Compound [55] which initially only used Coinbase Pro as their data feeder without any aggregation mechanisms [67]. In November 2020, a faulty price feed on Coinbase Pro, resulted in undercollarization of Compound loans and a liquidation of $89 million dollars of the collateral. This could have been prevented by using an oracle with sufficient data feeders and a proper aggregation mechanism.

Due to the commonality of this issue, there has been some Ethereum Improvement Proposals (EIPs) to standardize the interface of the oracles implementing a *feed* (*e.g.,*

EIP-2362 [202]). An interface would allow data consumer smart contracts to easily switch between feeds or use multiple oracle feeds in their logic.

In the *request-response* model, the data consumer smart contract sends a request for specific data to the oracle's smart contract. In some projects this request contains more information like the data feeder selection method, aggregation algorithm and parameters for dispute phase (*e.g.,* Service Level Agreement in Chainlink). It is crucial that the data consumer smart contract, verifies the authenticity of the oracle response. Failure to verify the oracle's response could result in malicious data injection in the data consumer smart contract. To illustrate, the insurance service Nexus Mutual [156] implemented an oracle's response function (or callback) without any proper access control. This opened the possibility of unauthorized entities providing data updates which would be wrongfully assumed to have originated from the oracle's smart contract.

## 5.5 Concluding Remarks

In this proposal, we described a specialized modular framework to analyze oracles. After our systematization, we present the following discussion points and lessons learned from our work.

1. Many oracles projects introduce their own governance tokens that are used to secure the oracle system (*e.g.,* through staking). Two conditions seem necessary: the market capitalization of the token stays material and the token is

evenly distributed. More consideration should be given to leveraging an existing token with these properties (even a non-oracle token) instead of creating new specialized tokens [41]. Also a collapse in the value of the governance token threatens the entire system.

2. Oracle systems with on-chain modules are expensive to run on public blockchains like Ethereum, which prices out certain use-cases that consume a lot of oracle data but do not generate proportional amount of revenue (*e.g.,* Weather data).

3. Diversity in software promotes resilience in the system. If the oracle market coalesces behind a single project, a failure within this project could cause cascading failures across DeFi and other blockchain applications.

4. While determining the profit from corrupting the oracle is a promising approach to thwarting manipulation (by ensuring the cost of corruption is greater), one can never capture the full extent of the potential profit. Attackers can profit outside of Ethereum by attacking oracles on Ethereum [91].

In summary of this proposal, the framework we present facilitates a modular approach in evaluating the security of any oracle design and its associated components that exist today or to be implemented in the future. As an example, the level of centralization can be measured using choke points such as aggregation 5.3.5, or how the data is proceeded to the blockchain 5.4.1. In order to design a secure oracle, all modules must be rigorously stress tested to make sure it cannot be gamed by participants or malicious actors. In addition, many security auditors and analysis

tools are specialized in detecting oracle-related attacks through code review of the smart contracts. Specially with the rise of DeFi smart contracts, the importance of a secure oracle system remain a paramount component of the decentralized blockchain ecosystem.

# Chapter 6

# Auditing Blockchain-based Assets

*This chapter is based on the paper "Systemizing the Challenges of Auditing Blockchain-Based Assets" [172] published on American Accounting Association Journal of Information Systems [172]. This paper was supervised by Jeremy Clark and co-authored by Erica Pimentel, Emilio Boulianne*

# Chapter 7

# Concluding Remarks

# Bibliography


[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] Aragon.org.

[3] Im-2110-3. front running policy, 2002.

[4] Ssac advisory on domain name front running, 10 2007. (Accessed on 08/15/2018).

[5] 5270. front running of block transactions, 2012.

[6] Notice of filing of proposed rule change to adopt finra rule 5270 (front running of block transactions) in the consolidated finra rulebook, 2012.

[7] Barclays trader charged with front-running by us authorities, 2018.

[8] worker.go - commitnewwork(), 2018. Accessed: 2018-12-07.

[9] r. o. t. s. s. o. t. o. m. t. t. s. 96th cong, 1st sess and exchange comission, 1978.

[10] M. E. Acer, E. Stark, A. P. Felt, S. Fahl, R. Bhargava, B. Dev, M. Braithwaite, R. Sleevi, and P. Tabriz. Where the wild warnings are: Root causes of chrome https certificate errors. In *CCS*, CCS '17, pages 1407–1420, New York, NY, USA, 2017. ACM.

[11] H. Adams. Uniswap. *URl: https://uniswap.org/docs*, 2019.

[12] H. Adams, N. Zinsmeister, M. Salem, R. Keefer, and D. Robinson. Uniswap v3 core. 2021.

[13] J. Adler, R. Berryhill, A. Veneris, Z. Poulos, N. Veira, and A. Kastania. Astraea: A decentralized blockchain oracle. In *2018 IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData)*, pages 1145–1152. IEEE, 2018.

[14] M. Alizadeh, M. Zamani, A. R. Shahemabad, J. Shayan, and A. Azarnik. A survey on attacks in rfid networks. *Open International Journal of Informatics (OIJI)*, 1(1):15–24, 2012.

[15] G. Angeris and T. Chitra. Improved price oracles: Constant function market makers. *arXiv preprint arXiv:2003.10001*, 2020.

[16] Anonymous. How the first winner of fomo3d won the jackpot? `https://winnerfomo3d.home.blog/`, 2018. Accessed: 2018-09-09.

[17] N. Atzei, M. Bartoletti, and T. Cimoli. A survey of attacks on ethereum smart contracts (sok). In *International conference on principles of security and trust*, pages 164–186. Springer, 2017.

[18] R. Aune, M. O'Hara, and O. Slama. Footprints on the blockchain: Information leakage in distributed ledgers. *Available at SSRN 2896803*, 2017.

[19] Balancer. A protocol for programmable liquidity. *URl: https://balancer.finance/*, 2020.

[20] 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.

[21] C. Baum, J. Hsin-yu Chiang, B. David, T. K. Frederiksen, and L. Gentile. Sok: Mitigation of front-running in decentralized finance. In *International Conference on Financial Cryptography and Data Security*, pages 250–271. Springer, 2022.

[22] BBC. Websites hacked to mint crypto-cash. `http://www.bbc.com/news/technology-41518351`, 2017.

[23] D. Beaver and S. Haber. Cryptographic protocols provably secure against dynamic adversaries. In *Workshop on the Theory and Application of Cryptographic Techniques*, pages 307–323. Springer, 1992.

[24] B. Benligiray, S. Milic, and H. Vänttinen. Decentralized apis for web 3.0.

[25] T. Bernani. Oraclize. *URl: https://www.oraclize.it/*, 2016.

[26] A. Biryukov, D. Khovratovich, and S. Tikhomirov. Findel: Secure derivative contracts for ethereum. In *International Conference on Financial Cryptography and Data Security*, pages 453–467. Springer, 2017.

[27] G. Bissias, B. N. Levine, A. P. Ozisik, and G. Andresen. An analysis of attacks on blockchain consensus. *arXiv preprint arXiv:1610.07985*, 2016.

[28] BleepingComputer. The internet is ride with in-browser miners and it is getting worse each day. `https://www.bleepingcomputer.com/news/security/the-internet-is-rife-with-in-browser-miners-and-its-getting-worse-each-day/`, 2012. Accessed: 2017-12-08.

[29] C. Blog. 16 ways to create dynamic non-fungible tokens (nft) using chainlink oracles. *URl: https://blog.chain.link/create-dynamic-nfts-using-chainlink-oracles/*, 2021.

[30] 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).

[31] D. Boneh, J. Bonneau, B. Bünz, and B. Fisch. Verifiable delay functions. In *Advances in Cryptology – CRYPTO 2018*, volume 10991 of *Lecture Notes in Computer Science*, pages 757–788. Springer, 2018.

[32] J. Bonneau, E. W. Felten, S. Goldfeder, J. A. Kroll, and A. Narayanan. Why buy when you can rent? bribery attacks on bitcoin consensus, 2016.

[33] G. Brassard, D. Chaum, and C. Crépeau. Minimum disclosure proofs of knowledge. *Journal of computer and system sciences*, 37(2):156–189, 1988.

[34] F. Brasser, U. Müller, A. Dmitrienko, K. Kostiainen, S. Capkun, and A.-R. Sadeghi. Software grand exposure:{SGX} cache attacks are practical. In *11th {USENIX} Workshop on Offensive Technologies ({WOOT} 17)*, 2017.

[35] L. Breidenbach, C. Cachin, A. Coventry, A. Juels, and A. Miller. Chainlink off-chain reporting protocol. *URl: https://blog.chain.link/off-chain-reporting-live-on-mainnet/*, 2021.

[36] L. Breidenbach, P. Daian, A. Juels, and F. Tramer. To sink frontrunners, send in the submarines, 2017. Accessed: 2018-08-28.

[37] L. Breidenbach, T. Kell, S. Gosselin, and S. Eskandari. Libsubmarine: Defeat front-running on ethereum, 2018. Accessed: 2018-12-07.

[38] L. Breidenbach, I. C. 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.

[39] B. Bünz, J. Bootle, D. Boneh, A. Poelstra, P. Wuille, and G. Maxwell. Bulletproofs: Short proofs for confidential transactions and more. In *2018 IEEE Symposium on Security and Privacy (SP)*, pages 319–338, 2018.

[40] B. Bünz, S. Goldfeder, and J. Bonneau. Proofs-of-delay and randomness beacons in ethereum. *IEEE Security and Privacy on the blockchain (IEEE S&B)*, 2017.

[41] V. Buterin. Uni should become an oracle token. *URl: https://gov.uniswap.org/t/uni-should-become-an-oracle-token/11988*, 2021.

[42] S. Buti, B. Rindi, and I. M. Werner. Diving into dark pools, 2011.

[43] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[44] Chainlink. Chainlink vrf: On-chain verifiable randomness. *URl: https://blog.chain.link/verifiable-random-functions-vrf-random-number-generation-rng-feature/*, 2020.

[45] ChainLink. What is the blockchain oracle problem. *URl: https://blog.chain.link/what-is-the-blockchain-oracle-problem/*, 2020.

[46] K. Chatterjee, A. K. Goharshady, and A. Pourdamghani. Probabilistic smart contracts: Secure randomness on the blockchain. In *2019 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*, pages 403–412. IEEE, 2019.

[47] CheckPointResearchTeam. October's most wanted malware: Cryptocurrency mining presents new threat. `https://blog.checkpoint.com/2017/11/13/octobers-wanted-malware-cryptocurrency-mining-presents-new-threat/`, 2017.

[48] H. Chen, M. Pendleton, L. Njilla, and S. Xu. A survey on ethereum systems security: Vulnerabilities, attacks, and defenses. *ACM Computing Surveys (CSUR)*, 53(3):1–43, 2020.

[49] R. Cheng, F. Zhang, J. Kos, W. He, N. Hynes, N. Johnson, A. Juels, A. Miller, and D. Song. Ekiden: A platform for confidentiality-preserving, trustworthy, and performant smart contract execution. In *2018 IEEE symposium on security and privacy (SP)*, 2018.

[50] 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.

[51] J. Clark, D. Demirag, and S. Moosavi. Sok: Demystifying stablecoins. *CACM*, 2020.

[52] J. Clark and P. C. Van Oorschot. Sok: Ssl and https: Revisiting past challenges and evaluating certificate trust model enhancements. In *2013 IEEE Symposium on Security and Privacy*, pages 511–525. IEEE, 2013.

[53] Coinhive. Coinhive monetize your business with your users cpu power. `https://coinhive.com/`, 2017. Accessed: 2017-11-20.

[54] Coinmonk. Vulnerabilities in maker: Oracle-governance attacks, attack daos, and (de)centralization. *URl: https://medium.com/coinmonks/vulnerabilities-in-maker-oracle-governance-attacks-attack-daos-and-de-centralization-d943685adc2f*, 2019.

[55] Compound. Open price feed. *URl: https://compound.finance/docs/prices*, 2020.

[56] Consensys. Ethereum smart contract best practices, 2020.

[57] ConsenSys-Diligence. Security review of 0x smart contracts, 2017.

[58] T. Core. Staking, disputes, and voting. *URl: https://medium.com/tellor/staking-disputes-and-voting-ad09c66eb7bc/*, 2019.

[59] V. Costan and S. Devadas. Intel sgx explained. *IACR Cryptol. ePrint Arch.*, 2016(86):1–118, 2016.

[60] T. B. Crypto. Chainlink nodes were targeted in an attack last weekend that cost them at least 700 eth. *URl: https://www.theblockcrypto.com/post/76986/chainlink-nodes-attack-eth*, 2020.

[61] T. B. Crypto. Defi attacks: the general picture. *URl: https://www.theblockcrypto.com/research/105472/defi-attacks-the-general-picture*, 2021.

[62] Cryptonote. Cryptonote technology. `https://cryptonote.org/inside.php#equal-proof-of-work`, 2017. Accessed: 2017-11-20.

[63] P. M. da Silva, M. Matos, and J. Barreto. Fixed transaction ordering and admission in blockchains. *EuroDW2018/papers/eurodw18-Silva. pdf*, 2019.

[64] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning in decentralized exchanges, miner extractable value, and consensus instability. In *2020 IEEE Symposium on Security and Privacy (SP)*, pages 910–927. IEEE, 2020.

[65] P. Daian, S. Goldfeder, T. Kell, Y. Li, X. Zhao, I. Bentov, L. Breidenbach, and A. Juels. Flash boys 2.0: Frontrunning, transaction reordering, and consensus instability in decentralized exchanges. In *IEEE Symposium on Security and Privacy*, 2020.

[66] P. Daian, T. Kell, I. Miers, and A. Juels. On-chain vote buying and the rise of dark daos. *URl: https://hackingdistributed.com/2018/07/02/on-chain-vote-buying/*, 2018.

[67] Decrypt. Oracle exploit sees $89 million liquidated on compound. *URl: https://decrypt.co/49657/oracle-exploit-sees-100-million-liquidated-on-compound*, 2020.

[68] DeepDotWeb. Coinhive hacked and launches new opt-in service. `https://www.deepdotweb.com/2017/11/11/coinhive-hacked-launches-new-opt-service/`, 2017.

[69] DIA. Dia — transparent oracles for a decentralised financial ecosystem, 2020.

[70] E. Discussion. Handling frontrunning in the permanent registrar, 2018.

[71] distribuyed. A comprehensive list of decentralized exchanges (dex) of cryptocurrencies, tokens, derivatives and futures, and their protocols, 2018. Accessed: 2018-09-24.

[72] DODO. The dodo advantage. *URl: https://dodoex.github.io/docs/docs/advantages*, 2020.

[73] J. R. Douceur. The sybil attack. In *International workshop on peer-to-peer systems*, pages 251–260. Springer, 2002.

[74] Z. Durumeric, D. Adrian, A. Mirian, M. Bailey, and J. A. Halderman. A search engine backed by Internet-wide scanning. In *ACM CCS*, Oct. 2015.

[75] B. Edelman. Front-running study: Testing report, 2009.

[76] S. Ellis, A. Juels, and S. Nazarov. Chainlink: A decentralized oracle network. *Retrieved March*, 11:2018, 2017.

[77] W. Entriken, D. Shirley, J. Evans, and N. Sachs. Erc-721 non-fungible token standard, 2018. Accessed: 2018-08-31.

[78] S. Eskandari. Etherroll bug thread. *URl: https://twitter.com/sbetamc/status/1263220679937265671*, 2020.

[79] S. Eskandari, D. Barrera, E. Stobert, and J. Clark. A first look at the usability of Bitcoin key management. In *USEC*, 2015.

[80] S. Eskandari, J. Clark, V. Sundaresan, and M. Adham. On the feasibility of decentralized derivatives markets. In *International Conference on Financial Cryptography and Data Security*, pages 553–567. Springer, 2017.

[81] S. Eskandari, A. Leoutsarakos, T. Mursch, and J. Clark. A first look at browser-based cryptojacking. In *2018 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 58–66. IEEE, 2018.

[82] S. Eskandari, S. Moosavi, and J. Clark. Sok: Transparent dishonesty: front-running attacks on blockchain. In *WTSC*. FC, 2019.

[83] S. Eskandari, M. Salehi, W. C. Gu, and J. Clark. Sok: oracles from the ground truth to market manipulation. In *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, pages 127–141, 2021.

[84] Ethereum. Official go implementation of the ethereum protocol, 2015.

[85] Etheroll. Ethereum casino. *URl: https://etheroll.com/*, 2020.

[86] European-Commission. Cookies. `http://ec.europa.eu/ipg/basics/legal/cookies/index_en.htm`, 2011. Accessed: 2017-12-08.

[87] ExtremeTech. Browser-based mining malware found on pirate bay, others. `https://www.extremetech.com/internet/255971-browser-based-cryptocurrency-malware-appears-online-pirate-bay`, 2017. Accessed: 2017-11-20.

[88] A. Firat, S. Madnick, and B. Grosof. Knowledge integration to overcome ontological heterogeneity: Challenges from financial information systems. 2002.

[89] flashbots. Flashbots is a research and development organization formed to mitigate the negative externalities posed by maximal extractable value (mev) to stateful blockchains, starting with ethereum., 2020. URL: https://www.flashbots.net/.

[90] Flashbots. Flashbots. *URl: https://github.com/flashbots/pm*, 2021.

[91] B. Ford and R. Böhme. Rationality is self-defeating in permissionless systems, 2019.

[92] Fortune. Popular google chrome extension caught mining cryptocurrency on thousands of computers. `http://fortune.com/2018/01/02/google-chrome-extension-cryptocurrency-mining-monero/`, 2017. Accessed: 2018-01-20.

[93] M. Forum. Scientific governance and risk #120. *URl: https://forum.makerdao.com/t/agenda-discussion-scientific-governance-and-risk-120-thursday-december-3-16-00-utc/5357*, 2020.

[94] J. Garay, A. Kiayias, and N. Leonardos. The bitcoin backbone protocol: Analysis and applications. In *EUROCRYPT*, 2015.

[95] S. Goldberg, D. Papadopoulos, and J. Včelák. Verifiable Random Functions (VRFs). Internet-Draft draft-goldbe-vrf-01, Internet Engineering Task Force, June 2017. Work in Progress.

[96] V. Gramoli. From blockchain consensus back to byzantine consensus. *Future Generation Computer Systems*, 107:760–769, 2020.

[97] W. C. Gu, A. Raghuvanshi, and D. Boneh. Empirical measurements on pricing oracles and decentralized governance for stablecoins. *Available at SSRN 3611231*, 2020.

[98] F. Hakimpour and A. Geppert. Resolving semantic heterogeneity in schema integration. In *Proceedings of the international conference on Formal Ontology in Information Systems-Volume 2001*, pages 297–308, 2001.

[99] R. Hanson. Combinatorial information market design. *Information Systems Frontiers*, 5(1):107–119, 2003.

[100] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoins peer-to-peer network. In *USENIX Security*, pages 129–144, Washington, D.C., 2015. USENIX Association.

[101] E. Heilman, A. Kendler, A. Zohar, and S. Goldberg. Eclipse attacks on bitcoin's peer-to-peer network. In *24th {USENIX} Security Symposium ({USENIX} Security 15)*, pages 129–144, 2015.

[102] L. Heimbach and R. Wattenhofer. Sok: Preventing transaction reordering manipulations in decentralized finance. *arXiv preprint arXiv:2203.11520*, 2022.

[103] S. Henningsen, D. Teunis, M. Florian, and B. Scheuermann. Eclipsing ethereum peers with false friends. In *2019 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 300–309. IEEE, 2019.

[104] C. Herley and P. C. Van Oorschot. Sok: Science, security and the elusive goal of security as a scientific pursuit. In *2017 IEEE Symposium on Security and Privacy (SP)*, pages 99–120, 2017.

[105] E. Hertzog, G. Benartzi, and G. Benartzi. Bancor protocol: continuous liquidity for cryptographic tokens through their smart contracts. *White paper*, 2017.

[106] D. Hopwood, S. Bowe, T. Hornby, and N. Wilcox. Zcash protocol specification. Technical report, Technical report, 2016–1.10. Zerocoin Electric Coin Company, 2016.

[107] D. Y. Huang, H. Dharmdasani, S. Meiklejohn, V. Dave, C. Grier, D. McCoy, S. Savage, N. Weaver, A. C. Snoeren, and K. Levchenko. Botcoin: Monetizing stolen cycles. In *NDSS*, 2014.

[108] initc3.org. Frontrun me, 2018.

[109] H. D. Insight. Price oracle - a must have infrastructure. *URl: https://www.huobidefilabs.io/en/Insight/1313163603254243330/*, 2020.

[110] G. Issue. Method 'decreaseapproval' in unsafe, 2017.

[111] S. C. J. Vermorel, A. Sechet and T. van der Wansem. Canonical transaction ordering for bitcoin, 2018.

[112] N. Johnson. Ethereum domain name service - specification, 2016.

[113] H. Kalodner, S. Goldfeder, X. Chen, S. M. Weinberg, and E. W. Felten. Arbitrum: Scalable, private smart contracts. In *USENIX Security*, 2018.

[114] 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.

[115] G. Kappos, H. Yousaf, M. Maller, and S. Meiklejohn. An empirical analysis of anonymity in zcash. *arXiv preprint arXiv:1805.03180*, 2018.

[116] 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.

[117] M. Kelkar, S. Deb, S. Long, A. Juels, and S. Kannan. Themis: Fast, strong order-fairness in byzantine consensus. Cryptology ePrint Archive, Paper 2021/1465, 2021. `https://eprint.iacr.org/2021/1465`.

[118] M. Kelkar, F. Zhang, S. Goldfeder, and A. Juels. Order-fairness for byzantine consensus. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part III 40*, pages 451–480. Springer, 2020.

[119] D. Khovratovich and J. Law. Sovrin: digital identities in the blockchain era. Technical report, Sovrin, 2017.

[120] M. King, J. Atkins, and M. Schwarz. Internet advertising and the generalized second-price auction: Selling billions of dollars worth of keywords. *The American economic review*, 97(1):242–259, 2007.

[121] M. B. Koch. Exploring cryptokitties part 2 - the cryptomidwives. 2018.

[122] 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.

[123] J. A. Kroll, I. C. Davey, and E. W. Felten. The economics of bitcoin mining, or bitcoin in the presence of adversaries. In *Proceedings of WEIS*, volume 2013, page 11, 2013.

[124] A. Kumar, C. Fischer, S. Tople, and P. Saxena. A traceability analysis of monero's blockchain. In *European Symposium on Research in Computer Security*, pages 153–173. Springer, 2017.

[125] H. Lambur, A. Lu, and R. Cai. Uma data verification mechanism: Adding economic guarantees to blockchain oracles. *Risk Labs, Inc., Tech. Rep., Jul*, 2019.

[126] Langner. The stuxnet story. *URl: https://www.langner.com/2020/07/the-stuxnet-story/*, 2020.

[127] R. Langner. Stuxnet: Dissecting a cyberwarfare weapon. *IEEE Security & Privacy*, 9(3):49–51, 2011.

[128] C. Lesaege, F. Ast, and W. George. Kleros the justice protocol. *URl: https://kleros.io/static/whitepaper_en-8bd3a0480b45c39899787e17049ded26.pdf*, 2019.

[129] I.-C. Lin and T.-C. Liao. A survey of blockchain security issues and challenges. *IJ Network Security*, 19(5):653–659, 2017.

[130] LiveHelpNow. Security incident nov 23rd, 2017. `https://blog.livehelpnow.net/security-incident-nov-23rd-2017/`, 2017. Accessed: 2017-12-14.

[131] S. K. Lo, X. Xu, M. Staples, and L. Yao. Reliability analysis for blockchain oracles. *Computers & Electrical Engineering*, 83:106582, 2020.

[132] LongForWisdom. Flash loans and securing the maker protocol. *URl: https://forum.makerdao.com/t/urgent-flash-loans-and-securing-the-maker-protocol/4901*, 2020.

[133] S. Madnick and H. Zhu. Improving data quality through effective use of data semantics. *Data & Knowledge Engineering*, 59(2):460–475, 2006.

[134] MakerDAO. Introducing oracles v2 and defi feeds, 2019.

[135] K. Malinova and A. Park. Market design with blockchain technology, 2017.

[136] D. Maram, H. Malvai, F. Zhang, N. Jean-Louis, A. Frolov, T. Kell, T. Lobban, C. Moy, A. Juels, and A. Miller. Candid: can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 1348–1366. IEEE, 2021.

[137] Y. Marcus, E. Heilman, and S. Goldberg. Low-resource eclipse attacks on ethereum's peer-to-peer network. Cryptology ePrint Archive, Report 2018/236, 2018.

[138] J. W. Markham. Front-running-insider trading under the commodity exchange act, 1988.

[139] G. Maxwell. Confidential transactions. URL: https://people.xiph.org/˜greg/confidential_values.txt (Accessed 09/05/2016), 2015.

[140] P. McCorry. Enablement of mev and the morals of extracting, 2023.

[141] P. McCorry, A. Hicks, and S. Meiklejohn. Smart contracts for bribing miners. IACR Cryptology ePrint Archive, 2018.

[142] 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.

[143] E. Medvedev. Python scripts for etl (extract, transform and load) jobs for ethereum blockshere is the bibtex file.

[144] S. Micali, M. Rabin, and S. Vadhan. Verifiable random functions. In *40th annual symposium on foundations of computer science (cat. No. 99CB37039)*, pages 120–130. IEEE, 1999.

[145] T. Micro. Malvertising campaign abuses google's doubleclick to deliver cryptocurrency miners. `https://blog.trendmicro.com/trendlabs-security-intelligence/malvertising-campaign-abuses-googles-doubleclick-to-deliver-cryptocurrency-miners/`, 2018. Accessed: 2018-01-31.

[146] I. Miers, C. Garman, M. Green, and A. D. Rubin. Zerocoin: Anonymous distributed e-cash from bitcoin. In *Security and Privacy (SP), 2013 IEEE Symposium on*, pages 397–411. IEEE, 2013.

[147] A. Miller, M. Moeser, K. Lee, and A. Narayanan. An Empirical Analysis of Linkability in the Monero Blockchain. Technical report, arXiv, 2017.

[148] A. Miller, M. Möser, K. Lee, and A. Narayanan. An empirical analysis of linkability in the monero blockchain. *arXiv preprint*, 1704, 2017.

[149] A. Moin, K. Sekniqi, and E. G. Sirer. Sok: A classification framework for stablecoin designs. In *Financial Cryptography*, 2020.

[150] S. Mondal, K. P. Wijewardena, S. Karuppuswami, N. Kriti, D. Kumar, and P. Chahal. Blockchain inspired rfid-based information architecture for food supply chain. *IEEE Internet of Things Journal*, 6(3):5803–5813, 2019.

[151] Monero. MONERO private digital currency. https://getmonero.org/, 2014. Accessed: 2017-11-20.

[152] J. H. Moor. What is computer ethics? *Metaphilosophy*, 16(4):266–275, 1985.

[153] S. Moosavi and J. Clark. Ghazal: toward truly authoritative web certificates using ethereum. In *Financial Cryptography and Data Security: FC 2018 International Workshops, BITCOIN, VOTING, and WTSC, Nieuwpoort, Curaçao, March 2, 2018, Revised Selected Papers 22*, pages 352–366. Springer, 2019.

[154] A. Müller and M. Grandi. Weather derivatives: a risk management tool for weather-sensitive industries. *The Geneva Papers on Risk and Insurance. Issues and Practice*, 25(2):273–287, 2000.

[155] T. Mursch. Cryptojacking malware coinhive found on 30,000+ websites. https://badpackets.net/cryptojacking-malware-coinhive-found-on-30000-websites/, 2017. Accessed: 2018-01-20.

[156] N. Mutual. Responsible vulnerability disclosure. *URl: https://medium.com/nexus-mutual/responsible-vulnerability-disclosure-ece3fe3bcefa*, 2020.

[157] S. Nakamoto et al. Bitcoin: A peer-to-peer electronic cash system. 2008.

[158] NakedSecurity. Unsecured aws led to cryptojacking attack on la times. https://nakedsecurity.sophos.com/2018/02/27/unsecured-aws-led-to-cryptojacking-attack-on-la-times/, 2018. Accessed: 2018-02-28.

[159] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder. *Bitcoin and Cryptocurrency Technologies: A Comprehensive Introduction*. Princeton University Press, 2016.

[160] NEST. Nest protocol: A distributed price oracle network. *URl: https://nestprotocol.org/doc/ennestwhitepaper.pdf*, 2020.

[161] T. Neudecker, P. Andelfinger, and H. Hartenstein. A simulation model for analysis of attacks on the bitcoin peer-to-peer network. In *2015 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, pages 1327–1332. IEEE, 2015.

[162] T. Neudecker and H. Hartenstein. Short paper: An empirical analysis of blockchain forks in bitcoin. In *International Conference on Financial Cryptography and Data Security*, pages 84–92. Springer, 2019.

[163] S. Noether. Ring signature confidential transactions for monero. Cryptology ePrint Archive, Report 2015/1098, 2015. https://eprint.iacr.org/2015/1098.

[164] N. J. D. of Consumer Affairs. New jersey division of consumer affairs obtains settlement with developer of bitcoin-mining software found to have accessed new jersey computers without users knowledge or consent. `http://nj.gov/oag/newsreleases15/pr20150526b.html`, 2015. Accessed: 2018-01-20.

[165] Opera. New year, new browser. opera 50 introduces anti-bitcoin mining tool. `http://blogs.opera.com/desktop/2018/01/opera-50-introduces-anti-bitcoin-mining-tool/`, 2018. Accessed: 2018-01-20.

[166] C. Oracle. Explicit staking in chainlink 2.0: An overview. *URl: https://blog.chain.link/explicit-staking-in-chainlink-2-0/*, 2021.

[167] PeckShield. bzx hack ii full disclosure (with detailed profit analysis), 2020.

[168] I. G. A. Pernice, S. Henningsen, R. Proskalovich, M. Florian, and H. Elendner. Monetary stabilization in cryptocurrencies: Design approaches and open questions. In *CVCBT*, 2019.

[169] J. Peterson and J. Krug. Augur: a decentralized, open-source platform for prediction markets. *arXiv preprint arXiv: 1501.01042*, 2015.

[170] C. Petty. A look at the status.im ico token distribution, 2017. URL: https://medium.com/blockchannel/a-look-at-the-status-im-ico-token-distribution-d0f8a905cb2c.

[171] C. Pierrot and B. Wesolowski. Malleability of the blockchain's entropy. *Cryptography and Communications*, 10(1):211–233, 2018.

[172] E. Pimentel, E. Boulianne, S. Eskandari, and J. Clark. Systemizing the challenges of auditing blockchain-based assets. *Journal of Information Systems*, 35(2):61–75, 2021.

[173] E. Piqueras. Generalized ethereum frontrunners, an implementation and a cheat. 2019.

[174] PoolTogether. Pooltogether is a no-loss, audited savings game powered by blockchain technology. *URl: https://www.pooltogether.com/*, 2020.

[175] B. Protocol. Bandchain whitepaper. *URl: https://docs.bandchain.org/whitepaper/system-overview.html*, 2020.

[176] K. Qin, L. Zhou, B. Livshits, and A. Gervais. Attacking the defi ecosystem with flash loans for fun and profit. *arXiv preprint arXiv:2003.03810*, 2020.

[177] K. Qin, L. Zhou, B. Livshits, and A. Gervais. Attacking the defi ecosystem with flash loans for fun and profit. In *International conference on financial cryptography and data security*, pages 3–32. Springer, 2021.

[178] H. Qureshi. Introducing cofix: a next-generation amm. *URl: https://medium.com/dragonfly-research/introducing-cofix-a-next-generation-amm-199aea686b6b*, 2020.

[179] R. Radner and A. Schotter. The sealed-bid mechanism: An experimental study. *Journal of Economic Theory*, 48(1):179–220, 1989.

[180] R. Rahimian. Multiple withdrawal attack. 2018.

[181] C. Reitwiessner. An update on integrating zcash on ethereum (zoe). *URL: https://blog.ethereum.org/2017/01/19/update-integrating-zcash-ethereum/*, 2017.

[182] M. Rosenfeld. Analysis of bitcoin pooled mining reward systems. *arXiv preprint arXiv:1112.4980*, 2011.

[183] S. Ruoti, B. Kaiser, A. Yerukhimovich, J. Clark, and R. Cunningham. Blockchain technology: What is it good for? *Communications of the ACM*, 63(1):46–53, 2020.

[184] S. Ruwhof. Massive child porn site is hiding in plain sight, and the owners behind it. `https://sijmen.ruwhof.net/weblog/1782-massive-child-porn-site-is-hiding-in-plain-sight-and-the-owners-behind-it`, 2017. Accessed: 2018-01-20.

[185] G. Rydstedt, E. Bursztein, D. Boneh, and C. Jackson. Busting frame busting: a study of clickjacking vulnerabilities at popular sites. In *IEEE SSP*, volume 2, 2010.

[186] M. Salehi, J. Clark, and M. Mannan. Red-black coins: Dai without liquidations. In *Financial Cryptography: DeFi*, 2021.

[187] S. Sallam and B. D. Beheshti. A survey on lightweight cryptographic algorithms. In *TENCON 2018-2018 IEEE Region 10 Conference*, pages 1784–1789. IEEE, 2018.

[188] E. B. Sasson, A. Chiesa, C. Garman, M. Green, I. Miers, E. Tromer, and M. Virza. Zerocash: Decentralized anonymous payments from bitcoin. In *2014 IEEE Symposium on Security and Privacy (SP)*, pages 459–474. IEEE, 2014.

[189] A. Shevchenko. Uniswap and automated market makers, explained. *URl: https://cointelegraph.com/explained/uniswap-and-automated-market-makers-explained*, 2020.

[190] D. Sillars. The performance impact of cryptocurrency mining on the web. `https://discuss.httparchive.org/t/the-performance-impact-of-cryptocurrency-mining-on-the-web/1126`, 2017. Accessed: 2017-12-20.

[191] O. Solmaz. The anatomy of a block stuffing attack. *URL: https://osolmaz.com/2018/10/18/anatomy-block-stuffing/*, 2018.

[192] K. Sonar and H. Upadhyay. A survey: Ddos attack on internet of things. *International Journal of Engineering Research and Development*, 10(11):58–63, 2014.

[193] A. Sotirakopoulos, K. Hawkey, and K. Beznosov. On the challenges in usable security lab studies: Lessons learned from replicating a study on SSL warnings. In *SOUPS*, 2011.

[194] J. Sunshine, S. Egelman, H. Almuhimedi, N. Atri, and L. F. Cranor. Crying wolf: An empirical study of SSL warning effectiveness. In *USENIX Security*, 2009.

[195] Synthetix. Synthetix response to oracle incident. *URl: https://blog.synthetix.io/response-to-oracle-incident/*, 2019.

[196] Synthetix. A derivatives liquidity protocol. *URl: https://www.synthetix.io/*, 2020.

[197] P. Sztorc. Truthcoin. *peer-to-peer oracle system and prediction marketplace.*, 2015.

[198] R. Tahir, M. Huzaifa, A. Das, M. Ahmad, C. Gunter, F. Zaffar, M. Caesar, and N. Borisov. Mining on someone else's dime: Mitigating covert mining operations in clouds and enterprises. In *RAID*, 2017.

[199] C. team. Cryptokitties, 2018. Accessed: 2018-08-31.

[200] S. Team. The status network, a strategy towards mass adoption of ethereum, 2017. Accessed: 2018-06-10.

[201] Tellor. A decentralized Oracle.

[202] Tellor-io. Eip-2362: Pull oracle interface. *URl: https://github.com/tellor-io/EIP-2362*, 2020.

[203] TheGuardian. Ads dont work so websites are using your electricity to pay the bills. `https://www.theguardian.com/technology/2017/sep/27/pirate-bay-showtime-ads-websites-electricity-pay-bills-cryptocurrency-bitcoin`, 2017. Accessed: 2017-11-20.

[204] ThePirateBay. The galaxys most resilient bittorrent site. `https://thepiratebay.org/blog/242`, 2017. Accessed: 2017-11-20.

[205] TheRegister. Cbs's showtime caught mining crypto-coins in viewers' web browsers. `https://www.theregister.co.uk/2017/09/25/showtime_hit_with_coinmining_script/`, 2017. Accessed: 2018-01-20.

[206] TheRegister. Crypto-jackers enlist google tag manager to smuggle altcoin miners. https://www.theregister.co.uk/2017/11/22/cryptojackers_google_tag_manager_coin_hive/, 2017. Accessed: 2018-01-20.

[207] TheRegister. Lets get ready to grumble! ufc secretly choke slams browsers with monero miners. https://www.theregister.co.uk/2017/11/07/ufc_coin_hive/, 2017.

[208] TheRegister. Uk ico, uscourts.gov... thousands of websites hijacked by hidden crypto-mining code after popular plugin pwned. https://www.theregister.co.uk/2018/02/11/browsealoud_compromised_coinhive/, 2018. Accessed: 2018-02-28.

[209] TheVerge. Hotel caught injecting advertising into webpages on complimentary wi-fi network. https://www.theverge.com/2012/4/7/2931600/hotel-caught-injecting-advertising-into-web-pages-on-complimentary-wi, 2012. Accessed: 2017-12-08.

[210] TheVerge. Showtime websites secretly mined user cpu for cryptocurrency. https://www.theverge.com/2017/9/26/16367620/showtime-cpu-cryptocurrency-monero-coinhive, 2017. Accessed: 2017-11-20.

[211] F. Tian. An agri-food supply chain traceability system for china based on rfid & blockchain technology. In *2016 13th international conference on service systems and service management (ICSSSM)*, pages 1–6. IEEE, 2016.

[212] TLSnotary. A mechanism for independently audited https sessions. *URl: https://tlsnotary.org/TLSNotary.pdf*, 2014.

[213] C. F. Torres, R. Camino, et al. Frontrunner jones and the raiders of the dark forest: An empirical study of frontrunning on the ethereum blockchain. In *30th USENIX Security Symposium (USENIX Security 21)*, pages 1343–1359, 2021.

[214] UMA. Uma data verification mechanism: Adding economic guarantees to blockchain oracles. *URl: https://github.com/UMAprotocol/whitepaper/blob/master/UMA-DVM-oracle-whitepaper.pdf*, 2020.

[215] Uniswap. Uniswap oracle. *URl: https://uniswap.org/docs/v2/core-concepts/oracles*, 2020.

[216] N. van Saberhagen. Cryptonote v 2. 0. https://bytecoin.org/old/whitepaper.pdf, 2013.

[217] M. Vasek, M. Thornton, and T. Moore. Empirical analysis of denial-of-service attacks in the bitcoin ecosystem. In *International conference on financial cryptography and data security*, pages 57–71. Springer, 2014.

[218] R. Ver and J. Wu. Bitcoin cash planned network upgrade is complete, 2018.

[219] 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.

[220] WallStreetJournal. Your computer may be making bitcoin for hackers. `https://www.wsj.com/articles/hackers-latest-move-using-your-computer-to-mine-bitcoin-1509102002`, 2017. Accessed: 2018-01-20.

[221] T. Walther. Multi-token batch auctions with uniform clearing prices. 2018.

[222] Y. Wang. Automated market makers for decentralized finance (defi). *arXiv preprint arXiv:2009.01676*, 2020.

[223] W. Warren. griefing and the perils of virtual settlement, 2017. URL: https://blog.0xproject.com/front-running-griefing-and-theperils-of-virtual-settlement-part-1-8554ab283e97.

[224] W. Warren and A. Bandeali. An open protocol for decentralized exchange on the ethereum blockchain, 2017. URL: https://github.com/0xProject/whitepaper.

[225] Washingtonpost. Hackers have turned politifact's website into a trap for your pc. `https://www.washingtonpost.com/news/the-switch/wp/2017/10/13/hackers-have-turned-politifacts-website-into-a-trap-for-your-pc`, 2017. Accessed: 2018-01-20.

[226] whiterabbit. Black thursday for makerdao: $8.32 million was liquidated for 0 dai, 2020.

[227] V. Wikipedia. Important milestones of the bitcoin project. `https://en.bitcoin.it/wiki/Category:History`, 2009. Accessed: 2018-01-18.

[228] D. Z. J. Williamson. The aztec protocol. *URL: https://github.com/AztecProtocol/AZTEC/*, 2018.

[229] A. D. Wood and J. A. Stankovic. Denial of service in sensor networks. *computer*, 35(10):54–62, 2002.

[230] G. Wood et al. Ethereum: A secure decentralised generalised transaction ledger. Technical Report 2014, Ethereum, 2014.

[231] M. F. Worboys and S. M. Deen. Semantic heterogeneity in distributed geographic databases. *ACM Sigmod Record*, 20(4):30–34, 1991.

[232] J. Wyke. The zeroaccess botnet: Mining and fraud for massive financial gain. *Sophos Technical Paper*, 2012.

[233] Zaugust. Cofix: A computable trading system. *URl: https://cofix.io/doc/Trading_Compensation_CoFiX.pdf*, 2020.

[234] P. J. Zelbst, K. W. Green, V. E. Sower, and P. L. Bond. The impact of rfid, iiot, and blockchain technologies on supply chain transparency. *Journal of Manufacturing Technology Management*, 2019.

[235] 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.

[236] F. Zhang, E. Cecchetti, K. Croman, A. Juels, and E. Shi. Town crier: An authenticated data feed for smart contracts. In *Proceedings of the 2016 aCM sIGSAC conference on computer and communications security*, pages 270–282. ACM, 2016.

[237] F. Zhang, S. K. D. Maram, H. Malvai, S. Goldfeder, and A. Juels. Deco: Liberating web data using decentralized oracles for tls. *arXiv preprint arXiv:1909.00938*, 2019.

[238] S. Zhang and J.-H. Lee. Double-spending with a sybil attack in the bitcoin decentralized network. *IEEE Transactions on Industrial Informatics*, 15(10):5715–5722, 2019.

[239] H. Zhu. Do dark pools harm price discovery? *The Review of Financial Studies*, 27(3):747–789, 2014.