TU WIEN Informatics

Race to the Door

Eine Goldfingerattacke auf Proof of Work Kryptowährungen

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering und Internet Computing

eingereicht von

Ing. Andreas Rosegger, Bsc.
Matrikelnummer 01029049

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar R. Weippl
Mitwirkung: Univ.Lektor Dipl.-Ing. Aljosha Judmayer

Wien, 19. Mai 2021

Andreas Rosegger | Edgar R. Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

TU WIEN Informatics

Race to the Door

A Goldfinger Attack on Proof of Work Cryptocurrencies

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieur

in

Software Engineering and Internet Computing

by

Ing. Andreas Rosegger, Bsc.
Registration Number 01029049

to the Faculty of Informatics

at the TU Wien

Advisor: Privatdoz. Mag.rer.soc.oec. Dipl.-Ing. Dr.techn. Edgar R. Weippl
Assistance: Univ.Lektor Dipl.-Ing. Aljosha Judmayer

Vienna, 19th May, 2021

Andreas Rosegger Edgar R. Weippl

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.at

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Erklärung zur Verfassung der Arbeit

Ing. Andreas Rosegger, Bsc.

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 19. Mai 2021

Andreas Rosegger

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Danksagung

Danke an meine Familie, die mich über ein Jahrzehnt bei diesem Abenteuer unterstützt hat.

Ich möchte Sarah Brunner danken, für ihre ständige Motivation und Unterstützung während dieser herausfordernden Zeiten. Danke, dass du mir zugehört hast, wie ich stundenlang versucht habe, den Sinn in diesem und anderen Themen zu finden.

Als Nächstes möchte ich meinem Assistenzbetreuer Aljoscha Judmayer dafür danken, dass er mir nicht das Thema gab, das ich wollte, sondern das, das ich brauchte. Ich möchte mich für die hervorragende Betreuung und den ständigen Input bedanken.

Außerdem möchte ich mich bei meiner Firma Iteratec dafür bedanken, dass sie mich freigestellt hat, damit ich mich auf den Abschluss meines Studiums konzentrieren kann. Danke an alle meine Kollegen, die Aufgaben übernommen haben und mir den Rücken freigehalten haben.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Acknowledgements

Thanks to my family, who supported me for over a decade in this adventure.

I want to thank Sarah Brunner for her constant motivation and support during these challenging times. Thank you for listening to me spend hours attempting to make sense of this and other topics.

Next, I want to thank my assistant supervisor, Alyosha Judmayer, for giving me not the topic I wanted but the one I needed. I would like to express my gratitude for the excellent support and constant input.

I would also like to thank my company Iteratec for giving me the time off to focus on finishing my studies. Thanks to all my colleagues who took over responsibilities and kept my back free.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Kurzfassung

Goldfinger-Attacken zielen darauf ab, den Wert einer Ziel-Kryptowährung zum Absturz zu bringen, indem die Mehrheit der Stimmrechte im System genutzt wird das zu Grunde liegende Konsensprotokoll untergraben. In einem Proof-of-Stake-Kontext, in dem die Stimmkraft auf der Menge der gehaltenen Kryptowährung basiert, kann dies in Form eines Buy-out-Angriffs erreicht werden, bei dem eine Mehrheit der Zielwährung gekauft wird. In diesem Zusammenhang wurde der Race to the Door (RTTD)-Effekt beschrieben, der dazu führt, dass immer mehr Inhaber aus der Zielwährung aussteigen, bevor diese wertlos wird. Dieser Effekt senkt den Preis für weitere Stimmenanteile, wodurch der Angriff billiger wird, je weiter er fortschreitet. Diese Arbeit soll zeigen, dass ein Angriff im Stil von Race to the Door auch in einem Proof-of-Work (PoW)-Kontext technisch möglich ist, ohne eine Mehrheit der Stimmrechte (d. h. der Hash-Rate) zu erlangen. Zu diesem Zweck werden die technische Machbarkeit und die Kosten eines solchen Angriffs am Beispiel von Ethereum untersucht.

Zunächst wird ein Systemmodell für RTTD-Angriffe auf PoW-basierte Kryptowährungen vorgestellt, um einen Überblick zu geben. Der Angriff wird dabei in die Phasen Vorbereitung, Rennen und Angriff unterteilt. Um die technische Machbarkeit zu demonstrieren, werden diese Phasen in Form von Smart Contracts auf Ethereum umgesetzt. Für die Angriffsphase werden drei Varianten vorgestellt, die jeweils einen unterschiedlichen Denial-of-Service-Angriff realisieren. Dazu werden In-Band-Zahlungen genutzt, um entweder das Auslösen zusätzlicher Transaktionen oder die Erzeugung leerer Blöcke durch Miner zu incentivieren.

Um die Kosten der vorgeschlagenen Angriffsvarianten abzuschätzen, wird eine empirische Analyse durchgeführt, bei der Transaktionsdaten von historischen Überlastungsphasen der Ethereum-Blockchain untersucht werden. Anhand der Ergebnisse werden die Kosten der Angriffsvarianten geschätzt und verglichen. Die stündlichen Kosten für das Blockieren von Transaktionen durch das auslösen weiterer Transaktionen betragen etwa 870 Ether. Die Incentivierung von Minern, die ein Drittel der Blöcke leer lassen, kostet etwa 790 Ether pro Stunde.

Die Arbeit zeigt, dass Race to the Door-Attacken auch im Kontext von PoW-basierten Kryptowährungen technisch durchführbar sind. Die Kosten des Angriffs hängen dabei von seiner Intensität und Dauer ab. Die Intensität kann in der Angriffsphase konfiguriert werden und die Dauer hängt von der für den Angriff verfügbaren Geldmenge ab.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Abstract

Goldfinger attacks aim to crash the value of a target cryptocurrency by undermining the underlying consensus protocol, utilizing the majority of the voting power in the system. In a Proof of Stake (PoS) context, where voting power is based on the amount of cryptocurrency held, this can be achieved in the form of a buy-out attack, where a majority of the target currency is purchased. In this context, the Race to the Door (RTTD) effect was described, which leads to more and more holders exiting the target currency before it becomes worthless. This effect reduces the price of further voting stake, rendering the attack cheaper the more it progresses. This thesis aims to show that a Race to the Door style attack is also technically possible in a Proof of Work (PoW) context without acquiring a majority of the voting power (i.e., hash rate). For this purpose, the technical feasibility and cost of such an attack are investigated on the example of Ethereum.

First, a system model for RTTD attacks on PoW cryptocurrencies is presented to provide an overview. The attack is thereby divided into the phases preparation, race and attack. To demonstrate the technical feasibility, these phases are implemented in the form of smart contracts on Ethereum. For the attack phase, three variants are presented, each realizing a different denial of service attack. For this purpose, in-band payments are used to incentivize either the triggering of additional transactions or the creation of empty blocks by miners.

To estimate the costs of the proposed attack variants, an empirical analysis is performed, where transaction data of historical congestion phases of the Ethereum blockchain are examined. Based on the results, the estimated costs of the attack variants are calculated and compared. The hourly costs of Blocking transactions using transaction triggering are around 870 Ether. Incentivizing miners to leave one-third of the blocks empty costs about 790 Ether per hour.

The thesis shows that Race to the Door attacks are also technically feasible in the context of PoW cryptocurrencies. The cost of the attack hereby depends on its intensity and duration. The intensity can be configured at the attack phase, and the duration depends on the amount of currency available for the attack.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Contents

Kurzfassung xi

Abstract xiii

Contents xv

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Problem Statement . . . 2
- 1.3 Aim of the Work . . . 3
- 1.4 Structure of the Work . . . 3

2 Background and State of the Art 5
- 2.1 Cryptocurrencies . . . 5
- 2.2 Consensus Protocols . . . 5
- 2.3 Smart Contracts . . . 6
- 2.4 Transactions and Fees . . . 6
- 2.5 Transaction Ordering . . . 6
- 2.6 Goldfinger Attacks . . . 7
- 2.7 Race to the Door . . . 8

3 Attack Model 9
- 3.1 Actors and Terminology . . . 9
- 3.2 Aim and Approach . . . 10
- 3.3 Preparation Phase . . . 11
- 3.4 Race Phase . . . 11
- 3.5 Attack Phase . . . 13

4 Implementation on Ethereum 17
- 4.1 Preparation Phase . . . 17
- 4.2 Race Phase . . . 18
- 4.3 Attack Phase . . . 23

5 Feasibility and Cost Analysis 39

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5.1 Ethereum Data & Data Model 39
5.2 Transaction Ordering on Ethereum 40
5.3 Transaction Blocking Costs 42
5.4 GasPrice Development under Congestion 46
5.5 Estimated Cost Comparison 55

6 Conclusion and Future Work **59**
6.1 Future Work 61

List of Figures **63**

List of Tables **65**

Acronyms **67**

Bibliography **69**

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 1

Introduction

This chapter shows the increase in the relevance of cryptocurrencies at present. It identifies the parties involved in the use of cryptocurrencies and the goals they are pursuing. Subsequently, problems are identified that stand in the way of these stakeholders to achieve their goals. Thereafter, research questions are defined, which will be addressed in this thesis. Finally, the structure of the work is laid out.

1.1 Motivation

As of March 2021 Bitcoin [1] and Ethereum [2] together have reached a market capitalization of around $1,142bn [3]. One year ago, market capitalization was only at $92bn and had grown over 1100%.

Decentralized Finance (DeFi) is another development in cryptocurrencies that has been gaining momentum lately. DeFi is a term for Decentralized Applications (DApps) providing financial services on blockchains via Smart Contracts. Most of them run on the Ethereum blockchain, and they currently lock a value of about $40bn, growing over 3900% compared to the year before [4].

The value growth attracts further interest. Tesla has recently announced to exchange 1.5bn of its cash into Bitcoin and may start to accept Bitcoin as a form of payment for their products [5]. Also, institutional investors like ARK Invest [6], who focus on disruptive technologies and have assets under management of $50bn [7], are investing in cryptocurrencies. The investors are primarily concerned with increasing or at least maintaining the value of their holdings. DeFi users, on the other hand, need a stable base system to conduct their financial transactions.

Another noteworthy event has recently been observed in the context of GameStop, and wallstreetbets [8]. A user of the online forum had noticed that the GameStop share was heavily shorted and, in his opinion, undervalued [9]. The expiration of these short

positions leads to the forced purchase of the share at the current price. The forum users took advantage of these circumstances and bought an increasing number of shares in the company in an attempt to trigger a short squeeze. Finally, the attack was interrupted by the blocking of buy orders on retail investor platforms like Robinhood [10]. This scheme could be described as synchronizing via digital means to attack a financial contract.

All these aspects highlight that enormous values are held and transferred with cryptocurrencies today. Consequently, any attack that would put these values at risk deserves closer investigation.

1.2 Problem Statement

There have been attacks on these cryptocurrencies since their early beginnings. Starting with the publication of the Bitcoin whitepaper in 2008, the possibility of double-spending attacks was addressed [1]. Some other strategies like selfish-mining [11] have been proposed to increase mining profits. Recently a broad range of attacks directly targeting the incentives of participants was summarized under the term Algorithmic Incentive Manipulation (AIM) attacks[12]. In these attacks, the adversary aims to manipulate the incentives of rational actors participating in cryptocurrency protocols.

One example of such an attack is the Goldfinger attack, first introduced by Kroll et al. [13]. This attack was named after the villain Auric Goldfinger from a James Bond movie [14]. His goal was to destroy the gold reserves in Fort Knox to increase the value of his gold reserves. The idea behind this attack was applied to cryptocurrencies. By gaining the majority of voting power, one would be empowered to block all transactions and render a cryptocurrency useless. Further approaches have been proposed which can implement a Goldfinger attack by obtaining the majority of voting power [15], including PoS-based cryptocurrencies. A first practical example of the Goldfinger attack was the GoldfingerCon [16], which uses a smart contract to incentivize Bitcoin miners to submit empty blocks by offering a reward.

The consequences of such a Goldfinger attack are severe for those affected. Holders of the cryptocurrency would suffer financial damage due to the loss of value. Furthermore, DeFi users would not be able to continue using these cryptocurrencies as a platform for their financial transactions.

Another variant of these Goldfinger attacks is the *Race to the Door* attack which has only been sketched so far [15][12]. The unique feature of the attack is that it builds funds in the targeted cryptocurrency to destroy it. Generally, Goldfinger attacks, and especially the Race to the Door attack, have not yet been extensively covered by academic research in this area.

So far, the Race to the Door attack has been mentioned as a buy-out attack in a PoS context [15], where more than 51% of the currency has to be bought to gain control of the consensus protocol. Only lately, this sketch was extended, and the Race to the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Door attack was considered to also work in a PoW setting, and without the need of the majority of the voting power [12].

This work aims to further explore the attack landscape in the area of algorithmic incentive manipulation. Therefore, the research questions are the following:

- RQ1: How can Race to the Door attacks technically be constructed for PoW-based cryptocurrencies?
- RQ2: What are the estimated costs associated with such attacks?
- RQ3: By which criteria are transactions ordered, and how can this be exploited to construct in-band transaction ordering attacks?

1.3 Aim of the Work

First, an overview of the Race to the Door attack model will be provided. In this, the actors involved will be introduced. The phases of the attack are described and how the actors participate in them. This part will provide a descriptive model of the attack without concrete technologies.

Next, concrete technologies are selected with which this attack is implemented. The individual phases of the attack will be implemented using smart contracts. Different variations are developed for the attack phase. The resulting implementation will demonstrate the technical feasibility of the attack.

Finally, the work deals with the analysis of the attack. The costs of the attack are considered. This part analyzes the monetary feasibility and compares the presented attack variants.

1.4 Structure of the Work

First, chapter 2 provides background information and state of the art. An overview of the system model of the Race to the Door attack is given in chapter 3, which is intended to provide an overview. This model is then implemented with concrete technologies in chapter 4. Thereby three different attack models are presented. In order to estimate the costs, historical congestion phases are first examined in chapter 5. On this basis, the costs of the attack variants are estimated and compared. Finally, the findings of this work are summarized in chapter 6.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 2

Background and State of the Art

In this section, the required background information for the remainder of this thesis is provided. Furthermore, an overview of related topics in the field of research relevant to this thesis is given.

2.1 Cryptocurrencies

Cryptocurrencies are digital currencies that process transactions through cryptographically secured transactions. These transactions are stored within a ledger in the form of a blockchain. Each of these blocks contains transactions that it has processed and refers to a previous block. All blocks and transactions are publicly visible. By using cryptographic functions, it is possible to check whether a block is correct. In this way, any changes made to processed transactions after the fact would be noticed.

2.2 Consensus Protocols

Unlike banks, new transactions are not processed by any central authority. Instead, they are processed via the Nakamoto consensus protocol. For this purpose, the transactions are distributed throughout the entire network. Miners collect these transactions and form a block. The block includes a reference to the highest block currently known in the network. In order to append the new block to the blockchain, a cryptographic puzzle must be solved. The chance of solving this puzzle depends on the processing power of the miner. Once the miner has solved the puzzle, the new block is published on the network. For solving the puzzle, the miner receives a block reward and the transaction fees for all processed transactions in the block. Other miners verify the published block. If the block is correct, it is considered the highest block and is built upon when forming new blocks.

The above consensus protocol is based on Proof of Work. A nonce is provided as proof of the work done, which is necessary to solve the puzzle. Another possible way of voting is

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Proof of Stake (PoS). In this case, the voting power depends on the amount of currency and how long it is held.

2.3 Smart Contracts

Blockchains such as Bitcoin allow their currency to be sent from a sender address to a recipient address within a transaction. Other blockchains such as Ethereum also allow transactions to be sent to smart contracts. These are programs that can be executed through a transaction. Miners can process these transactions invoking function of smart contract code running on the Ethereum Virtual Machine (EVM). Smart contracts can provide different methods to be called by a transaction and allow parameters to be passed.

2.4 Transactions and Fees

Transaction fees are rewards to miners for processing transactions and are paid by the issuer of the transaction. Each operation of a transaction is associated with a gas cost, and more complex operations have a higher gas price. When the transaction is executed, the gasUsed is calculated by summing up the gas costs of the executed instructions. The gasUsed is then multiplied by the gasPrice and represents the transaction fee in Ether that the miner can claim for himself. The issuer of a transaction has to provide the gas for each transaction, which is an upper limit for the executed instructions. Also the gasPrice to be provided, which is the amount the issuer is willing to pay per gas. The gasPrice is used to compensate for the changing value of Ether, so it falls when the value of Ether rises to keep the resulting transaction costs in check.

2.5 Transaction Ordering

Transaction ordering depends on the client used. There are several possibilities on how to order the transactions within a block. For example, transactions could be ordered by the time they arrive at the client. In the case of the permissionless blockchain Ethereum there exist several clients, each with their default ordering of transactions. Depending on the adoption of the clients used by miners, different orders in the blocks can be observed.

Transactions may depend on each other, which has to be considered in the sorting process. In Ethereum, every transaction contains a nonce. For each transaction issued by the same account, the nonce is incremented by one. By the consensus rules of the underlying protocol, miners must respect the order by nonce for each account, with no gaps allowed between two consecutive nonces. This scheme allows account owners to order the execution of their transactions as their order may of importance.

Clients order transactions to maximize their profits. The algorithms used in the mining client may order transactions by gasPrice to maximize their profits. It was found in

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[17] that 78,3% of transactions are sorted using the Geth [1] based sorting alorithm while 20.2% use Parity [2]. Geth sortes transactions by their gasPrice with respect to the nonces [3]. Parity additionally prioritizes locally submitted transactions [4].

Transaction ordering can be exploited by attacks. As most transactions are ordered by transaction fee, account owners may increase their transaction fee to speed up transaction execution. In a front-running attack, [18, 19] an adversary may place a high fee transaction in the hope to be executed before a lower fee competing transaction.

2.6 Goldfinger Attacks

The goal of the Goldfinger attack is to render an asset useless. The motivation behind doing so is extrinsic and includes appreciating one's own assets or some other external utility. Kroll et al. [13] first introduced Goldfinger as a new class of attacks on cryptocurrencies. In his paper, he refers to Auric Goldfinger, a villain from a James Bond movie [14]. In the movie, Auric holds gold reserves in several locations and tries to eradicate the United States' gold reserves in Fort Knox. By doing so, he aims to increase the value of his holdings. Kroll et al. took the concept to cryptocurrencies and considered an adversary to disrupt Bitcoin and destroy its value to gain some undefined external utility. This could be, for example, that everyone has to use a competing (crypto)currency under the control of the attacker.

As already noted, the motivation to destroy a cryptocurrency is extrinsic. Kroll et al. noted that if the attack were executed successfully, the targeted cryptocurrency would lose its value. Possible holdings of the attacker(s) in the cryptocurrency would therefore also be lost. In conclusion, the attackers' incentive must come from other sources. In [13] three possible motivations are named. First, government institutions may be interested in blocking transactions as cryptocurrencies are suspected to be used in illegal activities. Second, currencies may be blocked as a political message by a non-state attacker. Third, investment gains may be a possible incentive in the form of short positions on the currency. In PoW-based cryptocurrencies, a Goldfinger attack can be executed by attackers with at least 51% of the total hash rate.

Bonneau et al. [15] revisited the Goldfinger attack model. In his works, he assumes such an attack may be constructed by gaining the majority of capacity (the voting power) in the system and describes some common approaches to achieve this (build, rent, bribe, buy). In PoW-based cryptocurrencies such as Bitcoin and at the time of this writing, Ethereum an attacker needs to gain control of 51% of the mining power. This can be

[1] https://geth.ethereum.org/

[2] https://www.parity.io/ethereum/

[3] https://github.com/ethereum/go-ethereum/blob/290e851f57f5d27a1d5f0f7ad784c836e017c337/core/types/transaction.go#L372

[4] https://github.com/openethereum/parity-ethereum/blob/c49beccadcd3c60c9fd90d1393921b91598c8eb0/ethcore/src/miner/transaction_queue.rs#L525

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

done by permanently buying or temporarily renting mining machines. In PoS-based cryptocurrencies such as the planned Ethereum 2.0 [5], the adversary has to gather a majority of the currency itself. In opposite to PoW, in PoS the capacity cannot be borrowed temporarily. Also, the attacker's stake in the currency would render worthless in the event of a successful attack.

The importance in the research of Goldfinger attacks increases with their likelihood. Bonneau et al. [15] state that this class of attack has received relatively little research. However, he also noted that the costs of mounting such an attack are relatively low to the total value in the system, and the motivation to mount such an attack increases as competing currencies rival for adoption.

Goldfinger attacks in the context of cryptocurrencies aim at destroying the value of the currency. The motivation behind this includes the appreciation of an alternative currency under the control of the attacker. Claims state that Goldfinger attacks can be performed without a majority of the voting power. The research in this area is still somewhat modest, but the costs seem to be low related to the expected damage. This thesis will explore other possibilities to launch DoS attacks without the need for 51% of capacity in the system. It will show how rivaling cryptocurrencies motivated in increasing their value may perform such an attack. This aims to show the relevance of this area of research which has yet received little attention.

2.7 Race to the Door

Race to the Door attacks are a special form of Goldfinger attacks. A Race to the Door effect was mentioned as a side effect of the potential buy-out attack on PoS-based cryptocurrencies[15]. In this buy-out attack, the adversary would announce to gather a majority of the targeted cryptocurrency to use it for a 51% attack. If the attacker can convince the holders of the currency that his attempt may be successful, they may be more willing to sell their stake to avoid being left in the remaining 49% and lose all of their value.

This attack type has yet only been sketched, but detailed implementations have not been presented yet. In more recent effort in building a framework for Algorithmic Incentive Manipulation (AIM) attack categorization [12] the Race to the Door attack was brought up for discussion once again. It was found that it had not received any further attention since it was first mentioned. The previous sketch has been extended to PoW systems, and it was considered that a majority of the voting power might not be necessary.

[5] https://ethereum.org/en/eth2/

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 3

Attack Model

This chapter outlines the attack/system model of the Race to the Door attack. First, the currencies involved and their respective holders are presented in section 3.1. Subsequently, the attack goal and its overall approach are briefly presented in section 3.2. Then the individual phases of the attacks are presented in greater detail in sections 3.3, 3.4 and 3.5. Finally, in section 3.5.1, possible approaches of the attack phase are considered.

3.1 Actors and Terminology

For the Race to the Door attack, the currencies C with their respective holders are considered H as shown in figure 3.1.

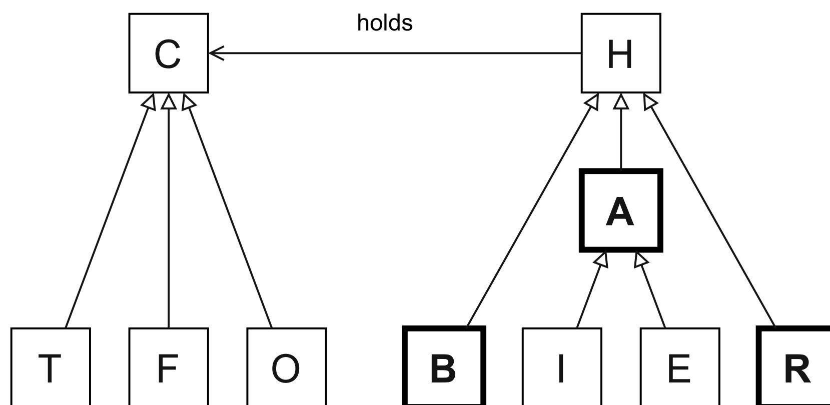

Figure 3.1: Holders and Currencies of the Race to the Door

The currencies C considered in this model are the following:

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

- **Target currency** $\mathcal{T}$ is a cryptocurrency which is aimed to be destroyed by the attack.
- **Funding currency** $\mathcal{F}$ is a cryptocurrency holding the funds sponsoring the attack and is expected to gain value in the event of a successful attack.
- **Other currency** $\mathcal{O}$ is another fiat or cryptocurrency whose value is not affected by the attack.

The currencies described above may be held by the holders $\mathcal{H}$, where a holder could theoretically own more than one currency. These holders are classified according to the BAR model [20] into Byzantine B, Altruistic A and Rational R. However, among the Altruistic holders A, a further distinction is made between the irrational holders I and the exchanging holders E.

- **Byzantine holder** B is the sponsor of the attack against the target cryptocurrency T. He holds funds in cryptocurrency $\mathcal{F}$ whose value he wants to increase with the attack.
- **Irrational holders** I are Altruistic holders owning funds in the target currency $\mathcal{T}$. Like all Altruistic holders, irrational holders are honest and follow the rules of the protocol. Irrational holders cope with the attack by doing nothing. They neither support nor prevent the attack by simply keeping their funds in the target currency. They are willing to risk the loss of value of their holdings and become victims if the attack were to succeed
- **Exchanging holders** E are also Altruistic holders with funds in the target currency $\mathcal{T}$ and take a neutral stance in the attack. Unlike the irrational holders, exchanging holders cope with an impending attack by exchanging their holdings for another fiat or cryptocurrency $\mathcal{O}$. Since the currency $\mathcal{O}$ is considered unaffected by the attack, these holders do neither lose nor gain any profits.
- **Rational holders** R are holders of the target currency $\mathcal{T}$ and act profit-oriented. In case of an attack, they become supporters if a reward is promised to them. This could mean exchanging their holdings in $\mathcal{T}$ for the funding currency $\mathcal{F}$ if they are offered a favorable exchange rate and appreciation.

3.2 Aim and Approach

The ultimate goal of the attack is to destroy the target currency $\mathcal{T}$. In this process, a welcome side effect is the potential appreciation of the funding currency $\mathcal{F}$. Since the funding currency $\mathcal{F}$ represents an alternative to $\mathcal{T}$, the attacker B expects an increase in demand and thus an increase in value. In order to destroy the target currency $\mathcal{T}$, a nested attack in the Race to the Door attack is necessary. This is to be financed by the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

target currency $\mathcal{T}$, which must be collected in the preparation. The attack phases are presented in the following sections, showing how attacker B implements this objective.

3.3 Preparation Phase

In the first phase, the attacker B announces that he will launch an attack on the target currency $\mathcal{T}$. He demonstrates that he has sufficient funds in the currency $\mathcal{F}$. Since $\mathcal{F}$ is a cryptocurrency, the attacker can use cryptographic means to support this statement credibly. In addition, the attacker announces that he is willing to exchange his funds in $\mathcal{F}$ for the targeted cryptocurrency $\mathcal{T}$. To increase its reach, the attacker may resort to using social media to spread its message. This phase aims to attract as much attention as possible and arouse fear about a possible decline in value that will result from the upcoming attack.

3.4 Race Phase

In the race phase, the attacker collects funds in the target currency $\mathcal{T}$ in exchange for its holdings in the funding currency $\mathcal{F}$. The goal of this phase is to achieve the funding goal, which is necessary to carry out the attack. In the following, we will describe how the different holders behave in this phase.

B	I_1 ... I_x	E_1 E_2 E_3 ... E_y	R_1 R_2 R_3 ... R_z
F F F F F F F F	T ... T	T T T ... T	T T T ... T

Figure 3.2: Race Phase - Initial State

The figure 3.2 shows the initial state of the race phase. In the upper line are the respective holders according to the extended *BAR* model. Below are the respective currencies held by the participants. In this simplified example, the attacker B holds several units of the currency $\mathcal{F}$ which he intends to exchange for the target currency $\mathcal{T}$. The irrational holders I, exchanging holders E and rational holders R, each own one unit of the target currency $\mathcal{T}$.

As the race phase progresses, holders exchange their holdings for other currencies. Figure 3.2 shows an updated state in this process highlighting these changes. Attacker B could exchange two of his units from $\mathcal{F}$ for units from $\mathcal{T}$ with R_1 and R_2. The exchanging holders E_1 and E_2 exchanged their units of $\mathcal{T}$ for another currency $\mathcal{O}$ to escape the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

B	I_1 ... I_x	E_1 E_2 E_3 ... E_y	R_1 R_2 R_3 ... R_z
T T	T ... T	O O T ... T	F F T ... T
F F			
F F			
F F			

Figure 3.3: Race Phase - Progress State

effects of the attack. All irrational holders I continue to hold their units of $\mathcal{T}$ as they are not willing to exchange them.

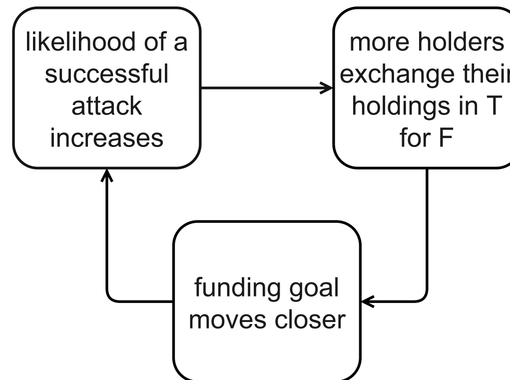

Figure 3.4: Race Phase - Race to the Door Effect

The more units of $\mathcal{T}$ the attacker can collect, the closer he gets to his funding goal increases the likelihood of a successful attack. This, in turn, encourages more rational holders R to exchange their holdings of $\mathcal{T}$ for $\mathcal{F}$, thereby causing a vicious cycle that enables the attacker to achieve his goal even faster. This is the Race to the Door effect that is driving the attack and is illustrated on figure 3.4.

Finally, the funding goal is reached with the resulting status of the race phase of the example shown in figure 3.5. In our simplified illustration, six units of $\mathcal{T}$ were needed to launch an attack. The attacker was able to obtain these units by exchanging the rational holders' units of $\mathcal{T}$ for $\mathcal{F}$. Also, the exchanging holders E continued to exchange into the currency $\mathcal{O}$ to avoid the attack. However, the irrational holders I remained invested in the target currency $\mathcal{T}$.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Figure 3.5: Race Phase - Final State

3.5 Attack Phase

Reaching the funding goal triggers the final phase of the Race to the Door attack. This attack phase uses the previously collected funds to finance an attack. The goal of the attack is to render the target currency unusable. A denial of service attack would be a suitable option for this purpose.

The impact of the attack on the stakeholders would be devastating. As the target currency is rendered useless by the attack, there may be a loss of value. The remaining holders fall victim to the attack.

The destruction of the target currency $\mathcal{T}$ also has other implications. Affected users of the target currency $\mathcal{T}$ may choose to switch to the funding currency $\mathcal{F}$ as it may be a viable alternative to the destroyed currency. As a result, the value of $\mathcal{F}$ might continue to rise. This has been the goal of the Race to the Door attack from the beginning. The remaining holdings of attacker B in funding currency $\mathcal{F}$ could justify the effort through the increase in its value.

The severity of the attack depends on two factors. The first factor is the proportion of a block that is blocked. The larger this percentage, the more devastating the attack. The second factor is the duration of the attack, i.e. the number of blocks. The longer it lasts, the more devastating the attack. However, also important is the combination of the two factors. Thus, an attack with a high percentage of blocked transactions and low duration has a fairly low severity. Nevertheless, also an attack with a low percentage and high duration has only a limited impact.

Another thing to consider is the cost of the attack. The attacker wants the greatest possible impact with the lowest cost and to use the collected funds optimally. Blocking a large portion of a block will increase costs, which will be analyzed in more detail in a later chapter. Also, the costs increase with the duration of the attack. Thus, the best possible combination of these two factors must be found to sufficiently disrupt the operation of the cryptocurrency but still have a relatively low cost.

The attack phase could be implemented in different ways. In the context of this thesis,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

three different variants of this attack phase are presented. Each of these variants is intended to contribute to the achievement of the attacker's objectives and has different advantages and disadvantages with a different cost structure. However, their common feature is that they provide incentives to hinder the processing capacity of regular transactions within the cryptocurrency. In the following, the two underlying approaches and the categorization of transactions are discussed.

3.5.1 Approaches & Transaction Categorization

The goal of the attack phase is to render the blockchain unusable. Two general approaches to achieve this goal are presented in this section. These approaches use incentives to influence the transactions on the blockchain. In order to better understand and compare the attack variants, the transactions involved in the attack process are categorized.

- **Incentivizing Users to perform transactions** The first approach is to incentivice the uses of the target currency to issue additional transactions. These transactions are intended to create congestion and thus hinder the processing of ordinary transactions. Part of the approach involves offering users a reward for the additional transactions they issue. The FreeMoneyGrab and Fomo10x attacks discussed below implemented this approach in different ways.
- **Incentivizing Miners to exclude transactions** The other approach discussed in this thesis is the targeting of miners. Incentives are set to reward them for not processing transactions. The PayEmptyBlocks attack variant implements this approach.

Transaction Categories

Considering the transactions that enter the blocks during the attack or remain in the mempool, they can be divided into four categories:

- **Rewarded triggered Transactions** are transactions that users issue due to the set incentives that promise rewards in return.
- **Unrewarded triggered Transactions** are also transactions that users triggered as a result of the set incentives but do not receive a reward.
- **Unaffected ordinary Transactions** are transactions issued by users regardless of incentives and are properly executed in a block.
- **Affected ordinary Transactions** are transactions issued by users regardless of incentives but are negatively affected by the attack, i.e., delayed execution.

A primary distinction is made between ordinary transactions, which would have been carried out regardless of the incentives, and triggered transactions, which are added

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

due to the incentives that have been set. Since the space in a block is limited, these triggered transactions displace some of the ordinary transactions referred to as affected ordinary transactions. The transactions that are not displaced from the blocks are called unaffected ordinary transactions. One distinguishes between rewarded and unrewarded between the triggered transactions depending on whether they are compensated or not.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

CHAPTER 4

Implementation on Ethereum

With the general attack model in the last chapter, an overview of the Race to the Door attack was given. In order to examine the technical feasibility of the attack, concrete technologies are selected with which the attack is implemented. First, a scenario is described in which a Race to the Door attack may be conceivable. This is followed by a presentation of the technical implementation of the respective phases of the Race to the Door attack. For the attack phase, three variants are presented to incentivize additional transactions to create congestion or incentivize miners to create empty blocks.

In the following, the possibility of an attack between two smart-contract-based cryptocurrencies is considered. Specifically, an attack between Ethereum and Ethereum Classic is examined. Where Ethereum takes the role of the funding currency and Ethereum Classic takes the role of the target currency.

4.1 Preparation Phase

The preparation phase is not really of a technical nature and therefore does not require any special implementation. As described in the attack model, all that is needed here is an announcement of the attack as far-reaching as possible. However, what can be supported technically is the credible demonstration of the availability of the necessary monetary resources in the funding currency.

To credibly demonstrate the availability of his funds, the attacker could transfer them to a smart contract, which is subsequently used during the race phase. Since the smart contract has its own address, anybody can check its balance. Furthermore, with the transfer, the attacker gives up control over his funds. Only the logic of the smart contract can now dispose of these funds. However, it may also have the functionality of repayment to the attacker. In any case, giving control to the smart contract shows that the attacker is serious about the attack.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The attacker can also collect funds in advance. If a single attacker does not have enough funds for the attack, a group of attackers can be formed. The necessary funds can be collected on the funding currency on a smart contract and used in the race phase to refund the vouchers.

4.2 Race Phase

This phase deals with the accumulation of target currency needed in the final step of the Race to the Door attack. For this purpose, the attacker offers *holders* of the target currency to exchange them for his own holdings in the funding currency. The implementation presented below is based on a voucher mechanism consisting of one smart contract per cryptocurrency.

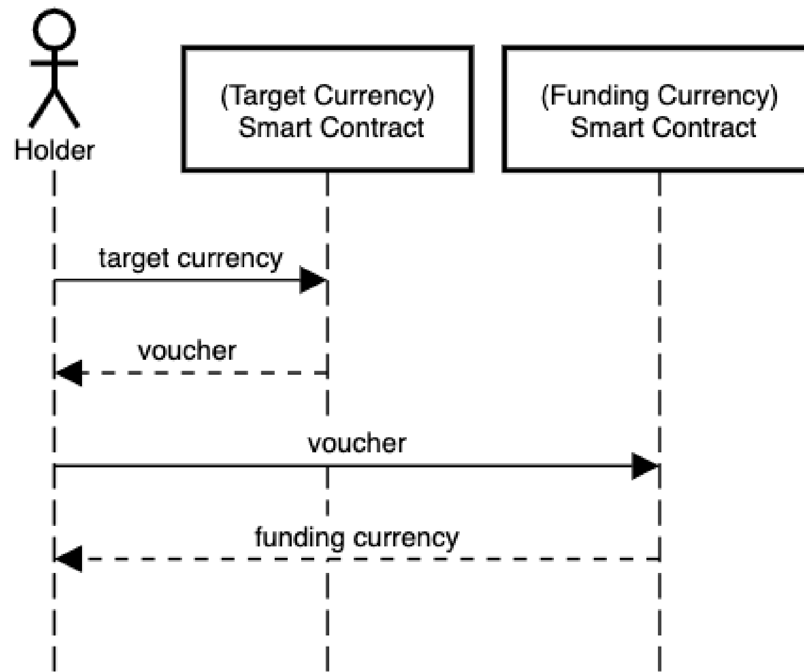

Figure 4.1: Simplified Voucher Mechanism of the Race Phase Implementation

The simplified process of the voucher mechanism is illustrated in Figure 4.1. It shows a *holder* of the target currency who wants to use the voucher mechanism to switch to the funding currency. In a first step, the *holder* sends target currency to a smart contract on the target blockchain. In exchange, a voucher is issued to the *holder*. In a second step, the *holder* can redeem this voucher by calling another smart contract on the funding blockchain and receives a corresponding amount.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

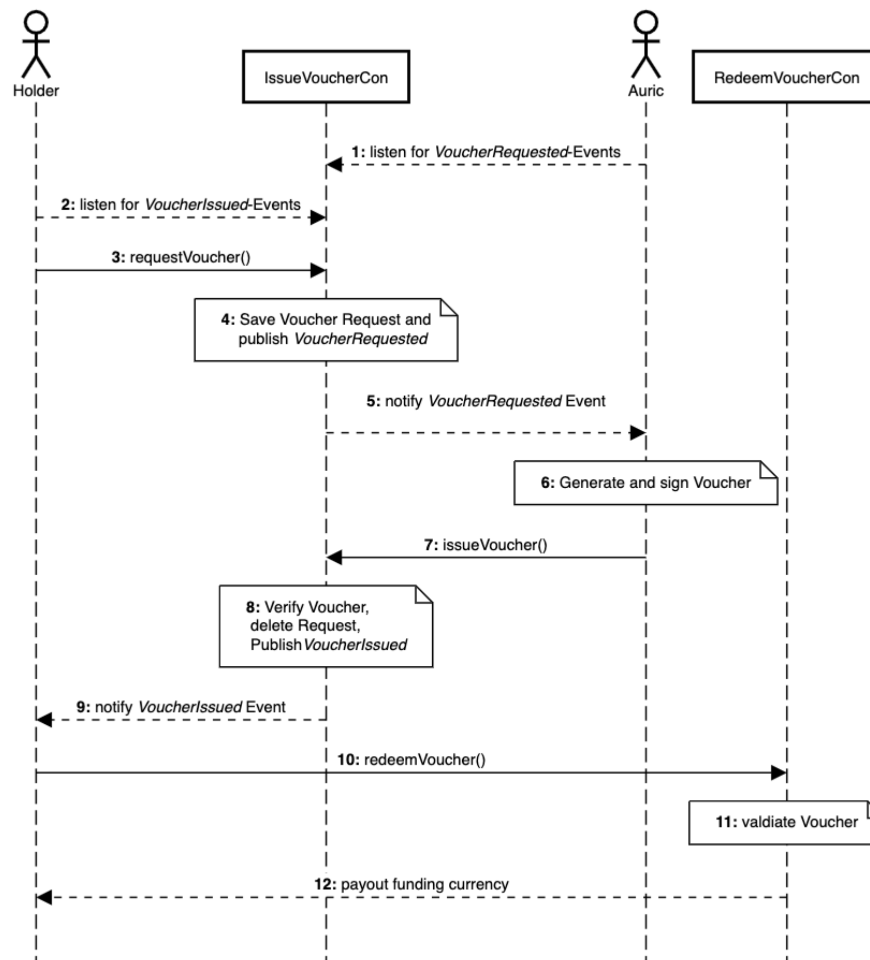

Figure 4.2: Requesting and Issuing of a Voucher in the Target Currency and Redemption in the Funding Currency.

4.2.1 Voucher Request and Redemption Process

The smart contract from the overview used to request and issue vouchers is described in more detail below. The sequence diagram in figure 4.2 shows the interaction between the following participants. The *holder* who wants to exchange his target currency into a funding currency using a voucher. The attacker *Auric*, who is responsible for issuing the vouchers. The smart contract *IssueVoucherCon* enables the interaction between the two parties. Another smart contract *RedeemVoucherCon* on the funding side enables the redemption of vouchers. The process of requesting and issuing a voucher works as described below in steps 1 through 9. After the *holder* receives the voucher and signature from *Auric*, it can be redeemed on the funding chain. This redemption process is described in steps 10 to 12. It should be noted that redemption requires only one interaction between the *holder* and *RedeemVoucherCon*. *Auric*'s involvement is not

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

required in this case. This assures the *holder* that the redemption of the voucher cannot be easily sabotaged by *Auric*, as the redeemable funds are under the sole control of the smart contract *RedeemVoucherCon*.

1. *Auric* subscribes to *IssueVoucherCon* for *VoucherRequested* events. These are published when a new voucher is requested.
2. The *holder* in turn subscribes to *IssueVoucherCon* for VoucherIssued events, which are published when a valid voucher is issued.
3. To request a voucher, the *holder* calls the method *requestVoucher()* on the *IssueVoucherCon*. In doing so, the *holder* must pay the smart contract an amount of the target currency. In addition, the *holders* address on the funding currency to which the amount should finally be transferred is specified in the request.
4. *IssueVoucherCon* generates a unique ID as a reference for the request. In addition, the sender address of the request and the current block number for the request is noted. All the data is stored in the list of pending requests of the smart contract, and the event *VoucherRequested* is published.
5. *Auric* is notified that a *VoucherRequested* event has occurred.
6. A voucher is then generated consisting of the unique request id, the funding address of the *holder*, and the amount to be paid out. To verify the validity, *Auric* signs the voucher with its private key.
7. *Auric* calls the *issueVoucher()* method of the smart contract passing the voucher and the signature.
8. When *issueVoucher()* is called, the smart contract checks the validity of the voucher and signature with the public key of *Auric*. Once the validity is confirmed, the request is deleted from the list of pending requests, and the VoucherIssued event is published.
9. Finally, the *holder* is notified that the voucher has been issued. The voucher and its signature are noted by the *holder* for later use.
10. For redemption, the *holder* calls the *redeemVocher()* method of *RedeemVoucherCon*. The voucher data (ID, funding address, funding amount) and the signature are passed to this invocation.
11. *RedeemVoucherCon* checks with the id of the voucher whether it has not already been paid out. The smart contract subsequently validates the voucher data against the signature that *Auric* generated when the voucher was created.
12. If the voucher is valid, the voucher id is added to the list of redeemed vouchers, and the amount of the funding currency is transferred to the funding address specified in the voucher.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.2.2 Edge Cases

In the above description, the interaction between the two parties went smoothly. However, this need not always be the case. In the following, some edge cases are described and mechanisms to ensure a safe interaction for both parties.

If no Voucher is issued, the Money must be refunded

It could be the case that *Auric* does not issue a voucher upon the request. In such a case, the *holder* would like to receive a refund for the request. For this purpose the *IssueVoucherCon* is extended with an additional method *payBackById()*. When called with the request id, it checks if the caller was the original requester and if a configurable number of blocks has passed since the creation of the request. This data was previously recorded in the list of pending requests when *requestVoucher()* was called and can therefore be accessed. If the check is successful, the amount is refunded. This gives *Auric* a configurable delay in which he can call *issueVoucher()* with a signed voucher.

Auric should either respond quickly or not at all

When notified of a VoucherRequested event, *Auric* should first check the locking period remaining (i.e., the number of blocks) before the *holder* can get a refund. If it seems unlikely that the transaction needed to respond to the request will be completed in time before a payout is possible, *Auric* should no longer attempt to respond to the request. This is because the transaction required for issuing the voucher must contain the voucher and signature data. This information becomes available to everyone (especially the *holder*) before the transaction is even processed. Suppose *Auric* publishes the transaction shortly before or even after the locking period. In that case, the *holder* could start a front-running attack [18, 19] by sending a transaction to the smart contract for payment. This creates a race between the two transactions, and the one that is processed first wins. Thus *Auric* could lose the collected target currency for the request. Therefore, it is crucial for *Auric* not to answer requests where it seems unlikely that they will be processed before the end of the locking period.

Auric should wait for confirmations before Issuing a Voucher

Due to forks, transactions that landed in the blockchain in the short term can disappear again. The more blocks (confirmations) that come after the block in which the transaction was processed, the less likely it will be affected by a fork. If *Auric* were to issue a voucher immediately after receiving the VoucherRequested event, it could happen through forks that this transaction is reversed. Nevertheless, the issued voucher would still be redeemable on the funding chain, and *Auric* would lose the amount of the voucher.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Miner may decide to block all Transactions by *Auric*

Miners could decide to block all transactions. However, *Auric* needs to access *IssueVoucherCon* to issue vouchers and claim the payment of the requests. The smart contract can circumvent this by not verifying the transaction's sender to be *Auric*. Instead, it has to be checked if the signature passed as a parameter comes from *Auric*. Therefore it is irrelevant whether the transaction comes from *Auric* or not. In case *Auric*'s address is blocked by the miners, he could switch accounts.

A Voucher must be invalidated when redeemed

The voucher should only be redeemable once. This must be ensured by the smart contract. ReedemVoucherCon enters the IDs of used vouchers in a list for this purpose. Before a payout takes place, it is checked whether the voucher has already been redeemed. After the payout, the voucher is entered into this list. Thus, a voucher can only be redeemed once.

Vouchers cannot be stolen

The *holder* already specifies to which funding address the voucher is to be paid out when it is requested from *IssueVoucherCon*. This address will be signed by *Auric* together with the other voucher data. If the payout address were changed later on, the signature would not match anymore, and the voucher would lose its validity. Transferring or stealing a voucher is therefore not possible. What is still possible is that a third party redeems the voucher for the *holder* on the target currency, which would also pay to the address specified by the *holder*. If such a behavior is not desirable, the RequestVoucherCon can be easily extended by checking if the caller has the same address as the payout address.

The holder should check RedeemVoucherCon Funds before Request

The *holder* must check that there is sufficient funding currency available before requesting a voucher. Due to high demand, the balance of the *RedeemVoucherCon* may be decreased. To ensure that the voucher can still be redeemed after receipt, the *holder* should ensure that the smart contract still has sufficient funds in the funding currency before requesting a voucher. Furthermore, the *holder* can observe how many vouchers have been officially issued by *Auric* and how many of them have already been redeemed.

Auric can issue Vouchers to himself

Auric is able to issue himself any number of vouchers. So he could therefore transfer the remaining funds of *RedeemVoucherCon* to himself. *Holder*s who have not redeemed their vouchers at this time would suffer damages equal to the outstanding amount. To protect himself from this, the *holder* has to watch the *RedeemVoucherCon* if only vouchers are redeemed which have been requested at *IssueVoucherCon* before. Furthermore, the *holder*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

could avoid the transfer of larger amounts and split them into smaller amounts which he transfers one after the other to minimize a potential loss.

Likewise, *Auric* could now wait until a few open requests have been received. Instead of answering them, *Auric* waits and issues vouchers to itself in the background. Next, he uses the self-issued vouchers to empty the *RedeemVoucherCon*. Finally, he can issue the vouchers on *IssueVoucherCon* to secure the payments of the requests on the target currency. However, the issued vouchers are useless because the *RedeemVoucherCon* has already been emptied. For the *holder* it is therefore always important to monitor the *RedeemVoucherCon* to see if vouchers have been redeemed that were not previously issued via *IssueVoucherCon*.

4.2.3 Employing a multiparty Singing Process

As already mentioned in the preparation phase, instead of the individual *Auric*, there could also be a group of attackers. They could also jointly issue the vouchers. In this case, the attackers could employ a threshold signature procedure. The voucher would only become valid after a threshold of signatures of the attackers. A scheme that enables this distributed key generation method and the verification in smart contracts can be found in [21]

4.2.4 Voucher Mechanism vs Traditional Exchanges

There is no third party that can interfere with the attack. Unlike traditional exchange platforms, no third party stands between the attacker and the holder. For example, if a traditional exchange platform were to notice that money was being collected for an attack, it would be able to stop trading or exclude individual users temporarily. Since there is no third party in the voucher mechanism, it cannot be stopped from the outside.

The collected funds are tied to a specific purpose. When using traditional exchanges, users can choose to spend the amount of the received amount as they choose. By using the voucher mechanism, only the holder can access the funds freely. The total funds on the target currency are tied to the purpose of the attack. The rules of disposing of the funds are defined in the smart contract and cannot be changed. Any holder who uses the exchange function can be sure that the deposited funds will be used for the attack.

4.3 Attack Phase

The third and final phase is initiated as soon as a deadline is reached. Instead of reaching a certain amount as a funding goal, this has the advantage that the attack takes place in any case. The severity of the following attack phase is then primarily dependent on the funds collected up to that point.

In this attack phase, the collected funds are used to attack the target currency. A possible implementation of such an attack would be a denial of service attack. In the next section,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

a mechanism will be explored to block transactions with transactions by exploiting the transaction ordering by gasPrice.

4.3.1 Blocking Transactions With Transactions

One way to create congestion is to use transactions to block other transactions. This can be shown under the assumption that the transactions in the blocks are generally sorted by gasPrice, which will be addressed again in a later chapter. If the assumption is correct, it also holds that miners following this sorting are trying to maximize their profits. This circumstance could also be exploited for attacks. In the following, such a method is illustrated using an example.

In the example, a mempool of a miner is considered, and block size (gasLimit) of 10 is assumed. A txId is introduced to identify the transactions. The index indicates the order according to gasPrice and thus the position in the block. Each transaction has gasUsed and a gasPrice as well as a resulting txCost which is the product of the two. It is also assumed that the transactions are independent, and therefore the gasUsed is not dependent on the order of execution of the transactions.

The miner in the example acts profit-oriented and will therefore sort the transactions by gasPrice. When assembling the block, the miner tries to put as many transactions as possible into one block. The gasLimit of 10 is an upper limit of the cummulated gasUsed of the transactions that fit into a block. Starting with the transaction with the highest gasPrice the miner will fill the block bit by bit until the blockLimit is reached.

Index	txId	gasUsed	gasPrice	txCost
1	A	2	50	100
2	B	2	40	80
3	C	4	30	120
4	D	2	20	40
5	E	2	10	20

Table 4.1: Initial State of a Mempool forming a Candidate Block

Table 4.3 shows the initial state of the mempool, which contains the transactions to be processed by a client. There are five transactions sorted by gasPrice in descending order. To fill the block, transactions A to D are selected by the miner. The sum of their gasUsed is 10, which is precisely the blockLimit of the example. Transaction E does not make it into the block as it is already full. Here it is also noticeable that transaction C has a higher txCost than the higher-ranked transactions A or B. This is because C has more gasUsed despite the lower gasPrice and the resulting txCost is therefore higher.

To effectively block transactions, another transaction can be published with a gasPrice that is greater than the lowest gasPrice of the block. This is illustrated with table 4.2. Here, a new transaction X has been published with a gasPrice of 35 and a gasUsed of

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Index	txId	gasUsed	gasPrice	txCost
1	A	2	50	100
2	B	2	40	80
3	**X**	**6**	**35**	**210**
4	C	4	30	120
5	D	2	20	40
6	E	2	10	20

Table 4.2: Updated State of a Mempool including a Transaction X created to block Transaction C and D

6. Due to the higher gasPrice, this transaction now reaches position 3 in the mempool. Since the transaction has a high gasUsed, it takes up more space and thus pushes both transactions C and D out of the block.

The transaction could be given an even higher gasPrice. This transaction would be ranked higher, but still, only the lowest transactions would be excluded from the block. Therefore, from an attacker's point of view, an even higher gasPrice only leads to increased txCost for the same effect. In order to remain cost-efficient, a gasPrice of a blocking transaction should therefore be chosen only slightly above that of the transaction to be blocked.

This leads to the following procedure. First, the transaction to be blocked is chosen. Then a transaction with a higher gasPrice is constructed. This transaction must also have a gasUsed which exactly pushes the transaction to be blocked out of the block. It is important that the transaction does not become too large so that it can no longer fit in the block. To minimize this risk, the constructed transaction can be split into several transactions with the same gasPrice but lower gasUsed.

4.3.2 FreeMoneyGrab

Auric's goal in the attack phase is to block the majority of ordinary transactions. For this purpose, the FreeMoneyGrap is introduced, which sets incentives to trigger transactions that push other transactions out of the block. The basic idea of FreeMoneyGrab is that when a smart contract method is called, it reimburses the caller for the transaction costs and pays a bonus. This triggers transactions from users who are interested in the reward. These triggered transactions take up space in the block which would be used by ordinary transactions. Some of these ordinary transactions are pushed out by the triggered transactions and thus become affected ordinary transactions.

Only a portion of ordinary transactions of a block should be blocked. Beforehand, the attacker would like to specify the percentage of a block that he wants to block. Triggered transactions which are within this limit should be rewarded. The attacker can reward these triggered transactions by reimbursing them for the transaction costs and paying them a bonus. Triggered transactions that go beyond the desired fraction should be

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

penalized. These triggered unrewarded transactions can be punished by neither refunding the transaction costs nor paying a bonus, leaving the issuer with the transaction costs.

Furthermore, only the cheapest transactions of a block are paid out. This causes that only the cheapest triggered transactions (in terms of fee/gasPrice) within the limit to be paid out. These then become rewarded triggered transactions. Calling transactions above the limit are not paid out and are unrewarded triggered transactions. Paying out only the cheapest transactions helps to make the attack cheaper and thus increases its duration.

Furthermore, the definition of an upper limit for the payout makes sense. Miners could otherwise create a block containing only transactions issued by them to FreeMoneyGrab with high gasPrice. The smart contract cannot detect this case and would pay out a very high reward. Due to the high gasPrice it would be possible for the miner to earn a large part of the funds of the smart contract. An upper limit could protect against this case to some extent.

Issues of detecting the Position of a Transaction within a Block

During the invocation of a smart contract, it is difficult to determine whether the invoking transaction should be rewarded or penalized. The EVM of Ethereum does not offer the possibility to view the position of the current transaction in the block. Also, there is no global variable how many transactions have been processed in the block so far. Furthermore, the cumulative gasUsed in the block so far is not accessible, although this seems technically feasible since the EVM has to keep a record of it. Thus it is not possible to determine the position of the calling transaction in the block. The position would give an insight into whether the transaction belongs to the part to be rewarded or not.

Even if the calling transaction is in the desired part of the block, it may not be one of the cheapest in the block. The miners process the blocks sorted by gasPrice, but they still have to consider the nonce. These circumstances are further discussed in the following chapter. This consideration of the nonce can lead to transactions in a later position in the block but having a higher gasPrice. This means that there is no point in paying out a certain number of transactions according to the reverse position in the block if it is intended to pay out the cheapest ones.

Calculating the Number of rewarded Transactions per Block

The percentage of the block to be blocked (percentToBlock) can be converted into a number of rewarded transactions as follows. The creator of the FreeMoneyGrab knows the typical gas usage of a call to the smart contract (gasPerCall). Furthermore, the gasLimit per block is known, i.e., the maximum amount of gas that can be processed per block. If the attacker now wants to block about 50 percent of the transactions of a block, the gasPerCall of the transactions to FreeMoneyGrab must also take a total of about 50 percent of the gasLimit of the block. So the attacker can multiply the gasLimit by the

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

percentToBlock divided by gasPerCall and get the number of transactions that should be rewarded per block.

Approach

The goal of FreeMoneyGrab is to pay out the cheapest N transactions of a block. As described above, the smart contracts can neither recognize the position of a transaction nor can they trust a correct sorting or whether further calls in a block follow. Therefore it is also not possible for FreeMoneyGrab to decide immediately if a reward is to be paid.

FreeMoneyGrab solves this problem by delayed processing of the calls. So the calls within a block are first only recorded. For this, the smart contract maintains a list with the N cheapest transactions of a block which it wants to reward in the future. As soon as a call takes place in a new block, the smart contract can assume that the previous block has been completed and begin with the payment of the collected transactions. This procedure is described in more detail below.

Execution Phases of the Smart Contract

In the following, a procedure is described that overcomes these problems and can be implemented as a smart contract. The smart contract offers only one method, which can be called repeatedly. With each call, the following 4 phases will be executed:

- **Record.** Each call of the method is recorded. The address of the caller is noted, which can be used for the payment of the reward. Since the transaction hash cannot be accessed, a consecutive id is generated for referencing. Furthermore, the gasPrice of the transaction and the current blockNumber are stored with the entry.
- **Reorder.** The calling transaction is grouped with other transactions of the same block into a collection and sorted by gasPrices. Since only one transaction is inserted into an already sorted collection at a time, a heap[22] is a suitable data structure for efficiency reasons.
- **Retain.** In advance, the attacker determines how many transactions will be rewarded. If the number of items would exceed this limit after inserting another item, the transaction with the highest gasPrice will be removed. This helps to keep a list of the current lowest transactions per block.

  When using a heap for keeping track of block transactions, a max heap regarding gasPrice should be used with a size equal to the number of transactions that are going to be rewarded per block. As long as the heap is not full, the newly recorded transaction is simply added to the heap. When adding a new transaction with a full heap, the gasPrice of the new transactions must first be compared to the most expensive transaction recorded so far. If the new transaction has a lower gasPrice than the highest one, it is replaced with the top transactions, and the order of the heap is restored. If the gasPrice of the new transaction is higher, it is ignored,

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

and the heap remains unchanged. This way, the heap retains only the relevant transactions of a block.

- **Reward.** With the processing of a call within a later block, it can be assumed that all collections built to the previous blocks contain only the calling transactions with the lowest gasPrice. Therefore, if possible, a transaction from a previous block that has not yet been paid out is selected in each call. For this transaction, the transaction costs, as well as a bonus, are paid out to the caller, and the transaction is marked.

To get a better picture of this procedure, the following example is considered. In this example, it is assumed that the four cheapest transactions are paid out. Table 4.3 shows the state of the smart contract after transactions A to D. The four transactions have been grouped into block 1 and sorted by gasPrice. Since block 1 is the current block of the contract, no transactions have been paid out yet.

BlockNr	txId	gasPrice	paid out
1	A	50	false
1	B	40	false
1	C	30	false
1	D	20	false

Table 4.3: Initial State of FreeMoneyGrab with Transactions A to D

Assuming that another transaction E with gasPrice of 10 is added. In the recording phase, the transaction is assigned an id, and the gasPrice and blockNr are determined. As the transaction happens still within block 1 it is regrouped with the collections containing transactions A to D and then reordered by gasPrice. In the retain phase, the size of the collection is checked. As the maximum collection size of 4 is exceeded, transaction A is removed since it has the largest gasPrice. Subsequently, the rewarding phase is carried out. Since the smart contract has not recorded any previous blocks and one cannot be sure if further transactions will follow in block 1, no transaction will be paid out at this point. The final state of this run is shown in table 4.4.

BlockNr	txId	gasPrice	paid out
1	B	40	false
1	C	30	false
1	D	20	false
1	E	10	false

Table 4.4: State of FreeMoneyGrab after Transaction E

Next, assume that another transaction F takes place in a new block 2 with a gasPrice of 60. The record phase will still identify the transaction and note the gasPrice as well as

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

the blockNr. During the retain phase, a new collection has to be initialized containing the new transaction. Since the collection size of 1 is below the limit of 4, there is nothing to do in the retain phase. However, the rewarding phase now detects that the current block number has changed from 1 to 2. Since it is no longer possible to add transactions to block 1, it can be assumed that its collection now contains only the four cheapest transactions that have called the smart contract. Therefore, the cheapest unpaid transaction E will be paid out to its sender and marked as paid. The state after processing the transaction F can be found on table 4.5.

BlockNr	txId	gasPrice	paid out
1	B	40	false
1	C	30	false
1	D	20	false
1	E	10	true
2	F	60	false

Table 4.5: State of FreeMoneyGrab after Transaction F

Finally, consider one more transaction G in block 2 with a gasPrice of 50. The record phase records the data as usual. Again, the collection size of block 2 is still under the limit, so the transaction is simply added. In the rewarding phase, another transaction from block 1 is selected again and paid out. The next cheapest unpaid transaction is D, so the sender is paid, and the transaction is marked as paid. The final state of our example after processing transaction G can be found on table 4.6.

BlockNr	txId	gasPrice	paid out
1	B	40	false
1	C	30	false
1	D	20	true
1	E	10	true
2	F	60	false
2	G	50	false

Table 4.6: State of FreeMoneyGrab after Transaction G

How the Incentives influence Participants

The smart contract cannot guarantee that the transactions it records are the last in the block. Only transactions that call the smart contract can be recorded, other transactions that are in the same block can therefore not be observed. So the smart contract cannot determine if the transactions are the last and therefore the cheapest in the block. However, the callers of the smart contract are incentivized to bring their transaction into the block with the lowest possible gasPrice in order to get paid. If a caller would call the smart

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

contract with a relatively high gasPrice, other callers could decide to submit lower-priced transactions which could prevent the first caller from being paid. Therefore, the callers are encouraged to monitor the current state of the mempool in order to position their transactions optimally. This is also desirable from the attacker's point of view, as it makes the attack cheaper since expensive transactions do not have to be paid out.

Another option to reinforce this kind of behavior could be to charge a fee to the smart contract, which has to be deposited when calling. This fee could be refunded when the transaction is paid out. However, if the transaction is not among the cheapest, the caller loses his transaction costs and the deposited amount, thus increasing his losses even more. The fees collected in this way could then be used to prolong the attack further.

Reward Size and Attacks against the Smart Contract

The size of this bonus is crucial to prevent attacks on the smart contract itself. As already mentioned, the cheapest transactions of a block will be refunded, and an additional bonus will be paid. A possible attack against the smart contract would be if the attacker would make many transactions with abnormally high gasPrice so that no other transactions in the block would go to the smart contract. In this case, only the cheapest ones would be paid, but they would also receive an abnormally high bonus. The calculation of the bonus size and the number of transactions to be paid out must therefore be made so that the losses due to the transactions that are not paid out are much higher than the profits gained from the transactions that are paid out.

4.3.3 Fomo10x

An already known contract that successfully performs transaction triggering is Fomo3D. This is a turn-based game and Ponzi scheme which sets three different incentives:

- In the game, keys can be bought. When buying a key, there is a certain chance to get paid a bonus.
- Every key gets dividends for each additional key that is purchased at a later time.
- An internal timer counts down to the end of the round. When a key is bought, the timer is increased. If the timer runs out, the last buyer wins a majority of the pot.

This combination leads to many transactions early on, which quickly drop off and only occur sporadically towards the end. This is because many players want to buy keys at the beginning to benefit from the dividends when additional keys are purchased later in the game. Just before the timer would expire, the incentive is high to buy a new key. This could be because the cost of buying a key is relatively small compared to the offered reward.

Fomo10x eliminates undesirable incentives and increases the desirable incentives. It represents a modified version of Fomo3D that satisfies the attacker's requirement to

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

create congestion. Thereby, the attacker uses the collected funds as a pot for the game. Like Fomo3D, a timer is decremented by a constant number (txPerBlock) per block. With each call to Fomo10x, however, the timer is incremented. If it reaches 0, the winner is determined who is the first caller of the previous block. Furthermore, the timer is increased only so little that at least txPerBlock calls per block are necessary not to let the timer run out.

The strategy from the player's point of view is to get the transaction as high as possible into the next block when the timer is about to run out. However, the other players following the same strategy would also try to push their transaction ahead to be counted as the winner. This, in turn, can lead to the block receiving enough transactions for the timer to not run out and for the round to continue.

Game and Game State

The smart contract is responsible for maintaining a game state and paying out the winnings at the end of a round. Fomo10x provides a method that goes through phases and updates this game state.

The game state of Fomo10x consists of the following three variables:

- **credits.** The variable credits acts as a timer, which is decreased by a fixed amount txPerBlock with each past block. Each call to Fomo10x increases the credits by 1. When the credits reach 0, the winner is determined, and the pot is paid out.
- **latestBlockNumber.** Fomo10x stores the last observed block number in the variable latestBlockNumber. This can be used to determine whether and how many blocks have passed since the last call.
- **currentWinner.** The first caller of a new block is stored in this variable. When the timer expires, this caller is declared the winner of the round and is paid the pot.

Furthermore, the following two constants are defined:

- **startCredits.** The startCredits are the initial credit of the round. These have the purpose of cushioning the start of the round. Especially at the first start of the game it can happen that some players do not know that the round has started and therefore less transactions are issued. This would lead to an abrupt end right at the beginning, which is prevented with the startCredits.
- **txPerBlock.** The amount of credits deducted per past block is txPerBlock. In other words, this constant defines the average amount of transactions in a block that is needed for the game not to end. Thus the strength of the desired congestion can be controlled by the attacker. The higher the constant, the more transactions need to be in a block for the game to continue and the higher the resulting congestion.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Fomo10x.play()

Fomo10x provides a method play() that updates the game state depending on the state of the contract and the block in which the transaction is executed.

When the method is called for the first time, the game is initialized. For this, the credits are set to the startCredits, the current blockNumber is set as latestBlockNumber, and the caller is set as currentWinner.

When called later, Fomo10x first checks whether the current block number is still the latestBlockNumber. If this is not the case, the credits must be updated. This is done by subtracting the amount of txPerBlock for each past block. If the credits are gone, the currentWinner is paid, and the round is finished. If the credits are not expired, the current caller is set as currentWinner because this is the first caller in the new block. In case the latestBlockNumber is equal to the current block number, the credits are increased by 1.

txId	blockNr	sender	credits	latestBlockNumber	currentWinner
A	1	player1	9	1	player1
B	3	player2	2	3	player2
C	3	player1	3	3	player2
D	3	player3	4	3	player2
E	4	player1	-	-	-

Table 4.7: Fomo10x Game State Changes while processing Transactions

For a better understanding, the following example is considered. The txPerBlock is set to 4 and the startCredits to 8. Table 4.7 shows the transactions and state changes of the game. On the left side, the txId, blockNr, and sender of the transactions calling Fomo10x are displayed. The right side of each line shows the game state of Fomo10x after each transaction is processed.

With transaction A the game is called for the first time and set to the start state. For this, the credits are set to the startCredits and incremented by 1 as with every call. The blockNumber of transaction A is 1, which is entered in latestBlockNumber. Furthermore the caller player1 is set as currentWinner.

The next call and Fomo10x happens with transaction B in block 3 and has been submitted by player2. Since the latestBlockNumber of 1 is 2 blocks behind the current blockNumber 3, the txPerBlock of 4 is subtracted from the credits twice, resulting in preliminary credits of 1. As the credits are above 0, the game continues, therefore the credits are increased by 1 and the current winner is set to player2. The latestBlockNumber is set to the current blockNumber 3.

Transactions C and D are still in the same block 3, therefore neither the latestBlockNumber nor the currentWinner changes. Only the credits are increased by 1 per call, resulting in the credits to be set to 4 at the end of block 3.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Finally, a call from transaction E reaches the smart contract in a new block 4. Since the blockNumber has increased by 1, txPerBlock is subtracted from the credits, . As the credit reaches 0, the winner is determined. This is done by simply taking the currentWinner variable, which is player2, the first caller of the previous block. Player2 will be paid the prize, and the round ends.

How the Incentives of Fomo10x influence Players

Fomo3D incentivizes transaction triggering by giving the prize to the last caller before the timer expires. When the timer approaches 0, players are willing to buy another key to become the last caller. The cost of a key is relatively low compared to the expected profit. However, the purchase increases the timer, and the game continues. If the timer is about to run out again, the phenomenon repeats itself.

Fomo10x reinforces this incentive by counting down the timer faster. The timer counts down so fast that several calls per block are necessary not to run out. Therefore, each new block could be the last one of the round. This incentivizes players to make a transaction with a high gasPrice which ranks them as the first caller and declares them the current winner.

Assuming the credits of Fomo10x are slightly above txPerBlock, the timer may expire with the next blocks, and the currentWinner of the next block would become the winner. This may motivate a player to issue a new transaction calling Fomo10x and make that player the currentWinner when the timer expires. Other players may observe such a transaction, for example, in the mempool that compiles the next block, and they have two options to counter.

- **Go higher.** Option one is to issue a transaction with a higher gasPrice which would push their transaction ahead of the other players' transactions. Therefore, the sender would become the currentWinner of the next block. This option makes sense if the cost of outbidding the transactions is low and there are not enough transactions at Fomo10x in the block so that the round is extended by another block.
- **Go lower.** Option two would be to issue cheaper transactions, generating enough calls to Fomo10x so that the timer is extended to run for another block and try to become the currentWinner in the following block. This makes sense if outbidding the currentWinner is very expensive, or there are already some transactions that would extend the timer for one more block when including the additional transactions.

When players choose the go higher option, they should be careful not to extend the timer with the additional call. However, suppose some high transactions have already been issued. In that case, it makes sense to issue cheaper transactions that only aim to extend the timer since the currentWinner is not updated for their senders. All together this results in an amount of transactions that is on average higher than txPerBlock as long as the game is running, which is the primary goal of the attacker.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

4.3.4 PayForEmptyBlocks

To achieve Auric's goal of generating congestion, the third smart contract uses a different approach than the previously presented attacks. Instead of incentivizing users to issue more transactions, this smart contract targets miners to process fewer transactions instead. This is achieved by offering miners a reward that they can collect from PayForEmptyBlocks if they can prove that they have added an empty block to the blockchain. When called, the smart contract can then check if the block contains no transactions and if the caller is the miner of the block. If the data is valid, a reward is paid out.

The reward size depends on the lost transaction fees. By mining an empty block, the miner cannot collect transaction fees that he would receive by processing transactions. Therefore, it only makes sense for the miner to work on an empty block if the smart contract promises to pay a reward that is higher than the amount of the lost transaction fees and pays an additional amount on top of that.

Validating if a given Block is empty

Empty blocks are blocks that do not contain any transactions. Ethereum's block headers include a transaction root hash of the Patricia tree [23][24] referencing the processed transactions of the block. If a block does not contain any transactions, the Patricia tree is empty, and its hash has a specific value. To validate if a block does not contain any transactions, the transaction root hash from its block header can be compared with this empty hash value.

EVM Restrictions

To validate the requests to PayForEmptyBlocks, the data of the corresponding block header data must be checked. So the transaction root hash must be read from a block header to compare it with the empty hash. In order to pay out the reward correctly to the miner, the coinbase address is also required. This is the address to which the block reward for creating the block was paid out and is also located in the block header. Unfortunately, the Ethereum Virtual Machine (EVM) allows only limited access to underlying blockchain data [25]. The EVM only allows access to the block hash (i.e. the hash of the block header) of the last 256 past blocks. Accessing the necessary data for verification and payout like the transaction root hash or the coinbase address of a block is not possible. Therefore, it is not possible for the smart contract to verify whether this block is empty by providing a block number. For the same reason, it is also impossible to make a secure payout to the correct miner.

Overcoming EVM Restriction to Validate Empty Blocks

Although it is not possible to read header data from the blockchain within a call, it is possible to verify whether a certain block hash occurs in the blockchain. This hash is calculated from the header of the block. To calculate the hash of a block, the block header

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

is needed, which contains 15 variables, including the coinbase address, the transaction root hash, and the block number. These value are now arranged in a certain order and Recursive Length Prefix (RLP) encoded [26]. The encoded header is then hashed with keccak-256 [27] which gives the hash of the block.

The solution for the EVM restrictions is that the caller provides the block data. This data can then be hashed and verified if the hash is present in the blockchain. The block header also contains all relevant variables necessary for checking for emptiness and the payment of the reward.

Submitting an empty Block on PayForEmptyBlocks

The smart contract provides a method for miners to submit mined empty blocks to collect a reward. This method takes the RLP encoded block header data as a parameter. When called, the following three steps are executed:

- **Extract.** First, the required variables must be read from the encoded block header data. The PayForEmptyBlocks subsequently requires the block number, the coinbase, and the transaction root hash. For this purpose, the header data is decoded, and the respective variables are read via their index. The strict order of the data in the header ensures that the data is read correctly.
- **Verify.** To verify the block data, the received block header data is first hashed with keccak-256. With the extracted block number and the block hash function of the EVM the actual block hash is fetched from the blockchain. These two hashes are checked for a match to determine if the received header has been processed on the blockchain. Furthermore, it is checked if the transaction root hash is identical to the empty hash to determine that no transaction was processed in the block. If one of the checks fails, the transaction is aborted.
- **Payout.** Once the block header has been verified, the miner is rewarded. To do this, the smart contract first checks an internal list to see if the block has already been paid out. If the block is not yet paid out, a reward is send to the extracted coinbase address, and the block is entered in the list of paid out blocks.

Example: Encoding a Block Header

To send the block header data to the PayForEmptyBlocks the variables must be ordered in a certain sequence and encoded with RLP. In this section, this is demonstrated with block number 12,398,939 from the Etherum network.

Obtaining the raw block data. First, the block header data must be retrieved from the network. For this, the mining client geth offers the method eth_getBlockByNumber via a JSON-RPC API [28]. If this method is called with the block number 12,398,939 of the example, the header data is displayed as shown in listing 4.1. This data includes all 15 required header variables as well as some extra information. It should also be

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

noted that this block is an empty block, i.e., it does not process any transactions. This is indicated by the gasUsed of 0 and the transactionRootHash, which is equal to the empty hash.


```
{
difficulty: '7798586123748724',
extraData: '0x73706964657231330e321fd0',
gasLimit: 14985035,
gasUsed: 0,
hash: '0x411f8c2770f0b68a829406a05c94a6b8d9d0c0d3d6da97d844d802a0657264ae',
logsBloom: '0x000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000',
miner: '0x04668Ec2f57cC15c381b461B9fEDaB5D451c8F7F',
mixHash: '0x83b2572892a253fc5495155048420bd5fc72fa1f990b2bf7c6baeee1693c3a2d',
nonce: '0x432a92900058319e',
number: 12398939,
parentHash: '0x1df88c44a6352759d63ab58418036cc5edebd169628834910a5092efae711458',
receiptsRoot: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
sha3Uncles: '0x1dcc4de8dec75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d49347',
size: 530,
stateRoot: '0x5f0d38ba68c6f453861e3dc4961c1b2ec86c641565917285aed211df8fe00592',
timestamp: 1620546814,
totalDifficulty: '24417290626396067328458',
transactions: [],
transactionsRoot: '0x56e81f171bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421',
uncles: []
}
```


Listing 4.1: Raw Block Data from Block 12,398,939

Ordering and hex conversion. After the header data has been fetched, the next step is to convert some variables. The values of difficulty, number, gasLimit, gasUsed, timestamp have to be converted to hexadecimal numbers. It is important that 0 values like the gasUsed are not converted to '0x0' but '0x'. The resulting hex values together with the remaining block header data must then be arranged in a list in the same order as it is processed by the geth [29] client:

1. parentHash
2. sha3Uncles
3. miner
4. stateRoot
5. transactionsRoot
6. receiptsRoot
7. logsBloom
8. difficulty
9. number

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek Your knowledge hub WIEN

10. gasLimit
11. gasUsed
12. timestamp
13. extraData
14. mixHash
15. nonce

This list is now passed to an RLP encoder function which returns the output of listing 4.2:

```
f9020aa01df88c44a6352759d63ab58418036cc5edebd169628834910a5092efae711458a01dcc4de8dec
75d7aab85b567b6ccd41ad312451b948a7413f0a142fd40d493479404668ec2f57cc15c381b461b9fedab
5d451c8f7fa05f0d38ba68c6f453861e3dc4961c1b2ec86c641565917285aed211df8fe00592a056e81f1
71bcc55a6ff8345e692c0f86e5b48e01b996cadc001622fb5e363b421a056e81f171bcc55a6ff8345e692
c0f86e5b48e01b996cadc001622fb5e363b421b901000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000000000000000000000000000000000000000000
0000000000000000000000000000000000000000000000871bb4c5d3a8057483bd315b83e4a74b8084609
794fe8c73706964657231330e321fd0a083b2572892a253fc5495155048420bd5fc72fa1f990b2bf7c6ba
eee1693c3a2d88432a92900058319e
```

Listing 4.2: Rlp encoded Block Header of Block 12,398,939

Validation. The RLP encoded data would now be ready to be submitted to PayForEmptyBlocks. To validate this data, the keccak-256 hash can be calculated before. If the steps have been done correctly, this hash should match the block hash we originally received from geth and is shown in listing 4.1.

4.3.5 How the Incentives of PayForEmptyBlocks influence Miners

Auric's goal in the attack phase is to create congestion. In contrast to the approach of incentivizing participants to issue additional transactions that push ordinary transactions out of the block, the approach behind PayForEmptyTx targets the miners. Here, the miners are offered a reward if they do not process any transactions in their blocks. This would disrupt the processing of all ordinary transactions that would have been processed in such a block.

The amount of the expected reward is essential for the miner. Under normal circumstances, the miner receives transaction fees for processing transactions. However, this source of profit is eliminated if the miner decides to mine empty blocks. For profit-oriented miners, mining empty blocks only makes sense if the expected reward from PayForEmptyTx for submitting empty blocks is higher than processing the currently open transactions.

As long as empty blocks are mined, congestion is created. The resulting congestion leads to delays in the processing of transactions. Therefore, some transaction issuers may

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

decide to increase the gasPrice and thus the transaction fees of their transactions to incentivize faster processing. In sum, this could tilt the threshold for the miner so that it is again more profitable to process transactions.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 5

Feasibility and Cost Analysis

This chapter focuses on the feasibility of the attack in the context of Ethereum. To this end, the topic of transaction ordering will be revisited, and the current status will be examined closely. In order to estimate the costs of the attack, historical congestion phases are first examined. On this basis, the estimated costs of the attack variants are calculated and compared.

5.1 Ethereum Data & Data Model

In order to perform the following analyses, data about the blocks and transactions of the Etherum blockchain must be collected. The collection of the relevant blockchain data is supported by a data model presented in the following. This data model also serves as a basis for queries and enables the evaluation of the data.

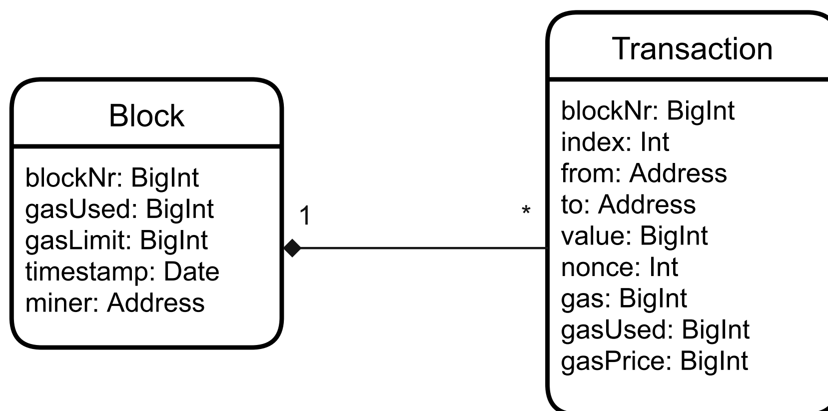

Figure 5.1: Data Model for Ethereum Blockchain Data

Figure 5.1 shows the data model for the two entities *Block* and *Transaction*. Blocks can contain a set of transactions. Each block has a unique consecutive blockNr. The gasUsed of a block is the sum of the consumed gas of the associated transactions. The gasLimit is an upper bound on the total amount of gas that is allowed to be processed in a block. The timestamp is the moment when the block was created by the miner. The miner property is the coinbase address which is the beneficiary, where the block reward is paid out.

Transactions reference the blocks via the blockNumber and have an index that specifies the order within the block. Also, they have a sender address from and a receiver address to as well as an optional amount value. The upper limit of the execution steps in of the transaction is the gas. These steps must be paid for, and the gasPrice indicates the amount of gas per step that the sender is willing to pay. The actual amount of gas consumed by the transaction is gasUsed.

For the following analysis, data was collected from the Ethereum blockchain network. Specifically, these are the blocks starting with blockNr 11564730 through 11660330. The data spans a 15-day period starting with the first and ending on January 15, 2021. A total of 95,601 blocks with 14,408,746 transactions were recorded.

5.2 Transaction Ordering on Ethereum

Transaction ordering is an essential prerequisite for the feasibility of the attack and estimation of the costs associated with a race to the door attack on Ethereum. In section 2.5 transaction ordering was already mentioned. In [17] generally implies sorting according to gasPrice. In order to make sure that the assumption of sorted transactions within the blocks is still valid, the collected data will first be examined.

Some blocks are empty. There are exactly 1837 empty blocks in the dataset[1]. This means that these blocks do not contain any transactions. The reason for this could be that there are too few transactions to process. Figure 5.2 shows the block utilization compared with the percent of empty blocks. The block utilization here is the ratio of gasUsed to gasLimit and is on average 97.02 percent for the entire period under consideration. These numbers suggest that although there are enough transactions, they are not being processed for a reason. One possible explanation for this phenomenon could be SPV mining [30]. SPV mining is a common practice in some mining pools where a new block is built before the previous one has been validated. The goal is to continue working on the longest chain and to maximize profits. When a new block header is received, an empty block template is created containing only the coinbase transaction. Only when the previous block has been fully downloaded and validated, a valid block can be created, which also contains transactions. Thereupon, blocks were searched for, which only consist of transactions originating from miners themselves. The miner transactions are identified by having

[1] In this section, the blocks and containing transactions from blockNumber 11564730 to 11660330 from the Ethereum main network are analysed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

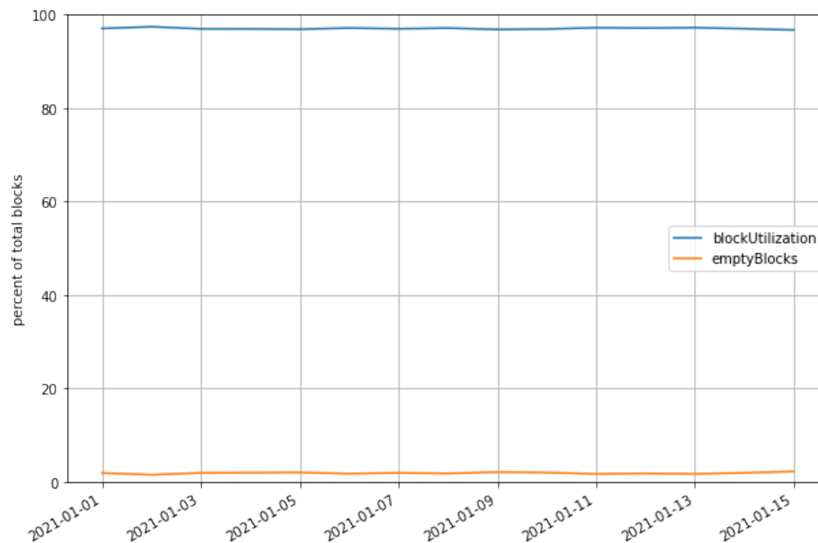

Figure 5.2: Average Block Utilization per Day

the same from address as the block miner (coinbase) address of the block in which they are processed in. In the dataset, 2177 further blocks of this type were found. These could also indicate SPV mining. Instead of an empty block to mine, own transactions are included to increase the utilization. A total of 4014 blocks or 4.20 percent can be attributed to SPV mining.

As described above, miners aim to maximize profits. They are therefore incentivized to prioritize the transactions with higher transaction fees. This transaction fee is calculated from gasPrice times gasUsed. However, the gasUsed can only be calculated when the order of the transactions has already been determined. Furthermore, the transaction occupies space in proportion to its gasUsed. Therefore it makes sense to sort the transactions by gasPrice. In the present dataset, only 30499 blocks, or 31.90 percent, are strictly sorted by gasPrice. This means that for every transaction, it holds that there is not a single other transaction in the block with a higher gasPrice. This is because the sorting algorithms have to consider the consecutive nonce of a sender. So transactions that have to be processed later can have a higher gasPrice. Taking this into account, we arrive at 75912 sorted blocks or 79.41 percent. Earlier, we saw that miners treat their own transactions differently. So miners could process own transactions preferentially despite a low gasPrice. If these miner transactions are ignored in the sorting, there another 10833 blocks can be considered sorted, and the total number of blocks amounts to 86745 or 90.74 percent.

The available data set confirms the assumptions that the miners act profit-oriented. About 91 percent of the blocks are sorted by gasPrice, taking into account the increasing nonces. Including the 2 percent of empty blocks, the total number of sorted or empty blocks is 93 percent. This leaves about 7 percent of blocks where no clear sorting can be

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

observed.

5.3 Transaction Blocking Costs

Considering the mechanism of blocking transactions with transactions as described in section 4.3.1 from the cost perspective, further observations can be made. Transaction X, which displaces transactions C and D from the block, has a txCost of 210, which is more than the sum of the two individual transactions. Assuming that X has the same gasPrice as C but still a gasUsed of 6 and remains in the same position, it would continue to displace the other transactions. In this case, its txCost would be 180, exactly 20 more than the sum of the displaced transactions.

Index	txId	gasUsed	gasPrice	txCost	premium
1	A	2	50	100	-
2	B	2	40	80	-
3	**X**	**6**	**30**	**180**	-
4	C	4	30	120	(30 - 30) * 4 = **0**
5	D	2	20	40	(30 - 20) * 2 = **20**
6	E	2	10	20	-

Table 5.1: Updated State of a Mempool including a Transaction created by the Attacker with equal gasPrice as the highest displaced Transaction

This is due to the difference in gasPrice compared with transaction D. The newly considered transaction X's gasPrice is by 10 higher. Multiplied with the gasUsed of transaction D results in a total 20 in txCost. As there is no difference in gasPrice for transaction C, there is also no additional txCost for this transaction. This premium of 20 is what that the attacker involuntarily pays to the miner for pushing out the transactions from this block. The state of the mempool for this example is shown in table 5.1.

Index	txId	gasUsed	gasPrice	txCost	premium
1	A	2	50	100	-
3	**X**	**6**	**40**	**320**	-
2	B	2	40	80	(40 - 40) * 2 = **0**
4	C	4	30	120	(40 - 30) * 4 = **40**
5	D	2	20	40	(40 - 20) * 2 = **40**
6	E	2	10	20	-

Table 5.2: Updated State of a Mempool including a Transaction created by the Attacker with equal gasPrice as the highest displaced Transaction

If the example is extended further by the attacker trying to block transaction B as well. The corresponding state of the mempool can be found on table 5.2. So transaction X

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

would have to be adjusted to a gasPrice of 40 with a gasUsed of 8. This results in a total txCost of 320. The difference between transaction X and the sum of the txCost of the pushed-out transactions amounts to 80. This is made up of the premium for C of 40 plus the premium for D, which is now also 40. The higher the gasPrice of the first transaction to be blocked, the higher the premium for each subsequent transaction. Likewise, the amount of gasUsed that is displaced is a factor. As a result, the lower limit for the transaction block cost is obtained by multiplying the highest gasPrice of the displaced transactions by the sum of the gasUsed displaced transactions.

By applying this formula to the data set, it is now possible to estimate the cost of blocking a portion of the transactions in a block. Under the assumption that the transactions of each block are strictly ordered by gasPrice, they are then equally divided among 10 buckets. To estimate what it costs to block a percentage of a block's transactions (up to a certain bucket), the sum of the displaced gasUsed is multiplied by the highest gasPrice of the displaced transactions for each block.

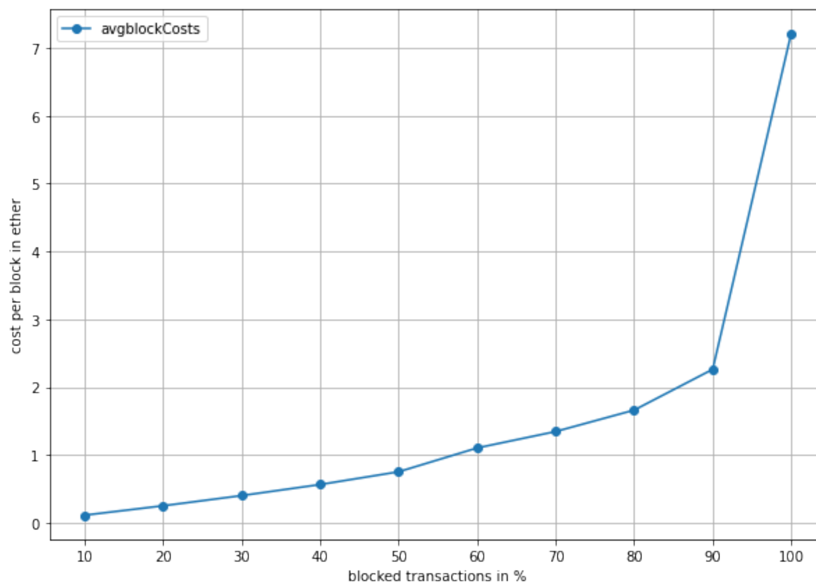

Figure 5.3: Average Costs of Blocking a Portion of Transactions

The results of this evaluation on the dataset[2]. can be found in figure 5.3. It shows the development of the average cost of blocking with an increasing percentage in Ether. Blocking 10 percent of the transactions is costing on average 0.11 Ether. Blocking half of a block requires an average of 1.10 Ether. The curve seems to increase almost linearly until blocking 90 percent of the transactions costs about 2.26 Ether. However, the average cost of blocking all transactions is 7.2 Ether, which is more than three times the cost of

[2]This section analyses blocks and containing transactions from blockNumber 11564730 to 11660330 from the Ethereum main network.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

blocking 90% of all transactions within a block. The linear trend, therefore, does not seem to hold.

At the time of writing, the reward for finding a block is 2 Ether. For processing transactions, the miners receive additional transaction fees averaging 1.27 Ether per block for the observed period. The costs related to blocking 60 percent of the transactions is 1.10 Ether, and blocking 70 percent is costing 1.34 Ether. This means that it makes more sense for the attacker to pay the miner a reward of more than 1.27 Ether to mine an empty block than to block 70 percent of the transaction and pay 1.34 Ether.

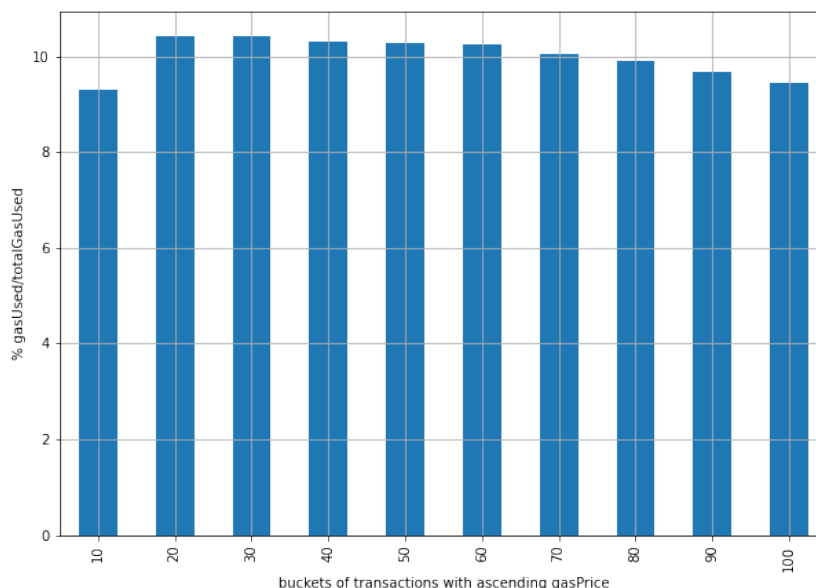

Figure 5.4: Relative gasUsed per Bucket

To find the reason for this irregularity in blocking costs on the upper end, another look at the formula is taken. One factor is the gasUsed of the transactions. It could be the case that the transactions in the block which have a high gasPrice and therefore are ranked higher also have a high gasUsed. To investigate this, the sum of all gasUsed is compared in relation to the total sum of all gasUsed per bucket. The comparison is shown on Figure 5.4. The proportions of the buckets range between 9.3 and 10.4 percent, roughly equal in relation to the gasUsed. An increased share of gasUsed in the bucket of the transactions with the highest gasPrice cannot be detected. The most expensive transactions even have the second-lowest share with only 9.45 percent. The decisive factor must therefore be a sharp increase or outlier in the gasPrice.

To confirm this assumption, the minimum, maximum, and median values are measured in addition to the average. These values are listed in table 5.3. The first thing to note here is the max value for the 100 bucket, which is almost 1800 Ether. This is a clear outlier with an average around 7 Ether. Since the calculation of the average is unfortunately very susceptible to such outliers, the median is also given on the right side, which is less

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

blockedTxPercent	meanBlockCosts	minBlockCosts	maxBlockCosts	medianBlockCosts
10	0.111949	5.354100e-05	7.657972	0.083490
20	0.250470	1.326915e-07	8.613791	0.188321
30	0.400604	8.651180e-11	8.628491	0.302427
40	0.563636	7.878352e-11	10.760531	0.425472
50	0.751645	2.424884e-11	26.778612	0.562087
60	1.102898	1.193381e-11	48.435922	0.765181
70	1.344781	1.225835e-11	63.232731	0.951148
80	1.660840	1.230476e-11	341.199061	1.161761
90	2.261684	1.246657e-11	453.057423	1.477563
100	7.212969	1.232384e-02	1797.566739	3.237533

Table 5.3: Min, Max, Mean and Median of Costs for Blocking certain Percentages of a Block in Ether

sensitive. The median values of each bucket are lower than their mean values, which could indicate a skewed distribution. For the 100 bucket, the median value is less than half of the mean value. The plot of the mean and median values is also shown in figure 5.5.

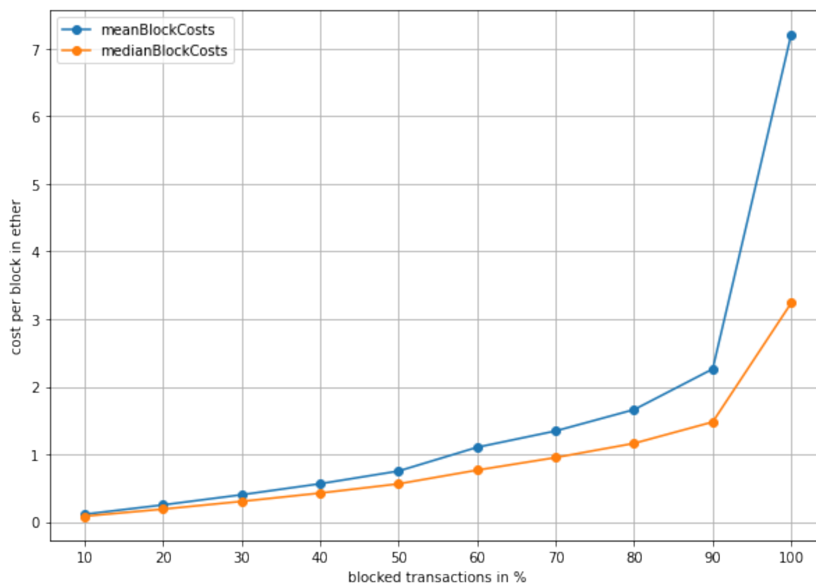

Figure 5.5: Average Costs of Blocking a Portion of Transactions including Median in Ether

Using the data above, it is possible to calculate the cost of blocking x blocks with y percent. To do this, multiply the percentage y in the 5.3 table by the desired number of blocks to be affected. For example, the cost of blocking 50 percent of transactions for

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

30 minutes can be calculated as follows. Assuming a block time (the time between two blocks) of about 15 seconds, or about 4 blocks per minute, one would have to block 30 times 4 equals 120 blocks. The cost of blocking 50 percent of transactions in a block is 0.75 Ether. Therefore, the cost to block 120 blocks at 50 percent is 120 times 0.75 equals 90 Ether.

Similarly, how many blocks can be blocked with a certain amount of Ether can be calculated. Suppose the attacker has 110 Ether and wants to block 60 percent of the transactions as long as possible. The cost of blocking 60 percent of the transactions is 1.10 Ether/block on average. The attacker's funds of 110 Ether have to be divided by 1.10 Ether/block, which results in 100 blocks. Assuming about 4 blocks per minute, this would result in an attack time of about 25 minutes.

5.4 GasPrice Development under Congestion

The previous section examined the cost of blocking a certain portion of transactions in a block. However, it did not take into account that the gasPrice of transactions can be affected by the congestion that occurs. Congestion in this context refers to the increased volume of transactions caused by high-profile or famous smart contracts that everyone wanted to interact with. The focus of this section is to analyse the magnitude of these additional transactions and the impact on the gasPrice.

5.4.1 Congestion

In this work, the congestion with respect to a set of smart contracts is defined as the sum of the gasUsed of transactions to these smart contracts divided by the gasLimit of the block in which they were processed. For example, consider a block with a blockLimit of 100. This block contains three transactions with 10 gasUsed each interacting with one certain smart contract. Furthermore, in this block are other transactions that interact with another smart contracts. This results in a congestion related to the smart contract of 3 * 10 / 100, which is 0.30 for this block.

block	tx	gasUsed per tx	gasUsed per block	blockLimit	congestion
A	50	2	100	100	1
B	30	5	150	200	0.75

Table 5.4: Example for the Calculation of Congestion in Block A and B

The congestion is a good value to compare different blocks. Assume that two different blocks are compared as shown in table 5.4 In one block A, there are 50 transactions to smart contract S1. In block B, there are only 30 transactions to another smart contract S2. Therefore, one could conclude that block A is more affected by transactions to S1 than the other block is by the transactions to S2, since 50 is larger than 30. However, the error here is that the gasUsed were not considered. Assuming the transactions to S1

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

now have a gasUsed of 2 and those of transactions to S2 a gasUsed of 5. So one could conclude that period B is more affected because 100 < 150. Here, too, another factor has not been taken into account. The gasLimit in block A is only 100, whereas the gasLimit in block B is 200. The congestion in block A is 100 / 100 = 1 and in block B 150 / 200 = 0.75. The congestion definition in this thesis takes these circumstances into account and allows for comparing transactions of different sizes in blocks with different gasLimits.

5.4.2 Congestion Periods Analysis

Looking at the Etheruem Blockchain, there are several possible candidates, which can be called a congestion phase. In the following, three examples are selected and examined in more detail.

CryptoKitties

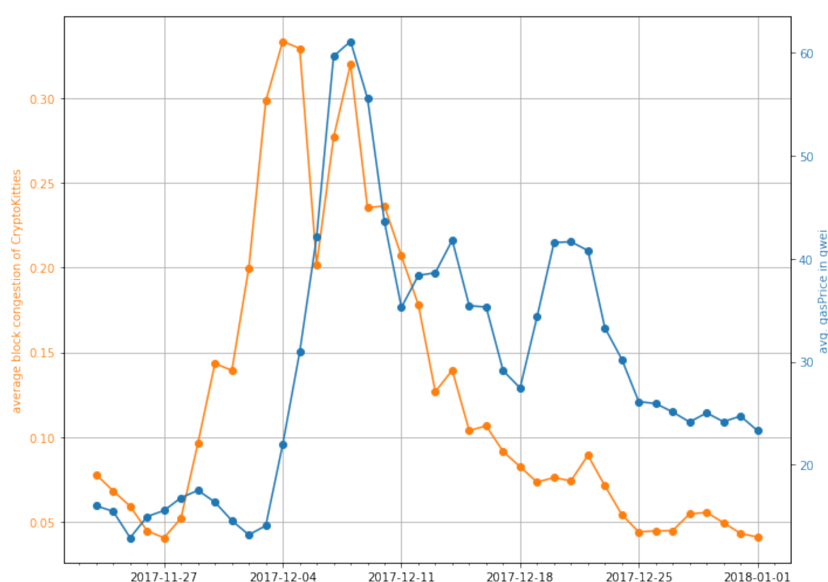

Figure 5.6: Congestion during Peak Transaction Volume of CryptoKitties after its Launch in 2017

One example of this is the rush to the game CryptoKitties [3] after its launch [31]. Figure 5.6 shows the Transactions the CryptoKitties Contract on 0x06012c8cf97bead5deae237070f9587f8e7a266d from November 27th of 2017 to Jan 6th of 2018. It shows how the congestion of transactions to the game's smart contract increases after the official start of the launch in December 2017 to almost 100,000 calls per day. Furthermore, it also shows the gasPrice development, which increases strongly at the same time. It is only after December 8th that the congestion related to CryptoKitties starts to decrease again.

[3] The analysis of CryptoKitties focuses on the blocks 4566385 to 4838611 of the Ethereum main network.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

During the peak congestion, the gasPrice quadruples to 60 gwei compared to about 15 gwei before.

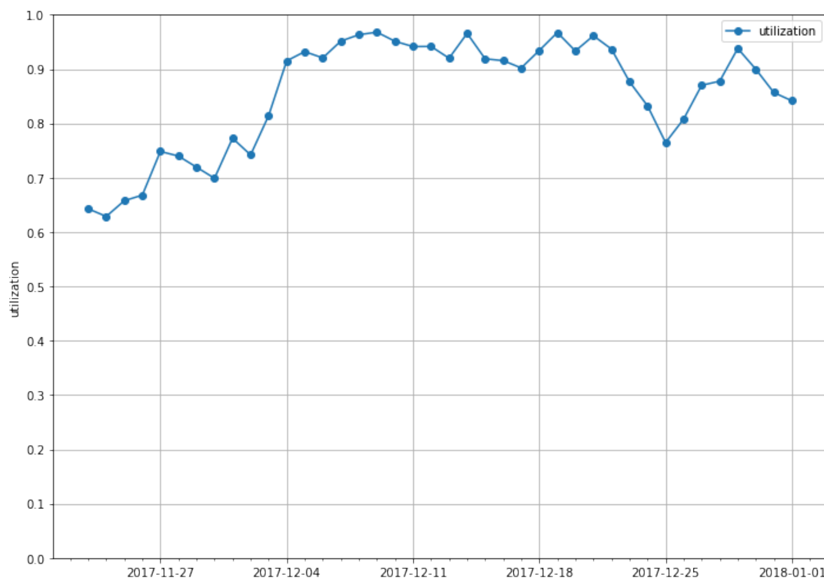

Figure 5.7: Block Utilization during the CryptoKitties Lauch

Noticeable in Figure 5.6 is the lag between the increased transaction volume from CryptoKitties and the rising gasPrices. One reason could be that the blocks were not yet full at the time. Since ordinary transactions still find space in unfilled blocks, gasPrices do not have to be increased to be processed. Figure 5.7 confirms this assumption. It shows the block utilization over the same period. In the beginning, the block utilization is around 0.6, slowly rising with the increased congestion caused by CryptoKitties. Only from the 4th of December, where the block utilization rises to over 0.9, the gasPrice starts to rise as well.

Figure 5.8 compares the gasPrice of transactions addressed to CryptoKitties with the gasPrice of other transactions. First, the transactions to CryptoKitties have a much lower gasPrice. With the increasing congestion, starting from December 3rd, the gasPrice of both increases strongly and stays at a similar level. By December 11th, the gasPrice of CryptoKitties transactions is again visibly below the level of the other transactions. The overall level of the gasPrice remains elevated after the peak volume compared to the beginning of the period.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

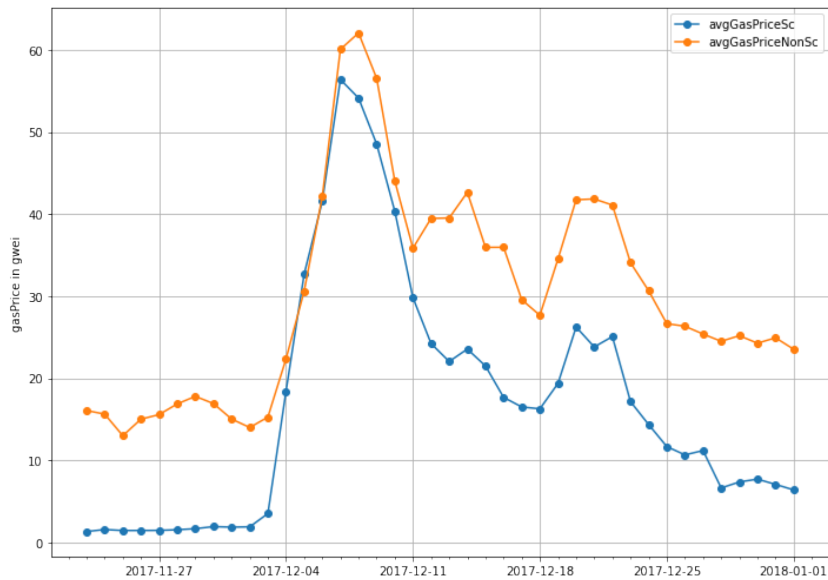

Figure 5.8: GasPrice of CryptoKitties and other Transactions compared during the CryptoKitties Launch

Fomo3D Long

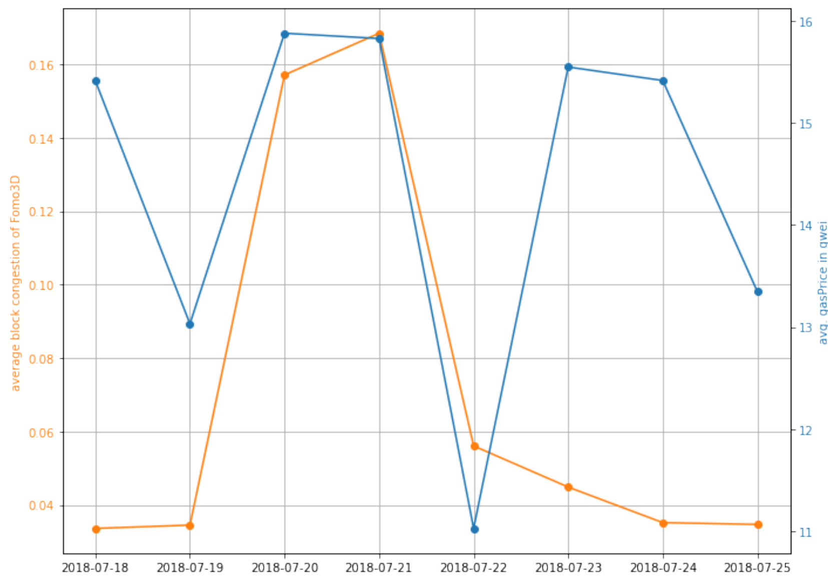

Figure 5.9: Congestion during Peak Transaction Volume of Fomo3D Long Smart Contract in July of 2017

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

The Fomo3D[4] smart contract is another example of high transaction volume, which has already been mentioned in this thesis. This game allows buying of keys as part of a Ponzi scheme. The smart contract distributes the revenue from later key purchases to owners of earlier keys. This incentivizes players to buy these keys as early as possible. Figure 5.9 shows the data of the Fomo3D Long smart contract at the address 0xa62142888aba83707 42be823c1782d17a0389da1 from July 18th to 25th of 2018. The selected period covers the peak volume of transactions to the smart contract. The congestion increases to over 0.16 and then decreases sharply. As the number of transactions associated with Fomo3D increases, so does the average gasPrice in the network. However, it can be observed that the gasPrice before and after the peak congestion is at a similar level as during the peak. Therefore, it can be assumed that the congestion was too low to have a significant impact on the gasPrice development.

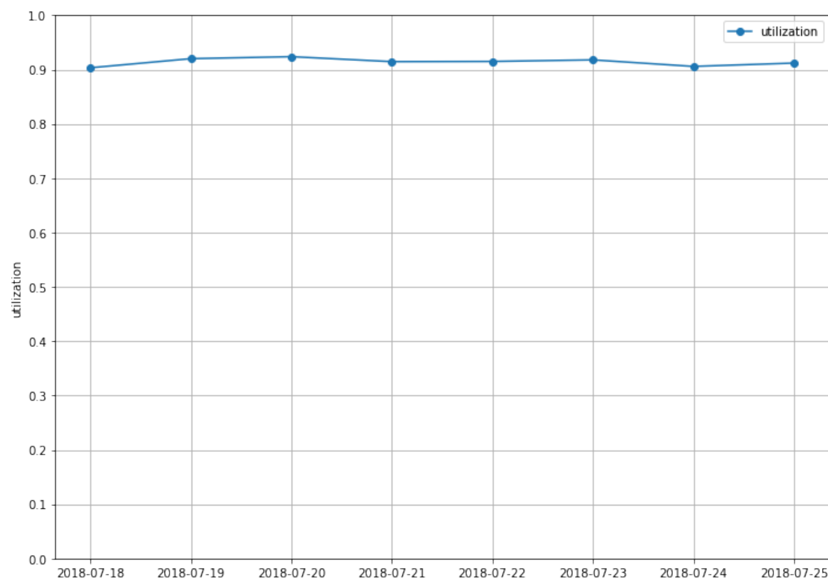

Figure 5.10: Block Utilization during the Peak Transaction Volume of Fomo3D Long in July of 2017

Besides the lower congestion, a lower block utilization could also be a reason for the gasPrice hardly being influenced. Figure 5.13 shows the block utilization, which is above 90 percent over the entire period. Due to this high block utilization, the low congestion is more plausible as a cause for the weak influence on the gasPrice.

The figure 5.11 shows the gasPrice comparison between transactions to Fomo3d and other transactions. The gasPrice of the transactions to Fomo3D rises from under 6 gwei on July 18 to over 12 gwei by July 20 and slowly starts to fall after July 21. The gasPrice of other transactions, on the other hand, moves at an elevated level compared to Fomo3D in a range from 11 to about 16 gwei.

[4]The analysis of Fomo3D focuses on the blocks 5983399 to 6030335 of the Ethereum main network.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

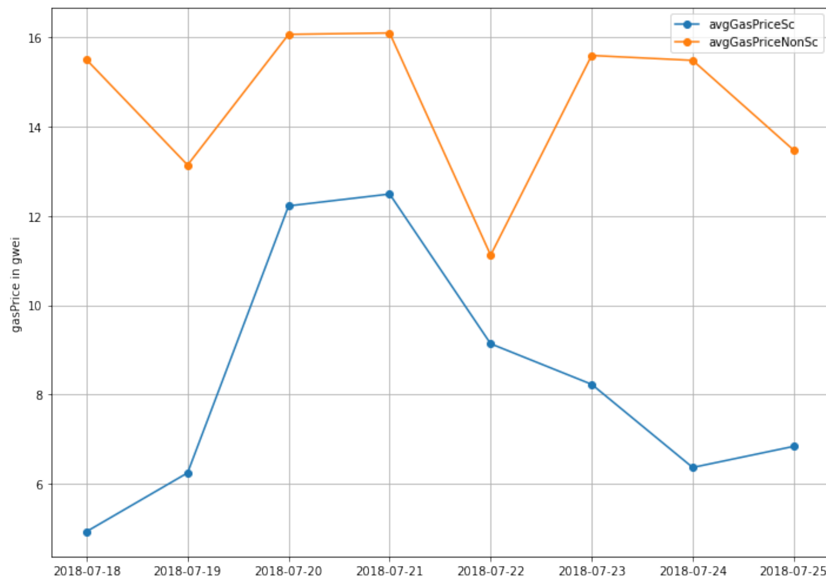

Figure 5.11: GasPrice of Fomo3D and other Transactions compared during the Peak Transaction Volume of Fomo3D Long in July of 2017

Uniswap

The third case of a possible congestion is Uniswap [32]. This decentralized exchange consists of several smart contracts:

- **Router 2.** The Uniswap router allows for swapping of ERC-20 Tokens. Deployed on 0x7a250d5630b4cf539739df2c5dacb4c659f2488d.
- **Token Distributor.** Manages the distribution of UNI Tokens. Deployed on 0x09 0d4613473dee047c3f2706764f49e0821d256e
- **UNI Token.** An ERC-20 token. Deployed on 0x1f9840a85d5af5bf1d1762f925bdad dc4201f984.

Of interest for this work is the time period of the introduction of the UNI-Token[5] on September 16th 2020 [33][34]. For this, the combined volume of the UNI Token and Token Distributor smart contracts are examined. Figure 5.12 shows the congestion and overall gasPrice development on the period before and during the launch. A close relationship between the two variables can be observed. The combined congestion caused by the two contracts increases to over 0.2 in just one day. With the increase in transactions, the gas price also increases from about 150 to 200 gwei to almost 550 gwei, which is more than

[5] This part of the analysis of Uniswap focuses on the blocks 10863238 to 10902386 of the Ethereum main network.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

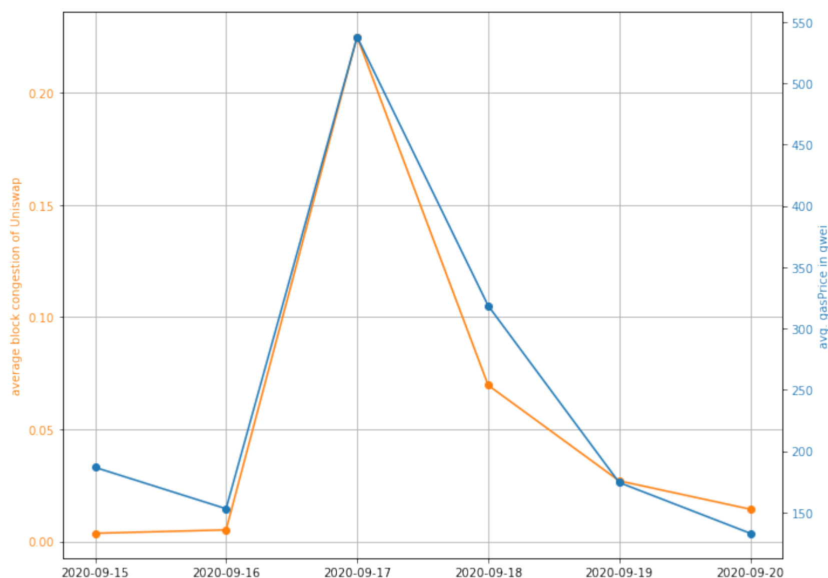

Figure 5.12: Congestion during Peak Transaction Volume of Uniswap Token Launch in September 2020

2.8 to 3.7 times higher than before. It can be said that the Uniswap token launch has had an impact on the gasPrice.

It should be noted that the router contract also received an increase from 130 thousand transactions to 180 thousand transactions on September 16, 2020. However, this was not taken into account as it is more often subject to similar fluctuations.

Figure 5.13 shows the average block utilization during the period. The block utilization is above 97 percent for the entire period. This shows that many transactions were processed before the peak volume phase.

Congestion Phases Compared

Fomo3D had almost no impact on the gasPrice. As the block utilization at the time was relatively high, this may be due to the low congestion related to the contract. CryptoKitties did manage to generate higher levels of congestion, and a strong gasPrice increase was evident. However, at the time, the block utilization was low in the beginning, and the gasPrice was only increased once the block utilization reached over 0.9. Current blockchain conditions at the time of writing show a very high block utilization in the network. Uniswap's token launch shows similar levels and, therefore, better reflects the current situation. To get a better insight on how a congestion could look like in these conditions, Uniswap will be further analyzed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Wien Bibliothek Your knowledge hub

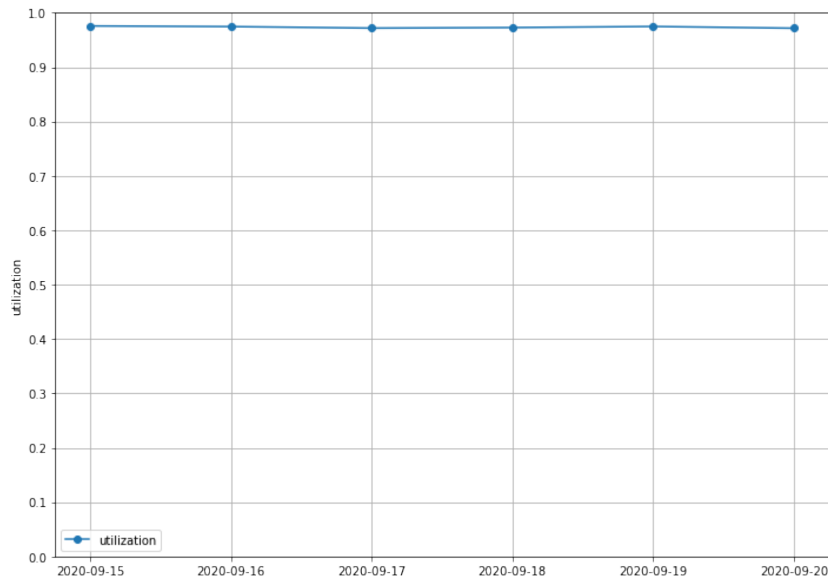

Figure 5.13: Block Utilization during Peak Transaction Volume of Uniswap Token Launch in September 2020

5.4.3 Analyzing Uniswap Launch

The Uniswap token launch is further examined. In particular, a closer look will be taken on September 17, when the smart contracts reached maximum congestion[6]. Furthermore, it will be examined whether the gasPrice of the transactions to the smart contract develops differently from the gasPrice of other transactions.

Figure 5.14 shows the hourly development of congestion and gasPrice on September 17. A substantial increase in congestion and gasPrice from 0:00 to 2:00 can be observed. From 03:00 to 09:00, the congestion moves between 0.25 and 0.4. The gasPrice at this time ranges between 500 and 750 gwei. At 10:00, the congestion moves back below 0.25 and drops to slightly above 0.1 by the end of the day. The gasPrice decreases a bit slower at this time from about 680 to about 440 gwei.

The average gas price for the previous day was 152.92 gwei. This value can now be taken as a reference value, and the period from 02:00 to 09:00 where the congestion was above 0.25 can be considered. At this time, the average congestion was at 0.33, and the average gasPrice was 4.34 times higher than the reference value.

Another question that remains open is whether the gasPrice of the transactions to the smart contracts also develops similarly to the gasPrice of the other transactions. Figure 5.15 shows exactly the development of the gasPrice for these two different sets

[6]This part of the analysis of Uniswap focuses September 17, including the blocks 10876243 to 10882832 of the Ethereum main network.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

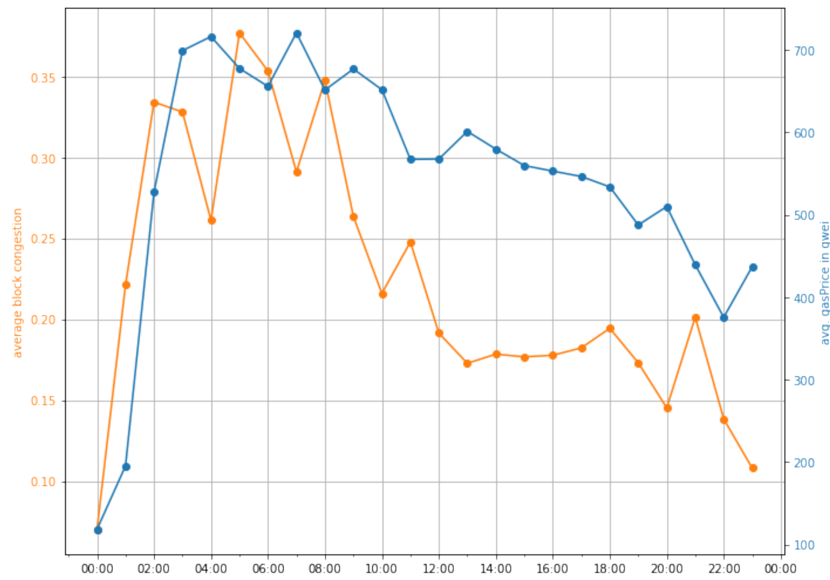

Figure 5.14: Hourly Congestion of Uniswap on September 17, 2020

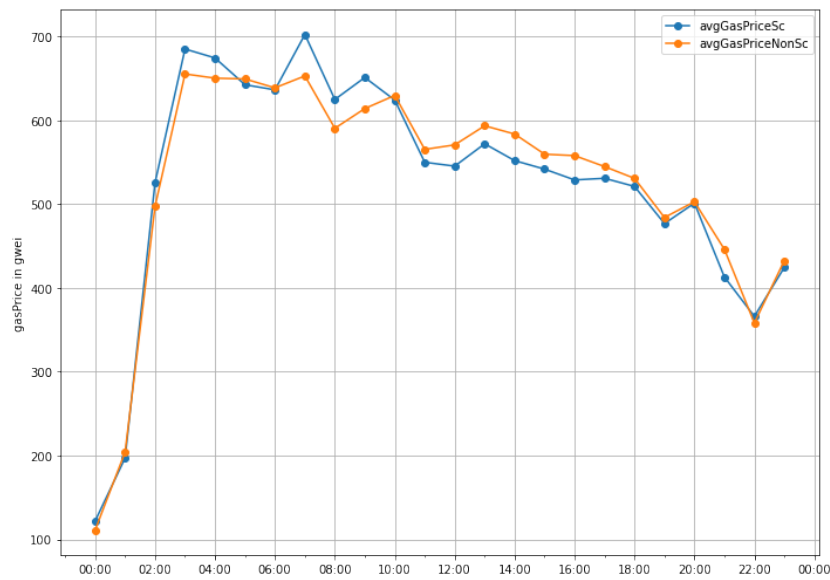

Figure 5.15: GasPrice of Uniswap and other Transactions compared from September 17th 2020

of transactions. It becomes apparent that in the case of the Uniswap token launch, the gasPrice of all transactions has developed in approximately the same way.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

5.5 Estimated Cost Comparison

In this section, the costs of the attack variants are estimated and compared. For this purpose, a scenario is assumed in which similar conditions prevail to those in which the Uniswap token was introduced. On this basis, parameters are selected for the smart contracts of the attack phase variants. Then, the costs associated with the parameterized smart contracts are compared.

5.5.1 Scenario

Consider that the accumulation phase of the Race to the Door attack has been completed and that the collected funds will now be used for an attack. Furthermore, it is assumed that the block utilization is above 0.9 during the entire period. The block limit is assumed to be 15M gas. The attacker aims to generate a congestion of 0.33 with the attack. This means for a targeted congestion, on average, 15M gas * 0.33 $\approx$ 5M gas per block has to be blocked. Similar to Uniswap, gasPrice is assumed to be 153 gwei before the attack. Since the congestion is also at the same level as Uniswap, the gasPrice is expected to increase to 663 during the attack. The block time (i.e., the interval in which blocks are mined) is assumed to be 15 seconds.

5.5.2 FreeMoneyGrab

```
1  block_limit = 15_000_000
2  gas_price = 663
3  congestion = 0.33
4
5  gas_per_block = block_limit * congestion
6  # 4950000.0
7
8  # calculate blockHeapSize
9  money_grab_call_costs = 250_000
10 blockHeapSize = round(gas_per_block / grab_call_costs)
11 # 20
12
13 # calculate costs per block
14 reward_percentage = 0.1
15 block_tx_fees = money_grab_call_costs * blockHeapSize * gas_price
16 block_rewards = block_tx_fees * reward_percentage
17 block_costs = block_tx_fees + block_rewards
18 block_costs = round(block_costs / pow(10,9), 4)
19 # 3.61
20
21 # calculate costs for 1 hour
22 costs = block_costs * 60 * 4
23 # 866.4
```

Listing 5.1: Calculating blockHeapSize, Costs per Block and Costs for 1 hour of the Attack using FreeMoneyGrap in the given Scenario

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU Bibliothek WIEN Your knowledge hub

FreeMoneyGrap has a parameter blockHeapSize which can be used to set the number of transactions to be paid out per block. The attacker knows that the payout function of FreeMoneyGrap consumes on average about 250k gas. To block 5M gas per block, about 20 calls are necessary since 20 * 250k gas = 5M gas. The blockHeapSize parameter of FreeMoneyGrap must therefore be set to 20. This calculation is shown in listing 5.1 in lines 1 to 11.

Another parameter to be set is the amount of reward to be paid out on top of the refund of the transaction fee. In the context of this cost estimation, the *reward_percentage* is set at 10 percent of the transaction costs. This allows calculating the *block_costs* as shown in lines 13 to 19. To do this, the paid transaction fees per block (*block_tx_fees*) are calculated, which is the product of the call costs, the number of calls paid out (*blockHeapSize*), and the *gas_price*. Then the additional reward costs per block (*block_rewards*) are calculated by multiplying the previous result with the *reward_percentage*. Adding the fees and rewards and converting the resulting number from gwei to Ether, the *block_costs* for FreeMoneyGrap in this scenario totals 3.61 Ether.

An estimation of the costs for an Attack of 1 hour is done in lines 21 to 23. A block time of 15 seconds results in 4 blocks every minute or 240 blocks per hour. The hourly costs are calculated by the product of the *blocks* and *block_costs* totaling 866.4 Ether for 1 hour.

5.5.3 Fomo10x

```
1  block_limit = 15_000_000
2  gas_price = 663
3  congestion = 0.33
4
5  gas_per_block = block_limit * congestion
6  # 4950000.0
7
8  # calculate txPerBlock
9  fomo10x_call_costs = 30_000
10 txPerBlock = round(gas_per_block / fomo10x_call_costs)
11 # 165
```

Listing 5.2: Calculating txPerBlock parameter of Fomo10x in the given Scenario

For Fomo10x the parameter txPerBlock must be calculated, which is done in a similar fashion as shown in listing 5.2. The parameter determines by how much the internal timer is counted down per past block. Since the timer is increased by 1 per call, at least txPerBlock transactions are required per block to prevent the timer from running out. The play function consumes about 30k gas per call. With 165 calls, 165*30 $\approx$ 5M gas will be generated. Hence 165 is taken as the value for the txPerBlock parameter.

The duration of Fomo10x depends on the game dynamics of the players and is therefore difficult to estimate. Assuming the 1 hour costs of FreeMoneyGrap of 866.4 Ether are

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

offered as a reward, it is still questionable this amount is a sufficient incentive to maintain the number of transactions for a long time. At the time of writing, the Fomo3D Long smart contract offers a reward of 1228 Ethern and needs far fewer transactions to keep the timer up. Unlike Fomo3D Long, Fomo10x does not require any additional fee from the player to play and emerge as a potential winner.

5.5.4 PayForEmptyBlock

```
1  block_limit = 15_000_000
2  gas_price = 663
3  congestion = 0.33
4
5  # calculate block reward size
6  reward = block_limit * gas_price
7  reward = round(reward / pow(10,9), 4)
8  # 9.945
9
10 # calculate costs for 1 hour
11 avg_block_costs = reward * congestion
12 costs = round(avg_block_costs * 60 * 4, 4)
13 # 787.644
```

Listing 5.3: Calculating Reward Size and Costs for 1 hour of the Attack using PayForEmptyBlocks in the given Scenario

PayForEmptyBlock can only control the level of congestion via the amount of the offered reward. Profit-oriented miners would mine on empty blocks as long as the offered reward is greater than the transaction fees to be collected. However, by not processing the transactions, congestion occurs, raising the transaction fees, and thus the miners start processing transactions again. By processing transactions, the congestion decreases again, and the transaction fees decrease again, making the rewards from the empty blocks seem more attractive again. Thus, the miner will swing back and forth between processing transactions and mining empty blocks.

By providing a block reward equal to the transaction fees of a block in time of congestion, the miners are incentivized to produce empty blocks until the desired congestion level is reached and the processing of ordinary transactions becomes profitable again. To calculate the *reward* offered by PayForEmptyBlocks the $block_limit$ has to be multiplied with the *gas_price* under congestion as shown in listing 5.3 in lines 5 to 8. This scenario results in a *reward* of 9.945 Ether for submitting an empty block.

For the sake of comparison, it is worth mentioning that the gas cost for a PayForEmptyBlock call is 68k gas. Since the call must happen in a timely manner as the contract only has access to the last 256 block hashes, this results in a transaction cost of 0.0045 Ether during the attack.

PayForEmptyBlock offers a reward of 9,945 Ether per empty block. This should incentivize miners to leave about every 3rd block empty and instead accept the *reward*

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

from the smart contract. The *costs* for 1 hour can be calculated by first calculating the *average_block_costs*, which is the product of *reward* and *congestion*. These *average_block_costs* can then be multiplied by the number of blocks of 240 for 1 hour, resulting in a total *costs* of 787.644 Ether.

5.5.5 Discussion

PayForEmptyBlock appears to be the cheaper option compared to FreeMoneyGrap to generate the same amount of congestion. If Fomo10x were given the same amount as a potential reward as the others, it might be the cheapest option if it can run for more blocks than the others can pay.

The prices with the current exchange rate in Euro seem to be high. However, due to the Race to the Door effect, the exchange rate could fall in the event of an approaching attack. In that case, the costs in Euro would look cheaper.

Costs are low compared to the volume. At the time of writing, the trading volume of Etheruem reaches about 24M Ether daily[7]. The cost of less than 1,000 Ether per hour is relatively low compared to the volume.

The cost might be too high for a single person, but several attackers could join forces. As mentioned above, a group of attackers could join forces for the attack. Together they could deposit their funds into the RedeemVoucherCon in the funding currency. They could also use threshold signatures to issue the vouchers so that no single party has access to all the funds. Thus, more funds could be made available for the attack.

Another cost advantage of the Race to the Door attack is that there are no high acquisition costs and only low operational costs. The attack does not have to buy or rent mining hardware. Only the smart contracts have to be deployed and the funds deposited. Costs are only generated by the payout of the rewards to the transaction issuer or empty block miner. These costs are based only on the displacement of space in the block (gasUsed) and the gasPrice and therefore have little overhead.

[7] https://www.coingecko.com/en/coins/ethereum

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

CHAPTER 6

Conclusion and Future Work

The goal of this work is to show that Race to the Door attacks, previously considered buy-out attacks in a PoS context, are also possible in a PoW context. The thesis thus demonstrates the potential threat posed by the execution of such an attack to holders and smart contract users of the targeted cryptocurrency. To investigate the feasibility of the Race to the Door attack, it was evaluated based on its technical implementation (RQ1), the associated costs (RQ2), and the underlying transaction ordering behavior of miners, as well as possible exploits (RQ3).

First, a general system model for RTTD attacks on PoW cryptocurrencies was presented to give an overview of the attack. For this purpose, the required actors and terms were introduced. Furthermore, the attack was divided into a preparation phase, race phase, and attack phase. In the preparation phase, smart contracts are set up on the funding currency and the target currency, and the attack is announced. In the following race phase, users of the target currency have the opportunity to exchange their target currency for the funding currency offered by the attacker with the help of a voucher mechanism. The collected funds will be used for a later attack. By exchanging, more and more target currency is collected for this purpose, which increases the likelihood of a devastating attack. This creates a vicious circle that encourages even more people to trade. The exchange mechanism represents the exit door to escape the attack, which is why it is called the Race to the Door attack. In the final attack phase, the total funds are used for a DoS attack. In-band payments are used to incentivize either the triggering of additional transactions or to get miners to create empty blocks.

In order to demonstrate the technical feasibility (RQ1), concrete technologies were selected in which the phases of the attack are implemented. In this context, Ethereum was chosen, and the smart contracts were implemented in Solidity. For the race phase and its voucher mechanism, two smart contracts were implemented, which allow the secure request and issuance of vouchers in the target currency and the redemption in the funding currency. For the implementation of the attack phase, three different variants were

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

implemented, two of them presenting possible exploits of in-band transaction ordering attacks (RQ3). FreeMoneyGrab incentivizes transaction triggering by reimbursing callers for their transaction costs and also paying a bonus. Fomo10x is a game based on the well-known game Fomo3D, which also provides incentives for transaction triggering by paying out the collected funds to a winner. Furthermore, PayForEmptyBlocks was introduced, which pays miners a reward for submitting empty blocks. These smart contracts implementations demonstrate the technical feasibility of the Race to the Door attack and the challenges involved.

Finally, an empirical analysis was performed on data from the Ethereum blockchain. First, the transaction ordering (RQ3) was analyzed, on which the blocking of transactions by means of transactions is based. It was found that despite high block utilization, which was 97 percent, about 2 percent of the blocks are empty, which could indicate SPV mining. It was found that 31.90 percent of the blocks contain transactions that are strictly sorted by gasPrice. By taking into account the nonce in the sorting, which imposes a strict order of transactions per account, this amounts to 79.41 percent of blocks with sorted transactions. If the transactions are ignored, which were issued by miners themselves and are possibly treated preferentially, the result increases to 90.74 percent sorted blocks. The data shows that the miners act profit-oriented and prefer transactions with higher gasPrice to potentially receive more transaction fees.

For the collected data set, the costs that would arise from blocking a certain proportion of transactions were also analyzed. A linear increase was observed up to around 90 percent of blocked transactions. Blocking all transactions in one block by using other transactions results in an average cost of more than 3 times the cost of blocking 90 percent of the transactions. With the average transaction fees per block of 1.27 Ether observed over the period, it makes sense to block a maximum of 60 percent, which would cost 1.10 Ether. Instead of increasing blocking from this point, it would make more sense to pay the miner directly to create empty blocks and offer him a reward covering the lost transaction costs plus a bonus.

Furthermore, the development of the gasPrice under contract-induced congestion was investigated. Congestion in relation to a set of smart contracts was defined as the gasUsed of transactions within a block addressed to these popular smart contracts in relation to the block limit. This measure was used to compare different congestion phases. For CryptoKitties, the congestion caused the gasPrice to quadruple. It was found that the Fomo3D Long start had little influence on the gasPrice. In more detail, the Uniswap token launch was examined, which also led to a quadrupling of the gasPrice due to its congestion.

Based on the empirical data collected, the cost of the attack variants was estimated. A scenario was considered in which circumstances similar to those of the Uniswap token launch were selected to further analyze the costs of the attack (RQ2) and compare the different attack variants. For this purpose, the smart contracts were parameterized to set their incentives to achieve a similar level of congestion as in the token launch. Subsequently, the attack duration of the different variants was compared at which they

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

can maintain this level of congestion. The costs associated with creating congestion for 1 hour with FreeMoneyGrab, are 870 Ether. PayForEmptyBlocks can bring the costs down to 790 Ether per hour. The attack costs of Fomo10x depend on the game dynamics and are therefore difficult to compare with the others.

Thus it was shown that a Race to the Door attack in a PoW context is technically feasible. Hourly operational costs for launching the attack amount to roughly 0.002 percent of the 24h trading volume (or 0.047 percent of hourly trading volume)[1]. The intensity of the attack can be adjusted by parameters. The duration depends on the degree of congestion and the funds collected in the race phase.

6.1 Future Work

This thesis has analyzed the gasPrice developed under times of congestion induced by popular contracts. However, it is not clear how the gasPrice develops under different levels and categories of congestion, for example, a general increase in demand. Furthermore, it could be investigated how gasPrice developments affect the exchange rates of the involved cryptocurrencies or how the exchange rate of cryptocurrencies relate to each other.

With the help of a framework as presented in [12] the Race to the Door attack could be categorized and compared to other attacks.

Furthermore, other attack variants of the attack could be explored. For example, the utilization of threshold signatures in the preparation phase or different techniques to optimize the funding of DDoS attacks on the target cryptocurrency could be analyzed.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[1]Calculation based on 24h trading volume from https://www.coingecko.com/en/coins/ethereum converted to Ether at the time of writing.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Figures

3.1 Holders and Currencies of the Race to the Door 9
3.2 Race Phase - Initial State 11
3.3 Race Phase - Progress State 12
3.4 Race Phase - Race to the Door Effect 12
3.5 Race Phase - Final State 13

4.1 Simplified Voucher Mechanism of the Race Phase Implementation 18
4.2 Requesting and Issuing of a Voucher in the Target Currency and Redemption in the Funding Currency. 19

5.1 Data Model for Ethereum Blockchain Data 39
5.2 Average Block Utilization per Day 41
5.3 Average Costs of Blocking a Portion of Transactions 43
5.4 Relative gasUsed per Bucket 44
5.5 Average Costs of Blocking a Portion of Transactions including Median in Ether 45
5.6 Congestion during Peak Transaction Volume of CryptoKitties after its Launch in 2017 47
5.7 Block Utilization during the CryptoKitties Lauch 48
5.8 GasPrice of CryptoKitties and other Transactions compared during the CryptoKitties Launch 49
5.9 Congestion during Peak Transaction Volume of Fomo3D Long Smart Contract in July of 2017 49
5.10 Block Utilization during the Peak Transaction Volume of Fomo3D Long in July of 2017 50
5.11 GasPrice of Fomo3D and other Transactions compared during the Peak Transaction Volume of Fomo3D Long in July of 2017 51
5.12 Congestion during Peak Transaction Volume of Uniswap Token Launch in September 2020 52
5.13 Block Utilization during Peak Transaction Volume of Uniswap Token Launch in September 2020 53
5.14 Hourly Congestion of Uniswap on September 17, 2020 54
5.15 GasPrice of Uniswap and other Transactions compared from September 17th 2020 54

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

List of Tables

4.1 Initial State of a Mempool forming a Candidate Block 24
4.2 Updated State of a Mempool including a Transaction X created to block Transaction C and D 25
4.3 Initial State of FreeMoneyGrab with Transactions A to D 28
4.4 State of FreeMoneyGrab after Transaction E 28
4.5 State of FreeMoneyGrab after Transaction F 29
4.6 State of FreeMoneyGrab after Transaction G 29
4.7 Fomo10x Game State Changes while processing Transactions 32

5.1 Updated State of a Mempool including a Transaction created by the Attacker with equal gasPrice as the highest displaced Transaction 42
5.2 Updated State of a Mempool including a Transaction created by the Attacker with equal gasPrice as the highest displaced Transaction 42
5.3 Min, Max, Mean and Median of Costs for Blocking certain Percentages of a Block in Ether 45
5.4 Example for the Calculation of Congestion in Block A and B 46

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

Acronyms

AIM Algorithmic Incentive Manipulation. 2, 8

DApps Decentralized Applications. 1

DeFi Decentralized Finance. 1, 2

DoS Denial of Service. 8

EVM Ethereum Virtual Machine. 6, 26, 34, 35

PoS Proof of Stake. 2, 6, 8, 59

PoW Proof of Work. 3, 5, 7, 8, 59, 61

RLP Recursive Length Prefix. 35, 37

SPV Simplified Payment Verification. 40, 41, 60

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.

Bibliography

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," tech. rep., Manubot, 2019.

[2] G. Wood *et al.*, "Ethereum: A secure decentralised generalised transaction ledger," *Ethereum project yellow paper*, vol. 151, no. 2014, pp. 1–32, 2014.

[3] "Cryptocurrency prices, charts and market capitalizations | coinmarketcap." https://coinmarketcap.com/, 2021. [Online; accessed 06-March-2021].

[4] "Defi - the decentralized finance leaderboard at defi pulse." https://defipulse.com/, 2021. [Online; accessed 06-March-2021].

[5] Tesla Inc., "Annual report on form 10-k for the year ended december 31, 2020." https://www.sec.gov/ix?doc=/Archives/edgar/data/1318605/000156459021004599/tsla-10k_20201231.htm, 2021. [Online; accessed 29-March-2021].

[6] "Ark invest | we believe innovation is key to growth." https://ark-invest.com/, 2021. [Online; accessed 29-March-2021].

[7] C. Ballentine, "Cathie wood amasses $50 billion and a new nickname: 'money tree'." https://www.bloomberg.com/news/articles/2021-02-05/cathie-wood-amasses-50-billion-and-a-new-nickname-money-tree, 2021. [Online; accessed 29-March-2021].

[8] "r/wallstreetbets." https://www.reddit.com/r/wallstreetbets/, 2021. [Online; accessed 29-March-2021].

[9] DeepFuckingValue. https://www.reddit.com/user/DeepFuckingValue/, 2021. [Online; accessed 29-March-2021].

[10] C. Gartenberg, "Robinhood blocks purchase of gamestop, amc, and blackberry stock." https://www.theverge.com/2021/1/28/22254102/robinhood-gamestop-bloc-stock-purchase-amc-reddit-wsb, 2021. [Online; accessed 29-March-2021].

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[11] I. Eyal and E. G. Sirer, "Majority is not enough: Bitcoin mining is vulnerable," in *International conference on financial cryptography and data security*, pp. 436–454, Springer, 2014.

[12] A. Judmayer, N. Stifter, A. Zamyatin, I. Tsabary, I. Eyal, P. Gaži, S. Meiklejohn, and E. Weippl, "Sok: Algorithmic incentive manipulation attacks on permissionless pow cryptocurrencies." Cryptology ePrint Archive, Report 2020/1614, 2020. https://eprint.iacr.org/2020/1614.

[13] 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*, vol. 2013, p. 11, 2013.

[14] (Director). G. Hamilton, "Goldfinger," 1964.

[15] J. Bonneau, "Hostile blockchain takeovers (short paper)," in *International Conference on Financial Cryptography and Data Security*, pp. 92–100, Springer, 2018.

[16] P. McCorry, A. Hicks, and S. Meiklejohn, "Smart contracts for bribing miners," in *International Conference on Financial Cryptography and Data Security*, pp. 3–18, Springer, 2018.

[17] L. Zhou, K. Qin, C. F. Torres, D. V. Le, and A. Gervais, "High-frequency trading on decentralized on-chain exchanges," *arXiv preprint arXiv:2009.14021*, 2020.

[18] 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)*, pp. 910–927, IEEE, 2020.

[19] S. Eskandari, S. Moosavi, and J. Clark, "Sok: Transparent dishonesty: front-running attacks on blockchain," in *International Conference on Financial Cryptography and Data Security*, pp. 170–189, Springer, 2019.

[20] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth, "Bar fault tolerance for cooperative services," in *Proceedings of the twentieth ACM symposium on Operating systems principles*, pp. 45–58, 2005.

[21] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Ethdkg: Distributed key generation with ethereum smart contracts." Cryptology ePrint Archive, Report 2019/985, 2019. https://eprint.iacr.org/2019/985.

[22] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd ed., 2009.

[23] D. R. Morrison, "Patricia—practical algorithm to retrieve information coded in alphanumeric," *Journal of the ACM (JACM)*, vol. 15, no. 4, pp. 514–534, 1968.

Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.
TU WIEN Bibliothek Your knowledge hub

[24] S. Edelkamp, "Patricia tree." https://xlinux.nist.gov/dads/HTML/patriciatree.html. [Online; accessed 17-May-2021].

[25] "Solidity - units and globally available variables." https://docs.soliditylang.org/en/v0.5.3/units-and-global-variables.html#block-and-transaction-properties. [Online; accessed 17-May-2021].

[26] A. Coglio, "Ethereum's recursive length prefix in acl2," *Electronic Proceedings in Theoretical Computer Science*, vol. 327, p. 108–124, Sep 2020.

[27] M. J. Dworkin, "Sha-3 standard: Permutation-based hash and extendable-output functions," 2015.

[28] "Ethereum json-rpc api." https://ethereum.org/en/developers/docs/apis/json-rpc/. [Online; accessed 17-May-2021].

[29] "go-ethereum block header." https://github.com/ethereum/go-ethereum/blob/master/core/types/block.go#L69. [Online; accessed 17-May-2021].

[30] B. Research, "Empty block data by mining pool." https://blog.bitmex.com/empty-block-data-by-mining-pool/, 2017. [Online; accessed 11-April-2021].

[31] O. Kharif, "Cryptokitties mania overwhelms ethereum network's processing." https://www.bloomberg.com/news/articles/2017-12-04/cryptokitties-quickly-becomes-most-widely-used-ethereum-app, 2017. [Online; accessed 16-May-2021].

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

[33] "Introducing uni." https://uniswap.org/blog/uni/. [Online; accessed 17-May-2021].

[34] S. Edelkamp, "Ethereum fees spike after uniswap's uni token launch." https://cryptobriefing.com/ethereum-fees-spike-uniswaps-uni-token-launch/, 2020. [Online; accessed 17-May-2021].

TU WIEN Bibliothek Your knowledge hub
Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar
The approved original version of this thesis is available in print at TU Wien Bibliothek.