CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

INDAGO

A tool designed to track illicit funds on the Ethereum blockchain

Master's thesis in Computer science and engineering

Max Arfvidsson Nilsson & Pontus Backman

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2023

Master's thesis 2023

INDAGO

A tool designed to track illicit funds on the Ethereum blockchain

Max Arfvidsson Nilsson & Pontus Backman

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2023

INDAGO
A tool designed to track illicit funds on the Ethereum blockchain
Max Arfvidsson Nilsson & Pontus Backman

© Max Arfvidsson Nilsson & Pontus Backman, 2023.

Supervisor: Ahmed Hassan, Computer Science and Engineering
Advisors: Dan Gorton & Leonard Saers, Handelsbanken's innovation department
Examiner: Robert Feldt, Computer Science and Engineering

Master's Thesis 2023
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2023

INDAGO
A tool designed to track illicit funds on the Ethereum blockchain
Max Arfvidsson Nilsson & Pontus Backman
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

A seemingly never-ending issue with cryptocurrencies is their association with illegal activities. In 2020, it was estimated that roughly 3% of the transaction volume of Bitcoin consisted of transactions performed by known illicit actors. This is a problem for financial institutions wanting to integrate with cryptocurrencies since they risk incurring large fines if they are found to be complicit in illegal activities. This thesis set out to provide insight into this issue by developing a tool capable of detecting illicit funds on the Ethereum blockchain. By utilising DAR clustering and four different blacklisting algorithms and running them on publicly available Ethereum transaction information, the tool was able to detect approximately 160 million possibly illicit Ethereum addresses at varying levels of suspicion. It was also able to detect 965,719 unique clusters, of which 238,536 contained illicit addresses. The blacklisting algorithms involved had previously been described in the literature, but this is, as far as we know, the first time concrete implementations have been created and tested on real data. The effectiveness of the algorithms was evaluated in isolation and in aggregate. It was found that all of the blacklisting algorithms had possible use cases, though haircut and seniority showed the most potential for use in real-world scenarios as they spread the funds in a desirable way while also having a runtime considerably less than that of FIFO. DAR Clustering in combination with at least one of the blacklisting algorithms also showed potential as it was able to detect illicit addresses inside otherwise clean clusters. The findings of this thesis are limited to Ethereum with only partial generalizability to other cryptocurrencies.

Keywords: Virtual Assets, Blockchain, Ethereum, AML, KYC.

Acknowledgements

We would like to give a big thanks to Dan and Leonard at the Handelsbanken innovation department for all the help throughout the thesis, helping us to brainstorm ideas, and giving feedback on the report.

We would also like to give a big thanks to Ahmed, our supervisor, for all the help with fixing our proposal and setting us on the right path.

Further, we would also like to thank our examiner Robert for the great patience shown as our thesis kept getting delayed.

Finally, we would also like to thank Bjarne for showing interest in our thesis and connecting us with Handelsbanken's innovation department.

Max Arfvidsson Nilsson & Pontus Backman, Gothenburg, 2023

Contents

List of Figures

List of Tables

List of Algorithms

1

Introduction

A major concern often expressed about cryptocurrencies is how they might be used to aid illegal activities. It has been estimated that during 2020, approximately 3% of the Bitcoins transferred between distinct entities were carried out by known illicit actors, resulting in a total of $5.3 billion [1]. Apart from this, roughly 20% of Bitcoin transactions were performed between unknown entities, all of which have the potential of being illicit. The total percentage of Bitcoin transactions involved in illicit activities has decreased over time, but this is only because the retail and investment markets have grown so fast [2]. The dollar value of all illicit Bitcoin transactions continues to rise.

Even though an increasing part of the population owns cryptocurrency, the widespread adoption of cryptocurrency as a means for transferring value has been slow. One of the reasons for this is the fear banks and retailers have of enabling money laundering. This fear stems from the fact that in most countries banks are required to carry out extensive background checks on the money they accept [3]. Failing to do so may result in the bank having to pay fines on the order of billions of dollars [4].

This thesis explains the process of creating an open-source tool that makes it possible to check whether specific assets on the Ethereum cryptocurrency have been involved in illicit activities. The tool was created in collaboration with Handelsbanken's innovation department, who provided key insight into the field of *Anti-Money Laundering* (AML).

The rest of this thesis is organised as follows. This chapter will continue to provide a general introduction to the thesis, after which comes the theory chapter, where general background information about blockchain and cryptocurrencies can be found. Then there is the literature study, which is a collection of summaries of studies relevant to this thesis. Here, different cryptocurrency AML methods are evaluated for use in the final tool. Information from the literature study also serves as background information for the rest of the thesis. After this comes the method chapter, where major design and development decisions are documented. Then comes an implementation details chapter where we talk more about the specifics of implementing the tool. After this comes the results chapter where the data obtained by the tool can be found, along with screenshots showing the final design of the tool. Last is the discussion chapter

where the results are discussed and a conclusion is made.

1.1 Statement of the problem

Banks are obligated to collect a certain amount of *know-your-customer* (KYC) information from their customers before granting them access to their services [5]. This process of collecting KYC information is known as *customer due diligence*. It requires that banks and other financial service providers collect certain information, such as name, address, and passport, on their customers. The main purpose of collecting this information is to prevent money laundering and terrorist financing.

Unfortunately, it is not easy for banks to do their customer due diligence when dealing with the pseudo-anonymity of cryptocurrencies. Instead of dealing with transactions between companies, individuals, and banks, they have to trace money flowing between anonymous addresses [6]. By design, most cryptocurrencies are public, which means that anyone with the right knowledge can look up the balance and transaction history of every address on the blockchain [7]. Although since there are no built-in identification requirements for using cryptocurrencies, most users are still anonymous even though their accounts are public.

The specifics of how transactions are performed with different cryptocurrencies also introduce extra complexity to the task of tracking assets involved in illegal activities. For example, Bitcoin allows its users to transfer assets from several addresses to several others in a single transaction [6]. This means that it is possible to mix the contents of different pools of funds, making it unclear which of the senders contributed to any given receiver. Some cryptocurrencies such as Monero go even further by intentionally trying to obscure the sender of funds using different cryptographic technologies [8].

1.2 Lack of existing research

Most of the published work within AML on cryptocurrencies is focused on Bitcoin. Since its launch, Bitcoin has remained the most popular cryptocurrency, both in terms of valuation and media coverage. However, in recent years, others have begun to catch up. The main competitor in terms of valuation and adoption is Ethereum [9]. While Bitcoin has remained virtually the same since 2009, Ethereum and other *second* and *third-generation* cryptocurrencies have expanded upon the original premise of Bitcoin, providing functionality beyond only being a store of value. *Smart contracts*, an innovation introduced by Ethereum in 2013, have expanded on Bitcoins technology, making it possible to execute code and trigger transactions when the right conditions are fulfilled [10]. This has sparked a wave of innovation that has resulted in decentralised financial services, play-to-earn games, and much more.

Banks wanting to integrate cryptocurrencies into their business will limit themselves both in terms of functionality and customers if they choose to only integrate their

systems with Bitcoin. When compared to Bitcoin, other cryptocurrencies have received a comparatively small amount of AML-related research, and the research that has been conducted is mostly theoretical. As far as we know, there currently does not exist a publicly available AML tool for any cryptocurrency, including Bitcoin, able to track and pinpoint the location of illicit funds live. There are companies such as Chainalysis that claim to have such tools, but these are held behind closed doors and are only available to other companies at a large cost [11].

1.3 Purpose & scope

The purpose of the study was to first evaluate existing methods for detecting illicit cryptocurrency transactions and then to design an open-source AML tool based on the knowledge gained from the evaluation. The tool should be capable of determining if specific assets on an account were at some point involved in illicit activities. To delimit the study, the primary target for the tool would be Ethereum. To further narrow the scope, no *Layer-two solutions* for Ethereum would be considered. Layer-two solutions are external tools that build on top of a cryptocurrency in order to decrease transaction fees or provide other extra functionality. These introduce extra complexity to the task of detecting illicit funds, and will thus be ignored in this thesis in order to narrow the scope.

We believe the novelty of this study will make it valuable to the research community. As far as we know there does not currently exist a publicly available comparison between cryptocurrency AML solutions. Similarly, we have not been able to find a single publicly available AML tool capable of analysing cryptocurrency assets in real-time. All the existing published solutions that we have found are machine-learning models that have only been tested against old data. An open-source tool in which the user enters a wallet address and receives an up-to-date analysis of the legitimacy of the funds in the wallet would be unique. It could also, depending on its accuracy, be of value within both industry and research.

1.4 Research questions

The objectives of the study are divided into three related research questions with RQ2 dependent on RQ1, and RQ3 dependent on RQ2.

RQ1: Are there any existing blockchain analytics solutions in the literature that financial institutions can use to help them comply with AML laws and regulations when acquiring cryptocurrency assets?

RQ2: How can the methods found in RQ1 be implemented and combined in a way that results in an effective blockchain AML tool?

RQ3: From a software engineering perspective, what are the biggest differences when working with blockchain and cryptocurrencies compared to other fields?

2

Theory

This chapter provides the background necessary to understand the concepts discussed later in this report. First, a simple explanation of what a blockchain is will be provided. This transitions into an explanation of cryptocurrencies in general, which is then built upon with a more in-depth look at Ethereum. Lastly, a section on the legal background provides context as to why finding and tracking illicit cryptocurrencies are so important.

2.1 Blockchain

Blockchain is the most central technology in almost every cryptocurrency [12]. Therefore, it is important to understand how a blockchain works in order to grasp concepts that will be discussed later in this thesis. The finer details on how the blockchain of different cryptocurrencies is implemented tend to vary. Thus, the explanation provided here will talk about the inner workings of a blockchain in more general terms, so that the explanation is applicable to any cryptocurrency that utilises a blockchain.

2.1.1 History

Contrary to popular belief, blockchain was not invented by Satoshi Nakamoto in 2008 with the introduction of Bitcoin, but was in fact invented much earlier by Stuart Haber and Scott Stornetta. In 1991 they published a paper titled *"How To Time-Stamp a Digital Document"* [13], in which they describe a form of chained hashing very similar to today's blockchains. One year later, they published a new paper called *"Improving the Efficiency and Reliability of Digital Time-Stamping"* [14], in which they refined their concept through the introduction of *Merkle trees*. Merkle trees will be explained later in section 2.1.5. The problem they were trying to solve was that of proving the validity of digital documents. At that time, companies and agencies were starting to digitise their paper records. Since it is much easier to manipulate digital documents without leaving a trace, Haber and Stornetta saw the need for a system capable of proving the validity of these digital documents. In order to do this, they created a system capable of proving that a document was created at a certain point in time and that it had not been tampered with since its creation. By

1994 they had used the technology to create a startup named Surety Technologies [15].

Blockchain technology went mostly unused up until the publication of the Bitcoin white paper in 2008, where Satoshi repurposed it to keep track of transactions instead of digital documents [6]. Although Satoshi integrated it with several other technologies, in order to make it decentralised, the core principles of Satoshi's blockchain are remarkably similar to the original definitions by Haber et al.

As Bitcoin grew in popularity it inspired the creation of a number of other cryptocurrencies which also utilised blockchain. Perhaps the most noteworthy of these is Ethereum, launched in 2014. Ethereum was the first to expand the scope of decentralised blockchains outside ledger keeping with the introduction of *smart contracts* [10]. Smart contracts are executable programs that can run on a cryptocurrency. The computer code used to execute these smart contracts is usually stored on the blockchain of the cryptocurrency.

2.1.2 A chain of blocks

A blockchain is in its purest form a data structure. As it is implemented in most cryptocurrencies, it serves as the primary place for data storage [12]. As the name suggests, a blockchain is made up of blocks placed in a chain. The blocks are created at different points in time, usually with even time intervals in between. The blockchain created by Surety Technologies published a block once every week, while Bitcoin and Ethereum, for contrast, publish a block every 10 minutes and 13 seconds, respectively [15]–[17]. The chain of blocks can be thought of as a reversed linked list, where each entry holds data along with a reference to the previous entry. In a blockchain, the newest block contains a reference to the second newest block, which contains a reference to the third newest block, and so on. See Figure 2.1 for an illustration of this behaviour. In addition to a reference, each block also contains a certain number of data entries. Exactly how many is defined in the implementation details of the cryptocurrency utilising the blockchain. These data entries can technically be anything, but in cryptocurrencies they are usually transactions or smart contract code [6], [18].

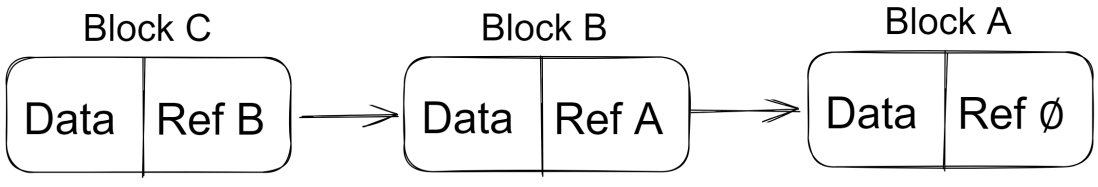

Figure 2.1: *Showing a simplified blockchain where each block contains data along with a reference to the previous block.*

2.1.3 Sequential hashing

Blockchains utilise what can be referred to as *sequential hashing*, which is a way of linking all of the data entries together, making them dependent on each other.

Sequential hashing takes advantage of the one-way properties of *hashing algorithms*. Hashing algorithms are algorithms that take data of any length and convert them into a string of fixed length, called a hash [19]. These algorithms are constructed in such a way that it is orders of magnitude more complex to run them in reverse. Creating a hash from a given input has *P-complexity* while creating an input only knowing the hash has *NP-complexity*. This means that, given an output from a hashing algorithm, there would be no way of knowing what input produced that output except by guessing at random. If the input is made long enough, the chance of guessing the correct input becomes practically impossible. These algorithms are also constructed in such a way that only a small change to the input will completely change the hash. Since these algorithms take inputs of any length and produce an output of fixed length it means that for each possible output, there has to be more than one possible input. However, given a well-constructed hashing algorithm, the chance of discovering such duplicates is very small. Blockchains use hashing for many of their key features, one of them being sequential hashing.

The idea with sequential hashing is to create a hash for each block that combines the data of the block with the hash from the previous block. This creates a dependency between each block and its predecessor, as can be seen in Figure 2.2. Since the hash that is passed on to the next block incorporates all of the data from the current block, it means that any change in the current block will cause a mismatch between the input of the hashing algorithm and the output it produces. Since the hash and the data are required to match, it means that any changes in the data will also require that the hash be recalculated. If this hash has already been passed along and used in the creation of the hash for the next block, the next block's hash also has to be remade.

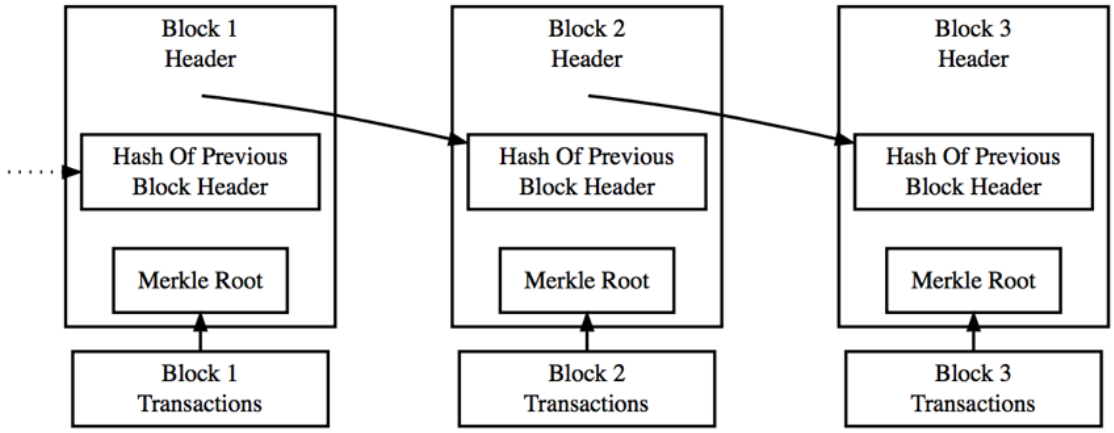

Figure 2.2: *Showing a simplified blockchain with block headers consisting of the previous block hash and a Merkel root. Taken from [20].*

The blocks in Figure 2.2 can be used to illustrate this behaviour. If a change is performed in the data of block 1 this means that its hash no longer matches the block and it has to be recalculated. The new hash is passed along to block 2 so that it has the most up-to-date hash of block 1. Since the hash from block 1 was used in

the creation of the hash for block 2, it means that block 2 also has to regenerate its hash. The new hash for block 2 is passed along to block 3 causing it to remake its hash, etc. This will continue until every block after block 1 in the chain has been updated. What this all means is that if any change is made to a block anywhere in the chain, every block after that block will have to be rehashed. Either that or the data in the block and its corresponding hash will be left unmatching.

This is a key feature since it makes it much harder for anyone to alter the state of the blockchain without being detected. As long as the blockchain data is public, it would be very obvious if the entire chain of blocks was suddenly rehashed. Alternatively, if someone were to change a data entry without rehashing the chain, it would also be easy to spot by simply rehashing the block before accessing any potentially altered data. If the new hash and the hash that was passed along to the next block do not match, it means that one of the data entries inside the block has been altered and that the block should not be trusted. This protects the data from both outside and inside manipulation, meaning that even the blockchain provider does not have to be entirely trustworthy. As long as at least the hash for every block is published, there will be no way for the blockchain provider to change the contents of blocks after publication, without it being detectable by people from the outside.

2.1.4 Blockchain timestamping

Another important feature of blockchain, which is made possible by sequential hashing, is the ability to prove the existence of data at certain points in time without the need for a trusted third party [13]. The blocks in Figure 2.2 can again be used as an example. Since the hash for block 2 uses the hash from block 1, it means that block 1 must have existed before block 2. It would be impossible to create a hash for block 1 before it was created. Similarly, since block 3 uses the hash from block 2, block 2 must have been created before block 3. This is true for all blocks along the chain, which means that the entire blockchain is, in fact, a logical clock that keeps track of the order of events. This is very useful when two conflicting events are present on the same blockchain. Imagine, for example, that the blockchain contains two contracts where both transfer the ownership of the same object. The first contract transfers the ownership from person A to person B and the second transfers it from person A to person C. In this situation, it would be very easy to verify which contract existed first, by looking up which contract was first added to the blockchain.

This provable ordering of blockchain events can also be used to obtain a timestamp for when a data entry was added to the blockchain. If a user wants to get a timestamp for a data entry residing in a specific block, this user can contact other users who have also made data entries in the same block, asking them when they made their entry. This assumes that the contact information for these users is available in some way. The more users who are contacted, the higher the probability that the timestamp given is correct. If there are no other data entries in the block, or if the number of data entries is deemed too few, then data entries in the blocks in front

and behind can also be used. Since blocks in a blockchain are guaranteed to have been created in chronological order, timestamps from the blocks behind and in front can be used to create a time span in which the data entry must have been created.

To explain why accurate time stamping of events is important, let us imagine that someone has signed a document using public-private-key signatures and uploaded the document to a blockchain. At some later time, the private key is stolen. Usually, this would mean that any signature done with that key should be deemed untrustworthy. However since the document was added to the blockchain, the document can still be considered trustworthy as long as the moment of theft is known.

2.1.5 Merkle trees

The simplest possible version of a blockchain would be to have only one data entry per block. The problem is that this would not be very effective since the boilerplate information of the block would require more storage space than the data itself. This is why Stuart et al., already in 1992, updated their blockchain design by introducing *Merkle trees* [14]. Merkle trees, invented by Ralph Merkle in 1988, is a form of combinatory hashing designed to effectively hash a large number of data entries into a single hash [21]. It is used by blockchains to break out of the otherwise linear sequential hashing described above by hashing together several data entries inside a single block, as seen in Figure 2.3. Data entries are first hashed individually and then combined pairwise. The combination of each pair is hashed again, after which new pairs are formed from the resulting hashes. This process is repeated until only one hash remains that combines the hashes from all of the data entries. This hash, called the Merkle root, is then combined with other block information from the block header as seen in Figure 2.2, and hashed one last time to form the block hash that is passed on to the next block. This configuration makes sure that if any of the data in the block is changed, the block hash will also have to change for them to match.

An alternative to the Merkle tree is to take all of the data entries and process them directly into a single hash. The Merkle tree has two major advantages over this. The first is how much data has to be stored in order to prove that a specific data entry exists in a block. If all of the data is directly hashed, it means that all of the data will also be required to recreate the final hash. The Merkle tree setup drastically decreases the data storage requirement. Since the hashing is performed pairwise, users only have to keep track of every opposing hash on the path through the bracket to be able to prove that a specific data entry contributed to the final hash [23]. See L1 in Figure 2.3. To prove that L1 contributed to the top hash, only the resulting hash from hash 0-1 and hash 1 along with the hash of L1 is enough to recreate the top hash. This decreases the data stored from n to $log(n)$, where n is the number of data entries in a block [24]. This means that the advantage for the Merkle tree becomes larger the more data entries there are per block.

The Merkle tree also makes it easier to detect manipulation within a block. If every data entry was hashed directly, there would be no way to know which data entry

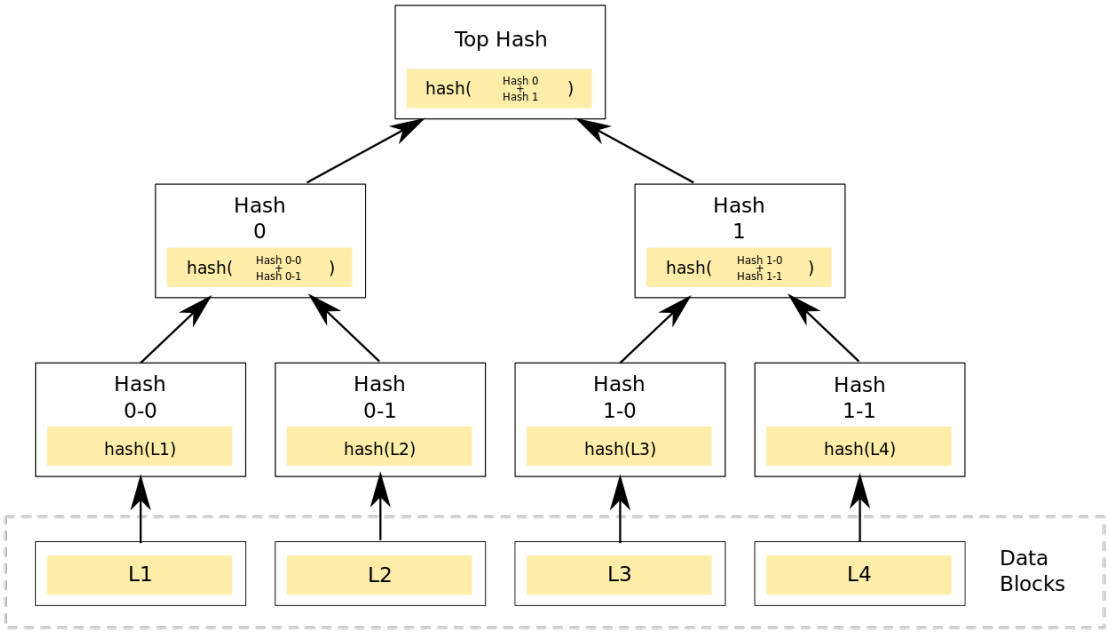

Figure 2.3: *Showing a Merkle tree with data chunks at the bottom and the combined hash at the top. Taken from [22].*

had been manipulated. If there is a mismatch between the block hash stored on the blockchain and the hash that is produced after the fact, it would mean that the block has been altered in some way. But without any backups, there would be no way of knowing exactly what had changed. Since the Merkle tree gradually creates the final hash, it would be possible to backtrack and find out exactly which of the data entries had been changed, while also reinstating the validity of the other data entries in the block.

2.2 Cryptocurrencies

The launch of Bitcoin in 2009 sparked a wave of innovation that would eventually lead to exponential growth in both the value and the number of available cryptocurrencies. Since then, the primary use of blockchains has been in building what could be referred to as *decentralised blockchains*. These are blockchains that are hosted using a distributed network of voluntary participants. These networks use consensus in order to maintain the blockchain. They are designed to be highly robust, not dependent on any one participant, and resistant to network manipulation. Almost every cryptocurrency currently has one of these decentralised blockchains at its core. This section will provide a general background on cryptocurrencies with a focus on these decentralised blockchains.

2.2.1 History

Bitcoin is today the most well-known cryptocurrency, but it was not the first. There were in fact several cryptocurrencies envisioned and launched in the years prior to Bitcoin. Some of these were fairly similar to Bitcoin and did, most probably, serve as an inspiration during Bitcoin's development.

Depending on the definition, eCash, envisioned in 1982 and launched in 1994, could be the first electronically transferable currency [25]. In 1982 Chaum published *"Blind signatures for untraceable payments"*, in which he envisions the payment system that would later be known as eCash [26]. In 1990 he started a company called DigiCash, which in turn released eCash in 1994 [27]. The eCash system facilitated the secure transfer of *CyberBucks* through public-private-key encryption [28]. Only a few banks decided to integrate with the system, and the company went bankrupt in 1998.

Another precursor to Bitcoin is E-gold. In 1996 Douglas Jackson and Barry Downey created a system in which parties were able to electronically purchase stakes in gold [27]. The system, dubbed E-gold, made it possible to send these stakes to other users, inadvertently creating a popular system for encrypted digital payments [29]. E-gold was arguably the first successful cryptocurrency with 3.5 million registered accounts as of 2005 [30].

In 1997 Adam Back created the Hashcash *proof-of-work* algorithm, which was designed to combat DoS attacks [31]. It was originally designed for use in email services but is today best known for its use in Bitcoin. The idea was that for every time someone wanted to interact with someone else, for example, by sending an email, they would first have to generate a token through so-called proof-of-work (PoW) [32]. PoW requires that the sender invests a certain amount of computing power in order to acquire a token. That token can then be used to interact with another user. This served as a disincentive for any malicious actor trying to overload a server with requests, since it would require a huge computing effort to generate enough tokens to overload the server. Hashcash was not the first PoW algorithm, but it was arguably the most influential. It was first used in mailing applications, but would later go on to be used in several of the earliest cryptocurrencies, including Bitcoin [31]. PoW will be discussed further in section 2.2.3.

In 1998 Nick Szabo proposed the cryptocurrency Bit Gold [33]. Bit Gold remains a concept as it was never implemented. Despite this, it is still one of the most influential cryptocurrencies as it heavily influenced Bitcoin's design. It was designed to use many of the technologies that would later be implemented in Bitcoin, such as PoW and sequential hashing [33], [34]. Thus, Bit Gold is, according to many, one of, if not the main precursor to Bitcoin [35].

In 2004 Hal Finney published the system Reusable Proof of Work (RPOW) [36]. This was a fairly simple system that implemented some of the technologies proposed by Bit Gold. It opted out of using byzantine fault tolerance, instead relying on a centralised server hosted by Finney himself. Users were able to create coins and

register their ownership over them through *public-private key pairs*. Public-private key pairs will be explained in section 2.2.6. Transactions were done in a similar way by sending requests to the server. The system was only really a prototype that demonstrated the feasibility of some of the technologies that would later be used in the creation of Bitcoin.

In 2008 an unknown group or individual published the Bitcoin whitepaper under the pseudonym Satoshi Nakamoto [6]. One year later Satoshi published the Bitcoin open-source code and mined the so-called *genesis block*, marking the beginning of the Bitcoin peer-to-peer network [37]. Bitcoin borrows a lot from earlier designs and Bit Gold in particular. There were two major changes however that set it apart. The first was to use proof-of-work in order to achieve *byzentine fault tolerance* [38]. Byzantine fault tolerance is a state a system can reach when it is able to withstand a certain level of malicious nodes in its network. This will be discussed further in 2.2.3. The second was to add block confirmations, which made double spending of coins impossible [33]. Double spending is when someone tries to spend the same coin twice by broadcasting two otherwise equal transactions but with different receivers. Bit Gold did not have an obvious solution to this problem. Bitcoin solved this by requiring that every validator in the network confirm the validity of a block before confirming it. This allowed Bitcoin to become the first functioning and secure decentralised cryptocurrency.

The purpose of Bitcoin was to create an alternative to the traditional financial system, forgoing the use of a trusted third party, such as a bank, when conducting financial operations [39]. The main motivation for creating Bitcoin is said to have been the financial crisis that started in 2007. Bitcoin was a way of putting the power over money and monetary policy back into the hands of normal citizens.

Bitcoin was not an immediate success and, as can be seen in Figure 2.4, for the entire first year the cost of a bitcoin remained below one cent. During this period the price of bitcoins was very volatile, even more so than today. It kept rising though, and at the beginning of 2011, it reached parity with the US dollar. The price continued to rise and by the end of 2013, the price peaked above $1000. After this followed a few silent years where the price remained in the triple digits. It was not until the beginning of 2017 that it once again broke above $1000. By the end of that year, a bitcoin was valued at $20,000 and with that Bitcoin had broken into the mainstream. After this followed another silent period up until the recent highs during the COVID-19 pandemic.

In 2013 Vitalik Buterin published the Ethereum whitepaper introducing the world to a new form of cryptocurrency utilising smart contracts allowing turing-complete code to be uploaded to the blockchain [10], [41]. In July 2015 the Ethereum Mainnet was launched. This sparked a second wave of innovation that has resulted in decentralised banking services, play-to-earn games, and much more.

Bitcoin, together with Ethereum, have created a continuously growing ecosystem

Figure 2.4: *Showing a logarithmic version of the historical price of one bitcoin. Source: [40].*

of interconnected blockchains and blockchain services. After Ethereum, several promising cryptocurrencies have been launched that have the potential to upset the digital currency industry to a degree similar to that of Bitcoin and Ethereum.

2.2.2 Cryptocurrency ledgers

Cryptocurrencies use ledgers to record their transactions. The ledger contains a list where each entry records a transfer of value from one party to another [42]. Key to the ledger is that it does not have a separate list for standing balances like a traditional bank account might. Instead, the balance of each participant is intrinsically known through the list of transactions.

Cryptocurrencies store their ledgers as individual transactions inside the blocks on their blockchains [43]. Although some cryptocurrencies, such as Ethereum, have chosen to complement the ledger with separate account data to make calculations more effective, they still have a ledger in order to prove the validity of the blockchain [44].

Ledgers outside of cryptocurrencies will sometimes allow participants to have a standing debt inside the ledger. This way, a ledger can be put to use without

the participants having to add funds. This is not possible when working with cryptocurrencies. Since accounts cannot be connected to individuals, there is no way to prosecute a user who refuses to settle their debt. Because of this, a cryptocurrency user will have to purchase the native cryptocurrency in order to partake in *on-chain transactions*. Such a purchase is usually performed by exchanging *on-chain assets* with *off-chain assets* through a cryptocurrency exchange. A cryptocurrency exchange is an online marketplace where users can purchase or sell cryptocurrencies. Many such exchanges also allow their users to purchase cryptocurrencies with other cryptocurrencies.

The use of ledgers by cryptocurrencies is crucial to this thesis. It is the transactions stored on the Ethereum ledger that make it possible to retroactively track the flow of illicit funds between Ethereum accounts.

2.2.3 Cryptocurrency validators

Every cryptocurrency is maintained by a network of validators who work together to keep the cryptocurrency up and running [45]. Validators are volunteers who keep the network online by providing the computers needed to receive and compute transactions. In return, they usually receive a small amount of the cryptocurrency they are helping to maintain. If someone wants to make a transaction on the network, they first send the transaction to at least one of the validators, who in turn makes sure that the transaction can be added to the next block on the blockchain [46]. This assumes that the transaction follows the rules of the network.

Anyone with a computer and an internet connection can be a validator. The network is designed so that validators can join or leave at any time. This is one of the mechanisms which makes cryptocurrencies so secure. Since anyone can be a validator there are usually at least a few thousand validators active at any given moment [47], [48]. This makes it much harder to take down the network compared to if it was hosted on a single server.

Every cryptocurrency has some level of decentralisation, meaning there is no central authority that makes final decisions on the state of the blockchain [49]. Because of this, it is very important that the validators of a cryptocurrency use a robust consensus mechanism that can make decisions without the need for a central authority. Since anyone can join the network, this consensus mechanism also has to achieve Byzantine Fault Tolerance, which means it is able to operate even when some of the nodes in the network provide false information. There are almost as many consensus mechanisms as there are cryptocurrencies. What all of them have in common though is some form of majority vote, where all of the validators have a say in what is added to the blockchain.

The simplest solution would be to give every validator an equal vote and then let the majority decide what should be added to the blockchain. Though simple, this system has a major flaw. Since anyone can become a validator, there is the possibility for an

attacker to boot up thousands of validators in order to acquire a majority vote on the network. If an attacker does this, they would be able to introduce transactions that violate the rules in their favour. They would be able to steal basically anything stored on the blockchain. This would have devastating consequences for both the network users and the trustworthiness of the cryptocurrency itself. Because of this, there has to be some protection in place that makes it harder to acquire a majority vote. There are almost as many solutions to this problem as there are cryptocurrencies, but they can all be roughly grouped into two major groups, *proof-of-work* (PoW) and *proof-of-stake* (PoS).

Proof-of-work makes it so that every validator has to spend a certain amount of computing power in order to be allowed to participate in the network [50]. The PoW validators, commonly called miners, use an algorithm, called a mining algorithm, to generate hashes of the block they are trying to validate. The point is to find a hash of the block that starts with a certain number of zeroes. In order to find this hash they alter the signature of the block by combining it with a number, called a *nonce*. After combining the nonce with the block, the miner passes it through a hashing algorithm to generate a result. If the result does not start with a sufficient number of zeros, the nonce is changed, and the process is repeated until an appropriate nonce is found.

All the miners on the network work together to find a satisfactory nonce. When someone finds a proper nonce, they publish it along with the block to the rest of the network. If everything has been conducted according to the rules, the other miners stop mining the current block and move on to mining the next block in line. There will be no point in saving the results from the previous block since the combination of the new block and any nonce used with the previous block will now be completely different.

The result of a hashing algorithm changes completely every time a change is made to the input, even if the change only involves a single character [51]. Because of this, there is no way to triangulate a nonce that will have the desired result, which means that the only way to find a matching nonce is to guess at random. This is important since it gives the nodes with slower computers a fair chance to find the nonce. If triangulation was possible, a faster computer would almost always find the nonce before a slower computer. Because of this system, a miner's chance of finding an appropriate nonce is, in fact, directly proportional to the amount of computing power they have available. This in turn means that the miner's voting power will also, over time, be directly proportional to their share of the network's total computing power. This is important since it allows miners with less powerful computers to be competitive in the race to find the nonce. Since smaller miners can participate it results in a larger number of validators overall which in turn makes the network as a whole harder to take down.

This linear relationship between computing power and voting power also makes it harder for an attacker to take control of the network. It does not matter how many

validators an attacker controls, as long as they do not collectively control more than half of the computing power of the network. Since computing power is a limited resource that is expensive to obtain it would be very difficult for an attacker to perform an attack that resulted in a financial net gain.

Proof-of-stake, the alternative to PoW, takes a fairly different approach to achieve secure consensus. Instead of relying on the scarcity of computing power, PoS relies on the scarcity of the cryptocurrency it is implemented on. On a PoS cryptocurrency, the voting power of a validator is directly proportional to the amount of native cryptocurrency the validator is holding [52]. The validator can do something referred to as *staking* their cryptocurrency. Staking usually involves sending the coins to a separate address where they will be locked for a certain amount of time. The staked coins will then allow the validator to participate in the validation of the next block. The creator of the block is randomly assigned, where the chance of winning is directly proportional to the portion of native cryptocurrency the validator has staked.

Assuming that most of the native cryptocurrency is used for staking, an attacker would have to purchase close to half of the native cryptocurrency in order to take control of the network [52]. Apart from being incredibly expensive, such an attack would also, most probably, cause a drastic fall in the price of the cryptocurrency. Since the attacker would be holding close to half of the total cryptocurrency, such a fall would hit the attacker harder than anyone else. This, combined with the large initial cost, acts as a strong disincentive against attacks trying to take control of a PoS cryptocurrency.

2.2.4 Block rewards

For validators to want to participate in the network, there has to exist some kind of incentive for them to do so. This incentive usually comes in the form of the so-called *block reward*. This is an extra transaction added to the beginning of every block that is addressed to the validator who created the block [53]. This transaction usually consists of two parts, transaction fees and the *block subsidy*.

Every user who creates a transaction has the option to add a fee [54]. This fee will be transferred to the validator who manages to create the next block in the blockchain. If there are more pending transactions than can fit inside the block, the fee will serve as an incentive for the block creator to add the transaction to the block. Since higher fees mean higher profits, validators almost always choose to add the transactions with the highest fees. Supply and demand thus dictate that the size of the fees will fluctuate up and down over time, depending on the number of pending transactions and users' willingness to pay.

The second part of the block reward is the block subsidy. This is a predetermined amount that is added to every block in order to encourage validators to participate in the upholding of the network [55]. The subsidy consists of newly minted cryptocurrencies, meaning that they are essentially created from nowhere in order

to reward the validator. Some cryptocurrencies have all of their native currency originating from this block subsidy. Bitcoin is one of these cryptocurrencies, while Ethereum had a fairly large fundraiser at the beginning where coins were created outside of the block subsidy [56], [57]. A cryptocurrency that has all of its native currency originating from the block subsidy is said to have had a fair launch [58]. Such cryptocurrencies are often held in higher regard than others who had large parts of their supply go to founding members or select investors. Cryptocurrency projects that held an open *initial coin offering* could also be considered a fair launch [59]. This is when the initial sale of coins is not limited to a smaller group of people but is instead sold openly for anyone to buy. Ethereum had most of its initial coins originating from such an open sale [57].

2.2.5 Fungibility and divisibility

A trait most cryptocurrencies share with traditional *fiat currencies*, such as dollars and euros, is their *fungibility*. This means that they are entirely interchangeable with each other [60]. Two coins cannot be told apart, and are as far as the protocol considers the same. This even goes as far as not having any form of identification that can distinguish one coin from another. This means that if someone receives crypto from two different sources there will be no way of telling the two sums apart after they have entered the account. This property is central to the issue that is being addressed in this thesis. Since there is no way of telling different coins apart it significantly increases the complexity of tracking the flow of funds between different addresses.

Cryptocurrency coins are also usually highly divisible, meaning that a single bitcoin or *ether* (the native currency on Ethereum) is not the smallest unit of exchange on their respective blockchains. A bitcoin is, for example, divisible by 10^9 while an ether is divisible by 10^{18} [61], [62]. These smallest possible units tend to have their own names. The smallest possible portion of a bitcoin is called a *satoshi*, while the smallest portion of an ether is called a *wei*. This high degree of divisibility is another reason why it is so hard to track the flow of funds on cryptocurrencies. Since it is possible to transact such small amounts it is also possible to spread out illicit funds among a large number of addresses where they can combine with untracked funds.

2.2.6 Public-key cryptography

Cryptocurrencies use what is known as *asymmetric cryptography* in order to prove ownership of accounts. This is a form of encryption that utilises the asymmetric complexity of certain mathematical functions in order to enable verifiable encryption of messages without the need for a secure communication channel [63]. To achieve this, *public-private key pairs* are utilised, where each user has a *public key* and a *private key*. The public key is shared openly with other users, while the private key is kept secret. The keys are cryptographically linked so that when the public key is used to encrypt messages, only the private key can be used to decrypt those messages.

Before public-key encryption, data could only be encrypted and decrypted with the same key. This meant that if a sender and a receiver of a message wanted to communicate securely, they would have to share the same *secret key*. This was a problem, as it meant that correspondents needed a secure way of agreeing on a key. It would be risky to send the key openly over the internet. The entire point of having a key is to encrypt messages so that no one can listen in on the conversation. If the key is sent in plain text through those same channels, anyone trying to listen in would have access to the key, which would defeat the purpose. This problem can be addressed by public-key encryption. Having a separate key for encryption and decryption makes it much easier for two users to establish a secure connection. Both can send their public keys to each other and then use their private keys to decrypt any incoming message. It does not matter if anyone else intercepts the public key since it can only be used to encrypt, not decrypt.

Public-private key pairs also have another key feature that is utilised heavily by cryptocurrencies. Usually, the public key is used for encryption and the private key is used for decryption, but they can also be used in reverse. If a message is encrypted with the private key, it can only be decrypted with the public key. This makes it possible to verify the sender of a message. If a message can be decrypted with the public key it means it has to be sent by the holder of the private key. This feature is crucial to prove the possession of accounts on cryptocurrencies. When a validator can decrypt a transaction with the public key of an address it proves that the true owner of the private key wants to perform said transaction.

2.3 Ethereum

As the main focus of this thesis, Ethereum needs some further technical introduction in order to give sufficient understanding going forward. The following sections will explain the most fundamental concepts necessary to understand Ethereum on a technical level.

2.3.1 Accounts

Ethereum utilises an account structure in order to make it easier to keep track of information [64]. Using these accounts, users on Ethereum can send and receive Ethereum native currency ether, as well as interact with the many other features Ethereum has to offer. Like most other cryptocurrencies, Ethereum uses public-private key pairs to secure its blockchain [65]. Every Ethereum account has an associated private and public key that is used to send and receive funds. An account is represented by its address, which on Ethereum is a 20-byte hash derived from the last 20 bytes of the public key. There are two kinds of Ethereum accounts, *externally owned accounts* (EOA), and *smart contracts accounts* (SCA) [64]. EOAs are ordinary accounts used by normal users to send and receive funds, while SCAs are special accounts associated with smart contracts, which will be discussed in more detail later. In addition to the associated public and private keys, each Ethereum account also has four properties: balance, nonce, code, and storage. Balance and nonce are

utilised by both types of Ethereum accounts, while code and storage are only used by SCAs.

The balance of an account is what keeps track of how much ether the account currently holds. Every new account starts out with a balance of zero and will only increase after receiving ether from another address. Though there is no upper limit to how much ether an account can hold, it will not be allowed to go below zero.

The nonce is an integer stored on the account that will be incremented every time a new transaction is sent from the account [46]. This is not to be confused with the nonce that is used in PoW consensus. The main purpose of the nonce is to prevent *double-spendings* of transactions. If someone were to send two transactions with the same amount to the same receiver, the contents of the transaction would be the same. Every new transaction on Ethereum has to be signed with the private key of the sender. Since the signature is simply a function of the contents of the transaction and the private key, it would mean that two identical transactions would also have the same signature. If this is the case, it means that the private key is not necessary when creating a new transaction, as long as the transaction matches a transaction that has already been performed. Anyone would be able to rebroadcast already created transactions, making them count twice. If Alice sends an ether to Bob, Bob will be able to repeat that transaction endlessly until Alice's account is drained. This is where the nonce comes in. The nonce, which is a counter of how many transactions an account has made, will be included in and sent along with every new transaction. Since the nonce is incremented for every new transaction, it ensures that two transactions from the same account will never have the same nonce. Since the nonce of two transactions can never be the same, it also means that the contents of the two transactions can never be the same, which in turn means that the signatures of two otherwise entirely equal transactions can never be the same. Since two transactions cannot have the same signature, it also means that every signature will have to be signed individually by the holder of the private key, which finally means that only the holder of the private key can make transactions from its associated account.

2.3.2 Smart contracts

Perhaps the most distinguishing feature of Ethereum is its use of smart contracts. In short, smart contracts can be thought of as a way of making it possible to run computer code directly on the blockchain [66]. Smart contracts were introduced in conjunction with Ethereum in 2013 through the publication of the Ethereum white paper [10]. Each smart contract behaves like its own self-contained program. It can take inputs, return values, and store information for later use. Every smart contract has its own associated account, an SCA, which it uses to send and receive funds.

In contrast to EOAs, SCAs actually use the code and storage properties of their accounts. The code property is used to store the compiled code that will be used to execute the smart contract. This code is usually written in *Solidity*, a coding

language created specifically for Ethereum, but can also be written in other languages, such as *Viper* [66]. The language does not matter as long as it can be compiled down to Ethereum native byte code, which is what is actually stored on the blockchain [67]. The storage property is used to store the data that will persist between different interactions with the contract. The code property is permanent after the smart contract has been created, whereas the storage property can be changed at any time by the smart contract during its operation.

Users can interact with smart contracts by sending them transactions. When this is done, the smart contract will act on the transaction according to its code. Solidity and the other Ethereum-compatible languages are all Turing complete, meaning smart contracts can perform all of the operations of a regular computer.

2.3.3 Ethereum Virtual Machine

In addition to the blockchain, Ethereum also has another data structure known as the *Ethereum virtual machine* (EVM). This is a state machine that encompasses the state of everything on the Ethereum blockchain [68]. The EVM is the decentralised computer that performs the computations necessary to execute calls to smart contracts. The state of the EVM is equal to the data stored in every EOA and SCA account, along with the accounts themselves. The state is updated every time a new block is produced and will change depending on the interactions between Ethereum accounts.

Every datapoint in the EVM is stored in a *Merkle Patricia trie* which is a modified and more advanced version of a Merkel tree [69]. The Merkle Patricia trie works similarly to the Merkle tree in that it produces a hash that results from every data point in the tree. This hash is then used in the production of the block hash, which makes the state of the EVM tightly bound to the sequential hashing performed in the blockchain. This makes the state of the EVM, very close to immutable, for the same reasons as was discussed in section 2.1.3.

Every Ethereum validator has their own version of the EVM that they keep up to date by computing the instructions created by Ethereum's users [70]. They validate their own version of the EVM by comparing the resulting Merkle Patricia trie hash to that of other validators. By computing the EVM state they can confirm that the incoming transactions are legal and that blocks produced by other validators can be trusted.

2.3.4 Transactions and messages

There are two types of communication between Ethereum accounts. The first is a normal transaction, which is when a normal user, through their EOA, interacts with either another EOA or a SCA [71]. The second form of communication is simply called a message. This is when a smart contract through their SCA interacts with another SCA or an EOA. The main difference between these two types of interaction

is that only transactions are recorded on the blockchain. Messages are almost never recorded, which means they only exist for a brief moment during the execution of the smart contract that sent the message. This is important since messages can also act as transactions, transferring value between addresses. This means that it is not enough to use the transactions recorded on the blockchain when trying to track illicit assets on Ethereum.

Messages are on rare occasions recorded by what is known as Archival Nodes [72]. These are special Ethereum validators who save every state of the EVM. In addition to this, they can also record every message on the blockchain. Messages can also be generated by running the EVM from the beginning and processing every transaction in order while recording the messages they produce. When this is done the node is instead referred to as a tracing node.

2.3.5 Tokens

Apart from Ethereum's native currency ether, it also has support for the creation of alternative currencies known as tokens. These tokens can take many shapes and represent almost anything, such as tickets to a ride, shares in a company, the US dollar, points in a game, or an image of a monkey. Users can create their own tokens by deploying smart contracts. The code of the contract is what defines the behaviour of the token, while the contract's storage, among other things, keeps track of which EOA owns which token.

In order to simplify the process of interacting with different tokens, standards have been created by the Ethereum community. These standards are a set of rules that define a number of functions that a token smart contract should implement. There are two main standards for tokens, *ERC20* and *ERC721*. ERC20 is a standard for fungible tokens which, as described in section 2.2.5, means that they are interchangeable. Any token created by the same contract will be considered equal to any other. An ERC20 token contract must implement the functions in the ERC20 token standard [73].

ERC721 is a standard for *non-fungible tokens* (NFTs). A non-fungible token is unique, meaning that different tokens from the same contract have different properties. This could be tickets with seat numbers or images of apes with unique hairstyles and clothing. An ERC721 token contract must implement the functions in the ERC721 token standard [74].

2.4 Cryptocurrency laws and regulations

In order to understand the importance of detecting and tracking illicit cryptocurrency assets, one must first understand what the chance is of acquiring illicit goods and what the consequences might be. There is an underlying legal framework that defines the rules of conduct for both private individuals and financial institutions. In this section, background on current and past cryptocurrency legislation will be provided along

with other general information on how financial actors perform *anti-money laundering* (AML). Financial regulators generally place Bitcoin and other cryptocurrencies under the umbrella term of *virtual assets* [75]. In this section, cryptocurrencies will thus be referred to as such.

2.4.1 Why virtual assets are used for illicit activities

Virtual assets have been used for illicit activities since before Bitcoin was launched. E-gold, one of the precursors to Bitcoin, was, for example, used to store and transfer proceeds acquired by selling illegally obtained goods [76]. Bitcoin has also had its fair share of illicit activity [1]. The initial lack of regulation and high level of anonymity made virtual assets an attractive choice compared to transacting via regular financial institutions. Since virtual assets are entirely disconnected from the traditional financial system and are usually not tied to any one organisation or nation, they are not subject to the same rules and regulations as banks or credit card providers. In a sense, Bitcoin and other virtual assets act more like cash, enabling criminals to perform transactions directly with each other, but without requiring any direct physical contact. In simple terms, Virtual assets provide a more reliable and safe way for these criminals to do business.

Many initially falsely assumed that Bitcoin transactions were completely private allowing them to transact without the fear of being tracked [77]. Even though it was possible to look directly at blockchain data and track individual transactions, this was an arduous task since it had to be done manually. In practice, this made Bitcoin transactions private, as long as the actors involved in the transactions were somewhat inconspicuous. Since then blockchain analytics companies, such as Chainalysis and Elliptic, have created services making it easier to track the movements of funds on blockchains. Despite this, illicit activity continues.

Due to the semi-anonymous nature of Bitcoin, it is hard to calculate a real figure for how large of a percentage of all transactions that are involved in illicit activity. Igor et al. estimated that in 2015 about 4% of bitcoin transactions could be directly connected to activities involving illegal transactions, scams, and gambling [1]. In 2021 this figure had dropped to 0.4%. Although this may seem like a small amount, it is important to note that the amount of money involved is still very large. Igor et al. estimated that in 2020 alone, all illicit Bitcoin transactions had a volume of at least $5.3 billion. The total value of all illicit trade involving Bitcoin has continued to rise even though its portion of the market has fallen. Some of the illicit activity has also, most likely, spread to other virtual assets, such as Monero [78].

2.4.2 Virtual asset service providers

Financial institutions involved with virtual assets can generally be divided into two groups. First, there are the *Virtual asset service providers* (VASPs), who are directly in contact with virtual assets. These are institutions providing services to purchase or exchange virtual assets, such as cryptocurrency exchanges, crypto ATMs, and

custodial services such as crypto asset managers [79] [80]. The second group is those who do not have direct contact with virtual assets but do on a regular basis interact with VASPs. This category can contain a large variety of financial institutions, but the majority of them will most probably be banks of varying sizes.

VASPs are the main target for virtual asset regulation since shady actors go to them for storing, transporting, and cashing out illicit funds. VASPs along with private virtual asset traders are the main targets of this thesis since they will be the ones making choices on which virtual assets to accept and which to avoid. Other institutions will largely have to trust VASPs when making transactions from and to their platforms unless some form of transparency agreement has been signed. It would be possible for an outsider to scan the transactions going from and to a VASP's virtual asset address. However, it would be hard to find which outgoing fiat transaction corresponds to which incoming virtual asset transaction without having access to their internal systems. It is likely that an illicit actor would choose to transact a common, even amount of the virtual asset in order to increase the chance that several similar transactions are made at the same time. It would also most likely be possible to leave the fiat waiting in the VASP's systems for a long period of time, making it even harder to make accurate correlations between incoming and outgoing transactions.

2.4.3 Current AML practises

Currently, financial institutions conduct ongoing financial crime intelligence collection and assess how current and emerging risks will affect their own customers and products [81]. To map the identified risks to specific customers, financial institutions regularly ask their customers questions where the answers are used to rate the risk of each individual customer, and if needed specific transactions. This *customer due diligence* process of collecting *know-your-customer* (KYC) information is done on an ongoing basis. The individual risk rating, in turn, is used for risk-based monitoring of the customer and the customer's transactions. A high-risk customer will probably have to answer KYC questions more often, and their transactions will be monitored more closely. See [82], for potential monitoring scenarios specific to virtual assets. Additionally, customer behaviour, the use of riskier products, and geography such as non-domestic transactions to and from high-risk countries, may raise the risk level further. Suspicious activities are investigated and, if necessary, reported to the responsible authorities.

For sanction compliance, at a high level, financial institutions pay for an updated feed of information about parties who have been put on sanction lists [81]. Transactions belonging to sanctioned parties must then be stopped in real-time. A recent example of new sanctions is the blocking of specific Russian transactions since the start of the war in Ukraine.

2.4.4 Current virtual asset regulations

Normal financial services are subject to a range of regulations that, at least initially, were not applied to virtual assets. For a traditional financial institution, the main regulatory concerns are to mitigate illegal transaction activities such as fraud, money laundering, financing of terrorism, and sanction avoidance [81]. With time it is thought that these will to an increasing extent be applied to virtual assets.

Virtual asset legislation is still maturing and tends to vary a lot from region to region. However, this has started to change as global actors move to standardise the legislation. The single entity that has the largest influence on global monetary policy related to money laundering efforts is the *Financial Action Task Force* (FATF). The FATF is tasked with setting international standards and recommendations for AML legislation [83]. In practice, the recommendation provided by the FATF often serves as the basis for both national and international AML regulations.

2.4.5 The travel rule

One of, if not the most discussed regulation related to virtual assets is the so-called *travel rule* introduced by the FATF in 2020 [84]. The idea of the travel rule is for financial actors to pass on additional information whenever a transaction is performed between financial institutions. The purpose is to create an information trail enabling law enforcement to detect and track the flow of illicit funds. The travel rule was first introduced by the *Financial Crimes Enforcement Network* (FinCEN), a bureau within the American Treasury Department, through the Bank Secrecy Act in 1997 [85]. This legislation was eventually adopted by the FATF for international AML policy in 2012 [80]. Initially, the rule was only applied to financial institutions, but in 2020 VASPs were also required to comply with the travel rules [84].

The travel rule is another reason why tracking the flow of virtual assets is so important. A decentralised entity cannot be held accountable for not providing an information trail on transactions. It will thus be up to the actors partaking in virtual asset systems to track the flow of funds.

2.4.6 The future of virtual asset compliance

It is likely that organisations such as the FATF along with governments will continue to increase the number of regulations that apply to virtual assets and that this will create a growing demand for virtual asset tracking tools.

Most financial institutions as of 2023, do not offer crypto services by themselves. However, it can be envisioned that many financial institutions will provide crypto services for their customers in the future, especially as new regulations will decrease the number of surrounding uncertainties and risks. This will further increase the need for proper tracking tools, and more specifically for tracking tools that can be used by parties not as experienced in working with virtual assets.

3

Literature Study

At the start of the thesis we performed a thorough literature study in order to get a sufficient understanding of the field and to guide the design of the AML tool. The most promising findings of that study are summarised in this chapter. Four main strategies for performing AML on cryptocurrencies were identified. These were clustering, blacklisting, machine learning and automatic Ponzi scheme detection. Below follows one section for each of the methods summarising one or more papers on the subject. Since only clustering and blacklisting ended up in the final tool it will be sufficient to only read those sections before moving on. The other sections are only necessary to better understand the context of the thesis.

3.1 Clustering from known illicit addresses

There is nothing stopping an actor from having control over more than one address on the blockchain. If an illicit actor has one address connected to a scamming smart contract, it could be very valuable to find out what other addresses the illicit actor is also controlling. These other addresses might reveal other scam contracts, or where stolen assets are kept. For this purpose, there exist clustering techniques designed to clump together addresses likely to be controlled by the same actor.

In 2016 Bernhard et al. presented GraphSense [86], a tool designed to cluster addresses on Bitcoin. The tool uses simple clustering heuristics to group addresses based on their transactional history. If two addresses are used as inputs in the same transaction, they are marked as belonging to the same cluster. Similarly, if a *change address* is created in a transaction, it is assumed to belong to the same cluster as the addresses used as input in the transaction. A change address is where all of the extra funds end up at the end of a Bitcoin transaction. Their system allows for identifying tags to be added to known Bitcoin addresses, for example, an exchange, a dark marketplace, or a cryptocurrency mixer. A cryptocurrency mixer is a contract-based service that obfuscates the origins of coins by pooling and redistributing them to enhance user privacy. Depending on what clusters a tagged address ends up in, conclusions can be drawn about the purpose of the entire cluster. For example, if a dark marketplace address ends up inside a cluster, all the other addresses inside that cluster could be marked as having a higher risk of containing illicit funds. Using

their tool, the team was able to identify roughly 2000 *super-clusters* containing 16% of all Bitcoin addresses and being responsible for 23% of all Bitcoin transactions. This they argue makes it easier for them to track assets moving on the blockchain since it reduces the number of nodes significantly.

In 2020 Victor Friedhelm set out to create similar clustering heuristics designed for Ethereum [87]. Bitcoin uses a fairly unique system for managing transactions called the *UTXO model*. In this system, each address has to spend all of its content every time a transaction is performed. A receiver is given the funds they need, while the remaining funds are transferred to a change address. Since Ethereum uses the more traditional account-based transaction model instead of the UTXO model, the heuristics established for Bitcoin would not be of any help. Thus, new heuristics had to be developed in order to enable address clustering on Ethereum.

His first heuristic is *Deposit address reuse* (DAR), which takes advantage of a common practice among exchanges. This is to create a so-called *deposit addresses* for each user. When a user wants to deposit funds to the exchange, they have to send their funds to this deposit address for them to show up in their account. This means that you can assume that all addresses sending funds to a specific deposit address belong to the same user. Standard practice for most exchanges is then to almost immediately forward the received funds from the deposit address to one of the exchange's vault addresses. This process of forwarding funds from deposit addresses to vault addresses is a pattern that can be recognised by scanning the blockchain. When this pattern has been found, the deposit addresses can be picked out, after which one cluster can be created for every deposit address.

The second heuristic is called *airdrop multi-participation* (AMP) and takes advantage of the accumulation patterns created by users receiving more than one deposit from a single airdrop. An airdrop is a common way of jumpstarting a cryptocurrency project. In order to get people excited about a new token on Ethereum and improve the distribution of the tokens, the project team might give out tokens for free. In some airdrops, the number of deposits is limited to one per address. When this is done, it is common for people to create and register multiple addresses in order to receive multiple deposits. After receiving the airdrop these users usually want to accumulate their airdropped tokens into a single address. This accumulation is what AMP exploits. If multiple addresses immediately pass on their entire token deposit to a single accumulation address it can be assumed that all of these addresses, including the accumulation address, belong to the same entity.

The third and final heuristic is called *self-authorisation* (SA) and takes advantage of the authorisation functionality that Ethereum offers. This functionality enables one address to allow another address to spend its assets. This functionality is for the most part used by smart contracts but is occasionally used between EOA addresses. The author believes that it is reasonable to assume that EOA addresses that allocate to each other belong to the same user. Two safeguards are added in order to filter out the rare instances where this is not the case. First, a maximum of 10 addresses is

allowed to allocate to the same address. Anything over that and all of the addresses are assumed to belong to different entities. Secondly, a single address is allowed to allocate to a maximum of 10 unique addresses. Anything over that and all of the addresses are again assumed to belong to different entities.

According to the authors, DAR was found to cluster by far the most addresses with a grand total of roughly 9.5 million addresses. AMP and SA also managed to cluster, though in much smaller numbers.

Later in 2020, Ferenc et al. proposed a more general approach to address clustering [88]. They argued that the heuristics developed by Friedhelm are powerful but limited since they require participation in voluntary on-chain events. They instead proposed a more general method of clustering that utilise *quasi-identifiers* to reach every address on the blockchain. These quasi-identifiers are softer values such as time of day activity and the transaction history of an address. Since every address on Ethereum has this data it meant that they were able to cluster for every address on the blockchain. They were able to create a list for every address containing other addresses that are thought to be closely related. They tested different *node embedding algorithms* on different quasi-identifiers and came to the conclusion that a combination of the *Diff2Vec* and *Role2Vec* generates the best results. Node embedding algorithms are methods for representing nodes in a graph in n-dimensional space. This is useful when trying to train machine learning algorithms on graph data. Diff2Vec and Role2Vec are both node embedding algorithms that try to improve on the otherwise popular *random walk* embedding method [89], [90].

In 2022 Mike et al. presented Tutela [91], [92], a privacy-preserving tool implementing the findings of Friedhelm and Ferenc et al. This tool has two purposes, the first is to detect whether an Ethereum address can be linked to other addresses, and the second is to determine how secure the Tornado Cash mixer is by trying to connect deposits with withdrawals on the service. The authors implemented their own version of the DAR heuristic developed by Friedhelm. Since this does not cover the whole transaction graph they have also implemented their own version of what they call the NODE algorithm which is a node embedding algorithm utilising Diff2Vec.

3.2 Blacklisting

Another method that has been used within traditional finance for a long time is blacklisting. Blacklisting involves creating a list of financial entities that are forbidden from interacting with other financial entities inside a given jurisdiction. The financial entities can be individuals, companies, or entire countries, while the jurisdiction is usually a country or larger region such as the EU. Within cryptocurrency, blacklists are believed to have been used as early as 2012, when the infamous now defunct exchange Mt.Gox was restricting access to accounts thought to have been involved in known cryptocurrency thefts [93], [94].

The earliest mention of cryptocurrency blacklisting in literature was done in 2013

when Möser *et al.* published a paper where they, among other things, purpose blacklisting as a way to perform AML on Bitcoin [95]. Here the authors argue that blacklisting in combination with other AML measures could be an effective way of fighting money laundering on Bitcoin. In traditional financial services, blacklists are most often used to mark individuals who are not allowed to partake in the financial system. As there are no identification requirements to interact with the Bitcoin network the authors argue that the bitcoins involved in illicit activity should be blacklisted instead of blacklisting the individuals in possession of those same coins. Since all transactions on the Bitcoin network are public, it should be possible to identify transactions related to specific illicit activities, after which they can be added to a blacklist. This blacklist could then be used to block the coins from entering exchanges and other places where they could be exchanged for fiat currencies. The blacklist could also be made public, discouraging private investors from purchasing blacklisted coins. This would gradually degrade the value of blacklisted coins until they are practically worthless, which in turn would discourage people from partaking in illicit activities in the first place.

Möser *et al.* published a new paper in 2014 specifically dedicated to discussing blacklisting on Bitcoin [96]. Here they introduce several new and important concepts that have to be taken into consideration when creating blacklists for cryptocurrencies. One of these concepts is the time delay between when an illicit transaction is performed and when the coins involved in the transaction are added to the blacklist. During this period illicit coins could be sold to innocent users who would be stuck with useless coins when the blacklist is finally updated. Because of this, the authors argue for the importance of risk analysis when acquiring new assets on a blockchain. Every user should take a range of factors into consideration when deciding whether to purchase any specific coin. One such factor is how long it has been since the last time the coins were moved. If coins have recently been moved, the authors argue that they have a higher chance of being involved in illicit activity.

In the paper, the authors also expand upon the concept of *taint* that was briefly mentioned in the paper from 2013. They define taint as the portion of a cryptocurrency address or transaction that is made up of illicit coins. Coins that have been involved in illicit activity are in this context referred to as *tainted coins*. This ties into another new concept they discuss which is how taint should be transferred when transactions with blacklisted coins are performed. As it is possible to have more than one input address in a single Bitcoin transaction, it is inevitable that clean coins will be mixed in with tainted ones. If one transaction contains a tainted input A, a clean input B, and a combined output C a decision has to be made on what level of taint should be assigned to C.

The first and simplest strategy they proposed is the *poison* strategy. Here, all transactions involving tainted coins produce purely tainted outputs. In the above example, C would become 100% tainted. This strategy would be simple to implement and would create strong incentives for users to not accept tainted coins. The main issue with this strategy is the time delay between when an illicit activity occurs and

when it is added to the blacklist. If the culprit manages to sell the illicit funds to several other users before the funds are added to the blacklist a much larger number of bitcoins might end up being blacklisted. If a transaction is added to the blacklist after a long period of time it could possibly infect a significant portion of the entire Bitcoin supply. To remedy this, the authors suggest adding a maximum time after which transactions cannot be added to the blacklist.

Apart from the poison strategy the authors also suggest two other strategies, *haircut* and *first-in-first-out* (FIFO). The haircut strategy assigns a new taint value to an output based on the percentage of tainted coins used as input. So if 20% of the input consists of tainted coins the output would have a taint value of 20%. This way the total amount of taint on the blockchain will remain constant no matter how many times it changes hands. The authors also suggest that the value of a tainted output can be adjusted down to be proportional to the untainted portion of the output. Therefore, if the level of taint of a transaction is 50%, the output should only sell for half the price.

The FIFO strategy, similarly to haircut, ensures that the taint on the blockchain remains constant. The main difference from haircut is that it tries to preserve the concentration of the taint. It takes advantage of how Bitcoin transactions work to keep tainted and clean coins separate. Since Bitcoin allows for more than one output address per transaction, it is conceivable to have different levels of taint for the different outputs. In FIFO the taint is only given to the coins that have the same position in the output as the tainted coins have in the input. Imagine a transaction where only the first of two inputs are tainted. If the receiver wanted to keep the tainted and clean coins separated, they could add an extra output address first in the transaction. If this transaction had the same size as the input with tainted coins, all of the tainted coins would be transferred to this separate address, thus preserving the purity of the coins.

In 2017 Abramova *et al.* continued where Möser *et al.* left off and applied game theory mechanisms to cryptocurrency mixers [97]. They argue that due to the public nature of most cryptocurrencies the coins are not truly fungible. Since each coin has a unique transaction history, it can be uniquely identified and evaluated based on that history. They also argue that this transaction history will increase in importance as blacklists become more prevalent within the cryptocurrency space.

In their study, they outline two theoretic scenarios for how people using cryptocurrency mixers will behave based on the transaction history of their and other participants' coins. The first scenario is that of perfect information, where all the participants always know the taint of every coin entering or leaving the mixer. Here all participants will likely want to partake in mixing as long as they do not increase the taint level of their holdings. There is no risk of participating since the taint level of the coins you receive is known before you commit to the transaction. Some participants might even accept a slight increase in taint if they deem the gain in privacy worth the price.

The second scenario is that of imperfect information, where participants cannot be sure of the taint level of the coins in the mixer. This sort of scenario is more realistic according to the authors. This is due to the window of time between when an illicit transaction has occurred and when it is added to the blacklist. During this time window, Illicit actors will have strong incentives to mix their coins before they are added to the blacklist. Due to this possibility, actors who have lower levels of taint will be hesitant to participate in mixers, since they cannot be sure what coins are tainted and what coins are clean. If they choose to exchange with seemingly clean coins, it is very likely that those coins shortly after will end up on a blacklist. This will cause participants with lower levels of taint to withdraw, which in turn will lead to an increase in the average level of taint in mixers. This will prompt even more participants to leave, causing the taint to rise further. Eventually, the level of taint in the mixers will reach 100%, after which mixers will have no utility as tools for laundering coins.

In this paper, the authors also introduce a new blacklisting strategy called the *seniority* principle. Here blacklisted coins are always placed first in any transaction. If more than one output address is used, it is thus, similar to FIFO, possible to separate tainted and clean coins.

In 2019 Möser *et al.* wrote an updated version of the paper from 2014 where they go more into detail about the ramifications of the different blacklisting strategies [98]. They are strong advocates for the use of public blacklists. Currently, there is no universal blacklist that is used by the entire Bitcoin community. Instead, the most complete blacklists belong to blockchain analytics companies such as Chainalysis and Elliptic. Since having a more comprehensive blacklist than your competitors might lead to a competitive advantage, companies are incentivised to hide their blacklists from each other. The indirect consequence of this is that those same blacklists are also hidden from the public. This means that blacklisted coins are mostly unknown to private investors, which makes it easier for illicit actors to launder their tainted coins on platforms dominated by these investors, such as mixers and decentralised exchanges. A decentralised exchange is an exchange entirely hosted on a cryptocurrency through smart contracts.

While one universal public blacklist would be ideal, the authors believe it would be more realistic for countries and agencies within those countries to have their own blacklists tailored towards their own areas of interest. In such a scenario quick exchange of information would be key in maintaining good coverage and hindering tainted coins from being cashed out on exchanges in other jurisdictions. Time delays on transactions could be used to decrease the likelihood of illicit coins being sold before relevant blacklists can be updated in time.

The authors also outline a new strategy called *output-based seniority*, which is a mix of the seniority and FIFO strategies described above. The ordering of the taint is preserved in the same way as in the FIFO strategy. The key difference is how taint is distributed inside each output. For FIFO the ordering is also preserved within

each output while in output-based seniority the taint is always moved to the front of each output.

The paper also examines the advantages and disadvantages of the different strategies. The FIFO and seniority strategies have the advantage of preventing the diffusion of taint, which is very likely when using the haircut strategy. They also allow for the possibility of separating tainted coins from clean ones, which could be useful if exchanges started to accept only 100% clean coins. However, this also makes these strategies what the authors call gameable. Since it is possible to separate out the taint, a person with knowledge about the origin of certain coins could manipulate the ordering of the inputs and outputs of a transaction in order to offload tainted assets onto other users. In a situation with limited information, these strategies favour users in control of tainted coins since they will be able to arrange transactions to favour themselves while not appearing suspicious to other users.

The authors also highlight the importance of taint being transferred to transaction fees. If taint is not transferred to validators, they might be complicit in money laundering without receiving any punishment. It is also conceivable that some might attempt to launder money by performing transactions with abnormally large transaction fees. If an illicit actor is in control of a validatory, they could create a transaction where the transaction fee is transferred to themselves. All they would have to do would be to wait for their validator to produce a block and then perform the transaction. If the validator withholds the block until they have broadcast their transaction, they will be able to make sure that their transaction is added to their own block. To increase efficiency, the transaction fee could be increased to an abnormally large number so that only a single transaction is necessary to launder a large number of coins. Because of this, it is important for any blacklisting approach to make sure that taint is also transferred over to the miner.

Towards the end of their paper, the authors also discuss some common concerns with blacklists for cryptocurrencies. They debate whether blacklisting might lead to criminals switching from public blockchains to privacy coins. If this was the case it would not necessarily be a bad thing. Bitcoin would become cleaner allowing it to be more widely accepted. Privacy coins already have tenuous relationships with regulators. It is likely that using privacy coins and especially trying to cash out privacy coins might become more and more restricted. There is also concern that blacklisting could quickly turn into whitelisting. According to the authors, this would be much harder to maintain since the portion of blacklisted coins is much smaller than all the coins that would have to be whitelisted. Finally, they address swaps across chains. This would make tracking much harder, but since the swaps usually involve some kind of selling to another party that party will be incentivised to check the funds they are receiving against the blacklist which alleviates the issue.

3.3 Machine learning

Since blockchain data is mostly public, it should be possible to use machine learning techniques to extrapolate purpose from cryptocurrency transactions and addresses. This can in theory also be applied to illicit actors on the blockchain. Since machine learning and blockchain are both relatively new and active research fields, there is a lot of research combining the two. Below follows summaries of two papers that we believe give a good idea of the state of the machine learning AML field, both within traditional and blockchain-related financial services.

In 2018 Chen *et al.* published a survey where they list the main machine learning techniques used for AML within the traditional financial system [99]. The authors chose to cover six major approaches to AML using machine learning and summarised a number of papers for each approach. The covered approaches were AML typologies, link analysis, geographic capabilities, behavioural modelling, risk scoring, and anomaly detection. AML topologies refer to methods able to detect already known money laundering patterns. This can be done using a range of techniques such as rule-based logic, and neural networks. Link analysis on the other hand concentrates on establishing the nature of relationships between different bank accounts. This can be done using clustering or a more rigorous approach where detailed profiling data is gathered to determine actual relationships between different users. Behaviour modelling tries to combine information from different AML domains to determine what users are actually doing based on transactional behaviour. One method used was comparing users with their closest neighbours in order to determine when they perform outlier transactions. In risk scoring, the purpose is to generate a risk score for any given transaction or user. This can be done using several different methods, including decision trees, sequence matching, and adaptive resonance theory. Anomaly detection deals with the ability to detect outliers from normal transactional behaviour. Geographical capabilities refer to the ability to track money laundering across language and country boundaries. Lastly, multi-aspect combination deals with the capability to combine data from different monetary domains, such as credit card and insurance payments. See the original paper for a more detailed explanation of these concepts [99].

The authors conclude that topology and outlier detection methods are the most popular among academic practitioners. Sadly, most studies performed on the area used generated data to test their solution. Real transactional data was used for normal transactions while money laundering activities were artificially introduced into the dataset. Only a handful of the methods examined used real illicit transactions. This makes it hard to compare the methods against each other since the accuracy results achieved by the different methods are highly dependent on how the artificial data was generated. The authors also observe that most of the examined methods used relatively small datasets containing less than 10,000 data points. This makes it hard to say how well these methods would scale for use in the real world where they will have to handle millions of transactions. The authors also note that almost none of the methods supports reinforcement learning meaning they would have to be

retrained every time new data is introduced. For more details

In a paper from 2021, Lorenz et al. set out to investigate what might be the most effective way of finding illicit transactions on Bitcoin through machine learning while dealing with label scarcity [100]. They begin their investigation by proving that the unsupervised learning used by earlier papers is not feasible in a realistic setting. They show that clustering algorithms falsely list outlier transactions as illicit when in reality there is no correlation between outliers and illicit transactions. They attribute the success of earlier papers to the use of generated data. This generated data used fake illicit transactions that contrary to the real data did in fact mostly turn out to be outliers. They then moved on to investigate the best way of implementing active learning in order to achieve high accuracy for label scarcity. The first step was to establish a supervised baseline which they did by reproducing the results achieved by Weber et al. [101]. The next step was to reproduce the results using active learning to create classifiers using random forest, XGBoost, and logistic regression. The former two are ensemble tree-based methods for classification and regression, and the latter is a statistical technique for binary classification based on probability. They were able to achieve accuracies with assisted learning that were at least as good if not better than the supervised baseline. They also did this while only using 1.7-10% of the labelled data available to the supervised counterpart. The highest classification accuracy they achieved was 83% while using 10%. Notable is that they were able to achieve 77% accuracy while only using 1.7% of the label data, which only accounts for 500 data points. This paper shows that it is certainly possible to achieve fairly accurate results for detecting completely new illicit transactions, though only if labelled data is present.

3.4 Automatic Ponzi scheme detection

EOA addresses are not the only possible form of illicit actors on the blockchain. Throughout the history of Ethereum, there have been many smart contracts posing as legitimate services when they were in fact Ponzi schemes designed to swindle users out of their assets. A Ponzi scheme is a form of fraud that is designed to give the appearance of generating money by paying out generous profits to investors. The catch is that the scammer is paying back earlier investors with the money of newer more recent investors. As long as the interest in the Ponzi scheme grows and the number of investors increases, there will be enough money to pay back early investors, which in turn will generate more interest. The end goal of the Ponzi scheme is to cut off payouts when a large number of people have simultaneously invested. If everything goes to plan, this will leave the scammer with a large amount of money. In a paper from 2019, Weili et al. set out to investigate whether it is possible to automatically detect Ponzi schemes by scanning the bytecode of published smart contracts on Ethereum [102].

The first step was to establish the ground truth. The authors did this by reading through open-source smart contracts and manually classifying them as legitimate or Ponzi schemes. This way they were able to create a dataset of 3780 smart contracts

containing 200 Ponzi schemes. Since smart contracts are stored on the blockchain in the form of bytecode, there would be no point in having a training set consisting of source code. To remedy this, the authors first compiled their dataset into bytecode and then disassembled it into operation instructions. This last step was done to make it easier to extract features from the dataset. Since bytecode is only a string of numbers, converting it into human-readable operation code made it easier to identify specific commands. These commands can then be counted resulting in 64 unique features. This conversion into operation code and the subsequent counting of bytecode instructions were also performed on the bytecode coming directly from the blockchain. The authors also extracted 13 account features pertaining to the transactional history of the smart contracts, such as the balance of the smart contract, the total number of transactions, and the largest outward payment made by the smart contract. These features were combined with the 64 code-related features and used as the training set in the next step.

A range of different classifiers were trained on the data, including decision tree, XGBoost, and random forest. Among these, random forest performed the best with an overall F-score of 0.79. Interestingly the score increased to 0.82 when only using code features. In contrast, none of the examined classifiers produced good results when only using transactional features, showing that these are not good predictors for whether a smart contract is a Ponzi scheme. Using the random forest classifier the authors then scanned all smart contracts introduced between August 2015 and September 2017. They found 507 smart contracts labelled as Ponzi schemes which is 0.03% of all smart contracts published during that time.

4

Method

This chapter describes the methods used in this thesis, with a particular focus on the more challenging aspects of the project. The first section describes three major design decisions we had to make before starting the implementation of the tool. Then comes a compact description of our tool development. A longer version of this can be found in chapter 5. The two last chapters talk about our algorithm implementations. First, the DAR clustering algorithm, followed by our various blacklisting algorithms.

4.1 Initial design considerations

At the start of the thesis, we had to take several factors into account when deciding how to implement the tool. The three largest of these are summarised in this section.

4.1.1 What strategies to use in the tool

From the strategies discussed in the literature review, we selected blacklisting and clustering to be implemented in the AML tool. We deemed blacklisting the most important since we believe it to be the most fundamental way of preventing money laundering with cryptocurrencies. If illicit coins are discovered using other means, there still has to be some way to keep track of them on the blockchain. There is no way to freeze the movement of funds on Ethereum. Thus some form of tacking is needed, otherwise detected illicit coins would be lost as soon as they are moved to another address. This makes blacklisting crucial, which is why we decided to implement it before clustering. Also important to mention is that blacklisting and tracking give plenty of value all on their own. Since there are several publicly available datasets of illicit or otherwise labelled addresses [103]–[105], another method is not necessary in order to have something to track.

We chose clustering as the second method, as we believed it would allow us to greatly expand the number of blacklisted addresses. Our plan was to first track from known illicit addresses and then cluster from the result. If a cluster turns out to contain one or more illicit addresses, the other addresses in that cluster can be marked as having a high likelihood of also being illicit.

We decided against using the other strategies either because of the believed difficulty or due to a lack of results in the literature. Machine learning was the most promising out of the other strategies and was thus put as a stretch goal that we never had time to investigate. Automatic Ponzi scheme detection is fairly specific in its use case, which meant that it would never be able to detect very many illicit addresses. This in combination with doubts about the feasibility of achieving an effective implementation led us to deprioritise it in favour of the other strategies.

4.1.2 When to process the data

One of the first decisions that had to be made during the design of the tool was when the data should be processed. Two main ways of doing this came to mind. We could either preprocess the illicit status of every address in advance or process the illicit status of one address at a time. The latter would mean that the tool would serve user requests on demand, only calculating the illicit status of an address after a user has requested it to do so. Since Ethereum has millions of addresses, calculating all the results in advance would most likely mean that a large majority of the data would never be used. Because of this, we initially considered calculating the illicit status of addresses one at a time.

The main problem with calculating the illicit status of addresses on demand is the amount of time a user would have to wait after making a request. Calculating the illicit status for a single address means checking whether any of the assets in that address originated from an illicit actor. This in practice involves analysing the transaction data on the blockchain in order to recreate the path the assets took before they ended up in the account that is being analysed. This is where on-demand tracking could incur major time delays. The illicit status of an address could depend on a large number of incoming transactions from other addresses. The illicit status of these transactions, in turn, depends on the illicit status of the senders who have incoming transactions of their own, and so on. From this, it is apparent that the number of addresses that will be involved when calculating the illicit status of one address is likely to grow exponentially the further back-in-time tracking is performed.

At this point, we considered whether it would be viable to put a cap on how far back in time a tracking request should travel. However, this would make it possible for illicit actors to avoid detection simply by waiting. Alternatively, the tracking could be limited by how many transactions backwards the algorithm would be allowed to track. But this would again be avoidable by performing a large number of transactions in succession. Having an easy way for illicit actors to avoid detection was not an option for us, which meant that lengthy loading times were inevitable when tracking on-demand.

Due to the aforementioned issues with on-demand tracking, we once again considered calculating the illicit status of every address in advance. It would avoid the issue of extended loading times since each request would involve only a single lookup in the database that stores the results. But it would also introduce new technical challenges,

the biggest one being how to make the algorithms effective enough to calculate the illicit status of every single address on Ethereum within a reasonable amount of time. We realised though that tracking for every address at the same time would make it possible to significantly increase the effectiveness of our tracking algorithms.

As we have already established, the illicit status of a single address in the present is likely to depend on the illicit status of a large number of other addresses in the past. If we were to calculate the illicit status for two addresses at the same time, there would be a possibility that both addresses depend on one common address in the past. If such a collision exists, the calculations required to determine the illicit status of that common address can be shared between the two tracking instances, which in turn will decrease the overall time required to complete the computations. The further back-in-time tracking is performed, and the more addresses that are tracked at the same time, the more likely these collisions are. See Figure 4.1 for an illustration of this phenomenon. When performing tracking for every address on Ethereum, these collisions become so common that the time required to track only grows linearly in relation to how far back in time tracking is performed. This is a significant improvement over the exponential growth that would otherwise be experienced when tracking for one address in isolation. The caveat to this efficiency increase is that it is only possible when calculating the illicit status for every single address on Ethereum at the same time. Since Ethereum has more than 230 million addresses, this is still a large undertaking [106].

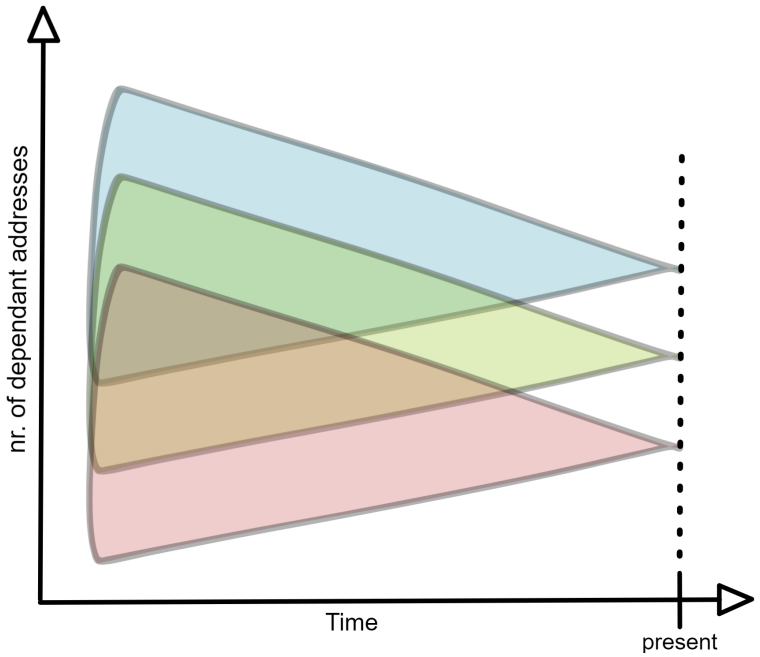

Figure 4.1: *An illustration of the overlap that occurs when tracking is performed for three addresses at the same time. Here each triangle represents the search space required to calculate the illicit status for a single address as it moves backwards in the blockchain.*

In order to avoid lengthy loading times, we decided to go with the approach of computing everything in advance. What this meant in practice was that we would be performing tracking starting at the beginning of the blockchain and moving forward in time, rather than starting at the present and moving backwards. Since we would be tracking for every address, there would be no point in having one backwards-propagating request per address. The easier solution would be to go through each transaction in chronological order and perform tracking incrementally, stopping and applying the relevant tracking rules whenever an illicit transaction is made. This way a record of the illicit state of every address will be built. Every time a transaction is found to contain tainted funds, the illicit status of the sender and receiver addresses is updated. After all of the transactions have been processed, the result would be a complete record of the illicit status for every address on Ethereum. This can then be used to serve user requests without any delays.

4.1.3 Multithreaded tracking

Despite the large efficiency gains of tracking the illicit state for every address simultaneously, we still had concerns about the total time required to complete the calculations. Thus, we decided to investigate whether there was a way to make the tracking multithreaded. As has been touched upon earlier in this thesis, a blockchain is essentially a long list of transactions ordered chronologically. This makes it very difficult to perform any tracking with more than one thread.

If there is a transaction from A to B directly followed by another transaction from A to C, it is important that these transactions are tracked in chronological order. If A at the time of the transactions contained illicit funds, the order of these two transactions could determine whether the illicit funds ended up in B or C. If we decide to perform tracking in parallel on two different threads, these two transactions might be processed independently of each other. In that case, the illicit funds would end up in B on one thread and in C on the other. If the threads continue to work in parallel, the errors caused by the threads being out of sync could grow exponentially as the result of one faulty transaction serves as the input to another. When merging, the result of at least one of the threads will have to be discarded, meaning that there was no point in parallelising in the first place.

To solve this issue of interdependence between transactions, some form of sorting of the values would be necessary. Perhaps, clusters could be created where all of the addresses within a cluster interact only with other addresses within the same cluster. If such clusters exist, tracking could be performed within the cluster separately without having any conflict on merge. However, such clusters would probably only exist during limited timeframes, which would make them hard to find. The search process would also have to be fast enough so that it does not waste more time than it saves through parallelisation.

Just such a solution appears to have been found by Baran et al. [107], but we were unaware of this at the time of implementation. Even if we had discovered it, it would

most likely still have been too complex to implement within the scope of the project. Since we neither found nor had the time to invent a viable solution, we decided to go with the single-threaded approach. This meant that we would have to optimise the algorithms as much as possible so that single-threaded tracking could be performed within a reasonable amount of time.

4.2 Tool development

Due to the broad research questions, there were many possible directions in which the tool could be developed. Thus, the first step was to narrow down the scope of the tool. This was done by discussing the research question both internally and with Handelsbanken. These discussions, combined with the literature review, did not result in concrete requirements but rather in a number of ideas regarding the general direction of the application development. These ideas can be summarised with the following list:

- Create an open-source tool.
- Take advantage of a cloud provider for computing and storage needs.
- Implement different blacklisting algorithms in order to calculate the taint of addresses.
- Use public data sets of known illicit addresses as a starting point.
- If there is time, investigate other promising AML methods, such as clustering and machine learning, in that order.

The purpose behind creating a tool was to have one coherent system that could be used for presenting the results. The decision to make it open source was motivated by the secrecy that otherwise surrounds these types of tools. As has previously been discussed, companies such as Chainalysis claim to have similar tools, but they are not public about what algorithms they use, or how these are implemented. In order to further an open discussion in the scientific community we decided to publish the code along with our results. Handelsbanken had no objections to this since they were mainly interested in the knowledge gained from creating the tool rather than using the tool in any of their own products or services.

Initially, the idea was to produce one coherent system capable of doing everything from data collection to serving the results to the front end. This quickly proved to be unrealistic due to the amount of flexibility required in a research project like this. By instead completely decoupling the different parts of the system, it became possible to develop each part independently, allowing us to perform complete rewrites of individual components without affecting other parts of the system.

The first step of development was to create a pre-processing pipeline that could acquire, prune, and sort the data necessary for our algorithms. We decided to use *Google BigQuery* for acquiring the necessary Ethereum blockchain data. From there we extracted the *traces*, *transactions* and *balances* datasets for use in the tool. The

data was transferred to a *Google Storage Bucket* where it was pruned and sorted to fit our algorithms. All of these steps were performed by a series of Python and bash scripts developed for the purpose. Initially, only small portions of the datasets were used during the development of the algorithms. Only when the algorithms were thought to be complete did we download and sort all 4TB of data.

The algorithms themselves were developed in Python. These were then tested using sample data before being applied to the entire dataset. The final execution of the algorithms was done using virtual machine instances provided by Google Cloud Compute. The results from the blacklisting algorithms were stored in a *PostgreSQL* database, while the less relational results of the clustering were stored in a *MongoDB* database.

In order to provide easy access to the results, a web API was built using the Python *FastAPI* framework. This API was hooked up to a simple *React* frontend and hosted on a public domain. The PostgreSQL and MongoDB databases, along with the API, were self-hosted.

Further details on the intricacies of implementing the tool can be found in chapter 5. What follows is a more in-depth explanation of how we implemented the algorithms. First DAR and then the blacklisting algorithms.

4.3 DAR algorithm

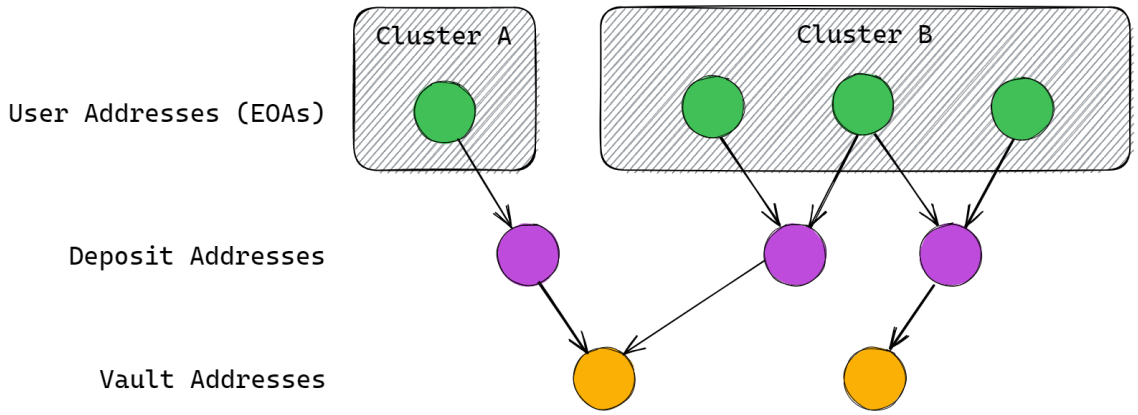

Figure 4.2: *An example graph illustrating the three different types of addresses in the DAR algorithm. Clusters A and B illustrate the addresses controlled by two different entities.*

As explained previously, the DAR algorithm is a powerful tool for identifying related Ethereum addresses, which can be useful for understanding transaction patterns and potentially detecting fraudulent activity. The implementation of DAR used in this thesis was developed by the authors of the *Tutela* paper [91]. Only minor tweaks have been done to make the algorithm work with the rest of our code.

When trying to find clusters of related addresses with DAR, the first step is to

find all transactions going to deposit addresses used by centralised exchanges (the purple nodes in figure 4.2). Exchanges usually have one unique deposit address per customer. This means that there is a high probability that the addresses that send funds to the same deposit address belong to the same entity.

4.3.1 Collecting transaction chains

Finding all of these transactions was done in a step-by-step process. First, all potential deposit addresses were collected by looping through each transaction in the *transactions dataset* while filtering out every transaction where the sender was a mining address or the receiver was not a vault address. These transactions, as well as the sender addresses in these transactions, were collected. After that, all addresses that had sent funds to any of the potential deposit addresses were collected by looking at the to-address of every transaction.

At this point, you have every potential transaction to a deposit address and every potential transaction from a deposit address to a vault address. Now these two lists of transactions have to be merged in order to find the tuples of transactions that make up the chain from EOA to vault address, as illustrated in figure 4.2.

The method used to find these chains is primarily dependent on two hyperparameters:

- α: the maximum difference in ether allowed between the incoming transaction to the deposit address and the outgoing transaction to a vault address. This is useful since the role of a deposit address is to forward the entire received sum, meaning that the difference should be small and similar to the transaction fee.

- τ: represents the maximum time difference, measured in blocks (approximately one block is produced every 12 seconds), between the incoming transaction to the deposit address and the outgoing transaction to a vault address. In other words, the DAR algorithm uses τ to determine if the time between two transactions is short enough to consider them related. This is useful because deposit addresses usually try to send funds relatively quickly to the connected vault address.

In order to combine these two lists of transactions into one, an *as-of-merge* algorithm was utilised. This works like an ordinary *left join* between two sets of data, but the merge of rows is performed on the nearest match of keys instead of equal keys. The implementation we used[1] also allowed us to specify columns that must be equivalent for a merge to occur. We used this to check whether any of the *to address* in the EOA to deposit address transactions ($EOA \rightarrow Deposit\,Address$) matched any of the *from address* in the deposit to vault transactions ($Deposit\,Address \rightarrow Vault$). If such a match occurs and the difference in block number lies within the range of τ, the two transactions are merged. This results in complete DAR chains going from

[1] `https://pandas.pydata.org/pandas-docs/version/0.25.0/reference/api/pandas.merge_asof.html`

EOA to deposit and then to a vault address ($EOA \rightarrow Deposit\,Address \rightarrow Vault$). If there are multiple valid combinations involving the same transaction, only the combination with the smallest difference in *block number* will be joined.

The other hyperparameter, α, is then applied to the list of joined transactions. If the difference between the amount received by the deposit address and the amount sent from the deposit address exceeds α, the transaction is filtered out.

The pseudocode implementation of the process described above can be found in Figure 1.

```
Data: Transactions, MiningAddresses & VaultAddresses
begin
    foreach transaction in Transactions do
        if MiningAddresses contains transaction.fromAddress then
            end iteration
        end
        if VaultAddresses contains transaction.toAddress then
            add transaction to TransactionsToVaults
        end
    end
    PotentialDepositAddresses ← all fromAddresses in TransactionsToVaults
    foreach transaction in Transactions do
        if PotentialDepositAddresses contains transaction.toAddress then
            add transaction to TransactionsToPotentialDeposits
        end
    end
    maxBlockDiff ← 3200
    maxEthDiff ← 0.01
    foreach transactionA in TransactionsToVaults do
        foreach transactionB in TransactionsToPotentialDeposits do
            if transactionA.fromAddress = transactionB.toAddress and
             transactionA.block ≥ transactionB.block and
             transactionA.block - transactionB.block ≤ maxBlockDiff and
             abs(transactionA.value - transactionB.value) ≤ maxEthDiff then
                add transactionB to PotentialDepositTransactions
            end
        end
        if PotentialDepositTransactions is empty then
            end iteration
        end
        trueDepositTransaction ← find transaction in
         PotentialDepositTransactions where transactionA.block −
         transaction.block is smallest
        add (transactionA, trueDepositTransaction) to ValidTransactionChains
    end
    return ValidTransactionChains
end
```

Algorithm 1: *Pseudocode for finding connected EOAs, deposit addresses, and vaults.*

All transactions needed to calculate the clusters are now collected. But before moving on to generating the cluster graphs, some cleaning is performed on the data. Since there is no uncertainty regarding the legitimacy of vault addresses, a quick check is done to remove any transactions where a known vault address appears as an EOA or deposit address. Another check is also done to make sure that no address appears as an EOA in one transaction and a deposit in another. In these cases, the transactions

where they appear as a deposit address are removed.

4.3.2 Generating cluster graphs

The first step in constructing directed graphs representing the clusters was to create the vertices and edges. The vertices were created by simply combining the EOA and the deposit addresses into a single dataset. The edges were also straightforward, as it only involved taking each transaction and generating a tuple consisting of the EOA address as the source vertex and the deposit address as the target.

To make graph analysis easier, the next step was to combine the vertices and edges into a data structure that represents a directed graph. This was done with the help of *igraph*[2], a collection of network analysis tools available in a variety of languages.

Finding the clusters from this point is done by simply finding the *weakly connected components* within the graph. Each weakly connected component is a cluster.

4.4 Blacklisting algorithms

This section explains our implementation of the blacklisting algorithms. They were originally implemented using Python but will be demonstrated here as pseudocode to make them easier to understand and to increase generalisability. The implementations of these algorithms are loosely based on the algorithm descriptions provided in the series of papers by Möser et al. [95], [96], [98], but are otherwise entirely our own contributions. This especially applies to Advanced FIFO which was not described in any of the papers mentioned above.

4.4.1 Poison

As can be seen in Algorithm 2, the implementation of poison was fairly straightforward. The core of the algorithm is the loop on rows 4-7. Each iteration checks if a given transaction is transferred from an address that is already on the blacklist. If this is the case, the receiver address is also added to the blacklist. This is repeated until every transaction has been computed. To start the algorithm, a list of addresses is provided that are known to be illicit. Since poison does not keep track of the balances of each blacklisted address, the result only consists of the addresses recorded in the blacklist.

[2]https://igraph.org

```
1  Data: IllicitAddresses & Traces
2  begin
3      add IllicitAddresses to Blacklist
4      foreach trace in Traces do
5          if Blacklist contains trace.sender then
6              add trace.receiver to Blacklist
7          end
8      end
9      return Blacklist
10 end
```

Algorithm 2: *Pseudocode for the poison blacklisting algorithm.*

4.4.2 Haircut

As can be seen on rows 3-7 in Algorithm 3, haircut begins by adding the list of known illicit addresses to the blacklist. These are given an infinite amount of tainted funds, which means that any funds leaving these addresses will always be 100% tainted. At rows 8-10, blockRewardAddress also receives an infinite amount of untainted funds.

Rows 11-24 contain the main loop that runs over every trace. The trace is first split into sender address, receiver address, and transaction value. If the receiver is not known, it is added to the blacklist with both balance and taint starting at zero. Here we can assume that the sender is already in the blacklist since a sender would first have to have received funds, which would have been captured in an earlier trace. On rows 21 and 22 the balances of the two addresses are updated by deducting the value from the sender and adding it to the receiver. On row 23 the new taint of the receiver is calculated from the taint it had before and the taint of the sender. Here, the ratio between the receiver's balance and the value of the transaction determines how much each of the initial taint values will influence the receiver's new taint value.

The final step of the algorithm is to return the blacklist, which is a combination of the known addresses, their corresponding balances, and their corresponding taint values.

```
1  Data: IllicitAddresses, Traces & blockRewardAddress
2  begin
3      foreach address in IllicitAddresses do
4          add address to KnownAddresses
5          Balances(address) ← ∞
6          Taints(address) ← 100%
7      end
8      add blockRewardAddress to KnownAddresses
9      Balances(blockRewardAddress) ← ∞
10     Taints(blockRewardAddress) ← 0%
11     foreach trace in Traces do
12         sender ← trace.sender
13         receiver ← trace.receiver
14         value ← trace.transactionValue
15         if KnownAddresses does not contain receiver then
16             add receiver to KnownAddresses
17             Balances(receiver) ← 0
18             Taints(receiver) ← 0%
19         end
20         oldBalance ← Balances(receiver)
21         Balances(sender) ← Balances(sender) − value
22         Balances(receiver) ← Balances(receiver) + value
23         Taints(receiver) ← (Taints(sender) · value / Balances(receiver)) +
               (Taints(receiver) · oldBalance / Balances(receiver))
24     end
25     return Blacklist(KnownAddresses, Balances, Taints)
26 end
```

Algorithm 3: *Pseudocode for the haircut blacklisting algorithm.*

4.4.3 Seniority

As can be seen in Algorithm 4 on rows 3-6, seniority starts off similarly to haircut by giving all known illicit addresses an infinite amount of tainted ether. On rows 7-23, it runs through each trace and tracks the movement of the funds originating from the known illicit addresses. First, it checks if the receiver is part of the blacklist and adds it if it is not. Then, if the transaction value is smaller than the tainted value in the sender, the trace is recorded as a 100% tainted transaction going from the sender to the receiver. If the tainted content of the sender is, instead, smaller than the value of the transaction, only the total taint in the sender is transferred. Since tainted funds are always passed first, there is no need to keep track of untainted funds. The resulting blacklist consists of a combination of the blacklisted addresses along with their tainted balances.

```
1  Data: IllicitAddresses & Traces
2  begin
3      foreach address in IllicitAddresses do
4          add address to KnownAddresses
5          TaintedBalance(address) ← ∞
6      end
7      foreach trace in Traces do
8          sender ← trace.sender
9          receiver ← trace.receiver
10         value ← trace.transactionValue
11         if KnownAddresses does not contain receiver then
12             add receiver to KnownAddresses
13             TaintedBalances(receiver) ← 0
14         end
15         if TaintedBalances(sender) > value then
16             TaintedBalances(receiver) ← TaintedBalances(receiver) + value
17             TaintedBalances(sender) ← TaintedBalances(sender) − value
18         end
19         else
20             TaintedBalances(receiver) ← TaintedBalances(receiver) +
                                           TaintedBalances(sender)
21             TaintedBalances(sender) ← 0
22         end
23     end
24     return Blacklist(KnownAddresses, TaintedBalances)
25 end
```

Algorithm 4: *Pseudocode for the seniority blacklisting algorithm.*

4.4.4 FIFO

Like the other algorithms, FIFO begins by adding known illicit addresses to the blacklist; see Algorithm 5 rows 3-6. Instead of giving each address a taint value they are given a queue that in turn contains values. A value inside one of these queues represents an either entirely tainted or untainted ether fragment. The initial values added to the queues of known illicit addresses are tainted fragments with an infinite balance. As can be seen on row 7, the block reward address is also given a fragment with infinite balance, but this one is untainted instead of tainted.

On rows 8-28, FIFO loops through each trace and tracks the movements of fragments between queues. Inside this first loop on rows 15-27, there is a second loop where each iteration moves a single fragment from one queue to another. Fragments will be transferred until the combined value of the transferred fragments is equal to the total value of the trace.

On row 16 the first fragment in the sender's queue is observed. This is then compared to the value of the trace. If the fragment is smaller, it will be transferred to the

queue of the receiver, as can be seen on rows 22-26. The fragment is removed from the sender queue, added to the receiver queue, and the trace value is reduced with the fragment value. If the value of the fragment is instead larger than the value of the trace, the entire value of the trace will be added to the receiver's queue, as can be seen on rows 17-20. Since the fragment is larger than the trace, only a part of the fragment is removed from the sender.

When all traces have been processed, the resulting blacklist consists of the address-queue pairs recorded by the algorithm.

```
1  Data: IllicitAddresses, Traces & blockRewardAddress
2  begin
3      foreach address in IllicitAddresses do
4          add address to KnownAddresses
5          enqueue (∞, tainted) in Queues(address)
6      end
7      enqueue (∞, untainted) in Queues(blockRewardAddress)
8      foreach trace in Traces do
9          sender ← trace.sender
10         receiver ← trace.receiver
11         leftToTransfer ← trace.transactionValue
12         if KnownAddresses does not contain receiver then
13             add receiver to KnownAddresses
14         end
15         while leftToTransfer > 0 do
16             fragment ← peak at first in Queues(sender)
17             if fragment.value > leftToTransfer then
18                 enqueue (leftToTransfer, fragment.taint) in Queues(receiver)
19                 update first in Queue(sender) to fragment.value -
                       leftToTransfer
20                 leftToTransfer ← 0
21             end
22             else
23                 enqueue fragment in Queues(receiver)
24                 dequeue first in Queue(sender)
25                 leftToTransfer ← leftToTransfer − fragment.value
26             end
27         end
28     end
29     return Blacklist(KnownAddresses, Queues)
30 end
```

Algorithm 5: *Pseudocode for the standard FIFO blacklisting algorithm.*

4.4.5 Advanced FIFO

The advanced FIFO algorithm is very similar to the original FIFO algorithm. The main difference is that it not only tracks tainted funds but also keeps track of their origins. On row 5 the address of each known illicit address is added to the corresponding fragment. On row 18 the origin of the fragment from the sender is transferred to the new fragment that is added to the receiver's queue. Similarly, on row 23, when the entire fragment is transferred from the sender to the receiver, the origin address will be transferred with it.

```
1  Data: IllicitAddresses, Traces & blockRewardAddress
2  begin
3      foreach address in IllicitAddresses do
4          add address to KnownAddresses
5          enqueue (∞, tainted, address) in Queues(address)
6      end
7      enqueue (∞, untainted, blockRewardAddress) in
         Queues(blockRewardAddress)
8      foreach trace in Traces do
9          sender ← trace.sender
10         receiver ← trace.receiver
11         leftToTransfer ← trace.transactionValue
12         if KnownAddresses does not contain receiver then
13             add receiver to KnownAddresses
14         end
15         while leftToTransfer > 0 do
16             fragment ← peak at first in Queues(sender)
17             if fragment.value > leftToTransfer then
18                 enqueue (leftToTransfer, fragment.taint, fragment.origin) in
                     Queues(receiver)
19                 update first in Queue(sender) to fragment.value -
                     leftToTransfer
20                 leftToTransfer ← 0
21             end
22             else
23                 enqueue fragment in Queues(receiver)
24                 dequeue first in Queue(sender)
25                 leftToTransfer ← leftToTransfer − fragment.value
26             end
27         end
28     end
29     return Blacklist(KnownAddresses, Queues)
30 end
```

Algorithm 6: *Pseudocode for the advanced FIFO blacklisting algorithm.*

5

Implementation Details

This chapter contains a more in-depth walk-through of what we did to achieve our results. It is mainly for transparency and to enable others to recreate our results. It is not strictly necessary to read in order to understand the rest of this thesis.

The chapter begins with a description of two different methods of acquiring Ethereum blockchain data and which one we chose. The following section describes the standard data format often used when storing Ethereum blockchain data. Then comes a section where we describe the preprocessing pipeline we constructed in order to download, prune, and sort the data. Then comes a section describing how our algorithms fit into the general architecture of the tool. After that comes a section discussing how we stored our results followed by a section on how we developed an API capable of serving those results. Then we describe how we built the frontend that makes the calls to the API, and then, finally, we have a section on how we processed and analysed the results to extract the data used for our graphs and tables in the result chapter.

5.1 Data acquisition methods

The sheer size of the Ethereum blockchain and the fact that blockchain data is ordered makes collecting it a difficult task. Two options exist for acquiring data from the Ethereum blockchain. Either you collect it yourself by setting up a private node, or you trust data collected by someone else. From a security perspective, the former is better since it removes any possibility of tampering with the data. The latter is often more convenient though, since the work required to acquire and store the data is done by someone else.

5.1.1 Local Ethereum node

In order to collect Ethereum data yourself, you need access to an Ethereum node. Running your own node is fairly easy, as detailed setup instructions are available online. The two main drawbacks are the hardware and time requirements. A computer running an Ethereum node requires a large amount of storage space to store all the blockchain data. The node will also need a long time to catch up with

the current state of the Ethereum VM.

There are several synchronisation modes to choose from when running your own node. To be able to collect all historical data and produce the datasets mentioned above, you need to run your node in a mode that downloads all blocks (including headers, transactions, and receipts). This results in the node becoming a *full node*. If you also want to be able to query historical balances, the node must be run with the *archive mode* special option, which builds an archive of all historical states of both the blockchain and the EVM. This greatly increases the storage required by the node.

As of March 2022, *Erigon*[1], the most effective Ethereum client with regard to disk-space efficiency [108], recommends an SSD with at least 3 TB storage space and a minimum of 16 GB RAM for running a full archive node according to the project's `README.md`.

When the node is up and running, you can generate the dataset by running the *ethereum-etl* tool and pointing it at your local node. The tool will then query your node for all historical data and save it to disk.

5.1.2 Google BigQuery

An up-to-date version of all the *ethereum-etl* datasets is always available among the public datasets provided by Google through BigQuery. Advanced queries can be performed directly through the browser, or you can download smaller CSV files to your local storage. The two main drawbacks of this approach are that you have to pay for the data you need and that the only way of downloading all the data is to export it to buckets on Google Cloud Storage. The CSV files then, in turn, have to be downloaded from the buckets to your local file or AWS instance.

When you export large datasets from BigQuery, the data will be split seemingly at random into so-called *blobs*, which are smaller files containing subsets of the data. This means that if you need the data in its original order, you have to download all of the blobs, concatenate them, and finally sort them. Depending on the size of the dataset, this process can be very time-consuming. Despite this, it was still decided that Google BigQuery would be used to acquire the data, as the practicality of not having to run an Ethereum node was worth the extra effort.

5.2 Standard Ethereum data format

Ethereum data is usually organised in the same set format, consisting of several distinct categories where each category is its own dataset. This format is produced by the *Ethereum ETL*[2] Python tool. Since Ethereum ETL is used by industry

[1] https://github.com/ledgerwatch/erigon
[2] https://github.com/blockchain-etl/ethereum-etl

giants such as *Google* (using it for their Ethereum *BigQuery* datasets) and *Nansen* (a blockchain analytics company), along with various other actors, this format has become a standard in the Ethereum ecosystem. As the thesis mainly drew data from Google's BigQuery datasets, the categories produced by the ETL tool were also used during the construction of the tool. The categories are listed below:

- **Balances**: Ether balances of all Ethereum addresses.
- **Blocks**: All blocks in the blockchain and their attributes, such as block number, timestamp, difficulty, and miner.
- **Contracts**: Ethereum addresses that, in turn, have attributes such as contract bytecode, function signature hashes, contract creation date, and what type of contract it is, for example, an ERC20 or ERC721 contract.
- **Logs**: Triggered smart contract events along with information such as which transaction triggered the event, the signature of the event, for example, `Deposit(address, bytes32, uint256)`, and the data passed in for the given signature, such as `Deposit(0xb4f...5ea0b, 0x7463...0000, 56)`
- **Token transfers**: Subset of the *Logs* dataset, only containing events where the contract is of type ERC20 and the event signature is `Transfer(address, address, uint256)`. This means that this dataset contains all transfers of ERC20 tokens on the Ethereum blockchain.
- **Tokens**: Contains more information about all ERC20 tokens on the Ethereum blockchain, such as the address of the token's smart contract, its symbol, name, when it was created, and total supply.
- **Traces**: When transacting with a smart contract, actually seeing what the transaction did is very difficult with ordinary transaction data. You can only see whether the transaction succeeded or not. To understand what data it modified and what smart contracts the transaction invoked the transaction must be *traced*. This means that the transaction has to be re-executed through an Ethereum node with access to the historical state of the blockchain and the EVM at the time of the transaction. When re-executing the transaction, additional data collection is enabled. This data contains the otherwise discarded messages sent by smart contracts during the execution of the transaction. These messages make it possible to fully understand how the transaction influenced the internal state of the EVM.
- **Transactions**: All transactions of ether sent from non-smart contract addresses (EOA) along with transaction attributes such as transaction hash, timestamp, gas price, value transferred, nonce, sender, and receiver.

Out of the datasets in Google BigQuery we chose to use the traces dataset for blacklisting since it contains everything necessary to track the flow of funds through the blockchain. For clustering, we instead used the transactions dataset.

5.3 Data preprocessing

Before applying our algorithms to the data it first had to go through a series of preprocessing steps. This involved scripts for downloading, pruning and sorting the data. This section describes these steps in detail.

5.3.1 Getting the data

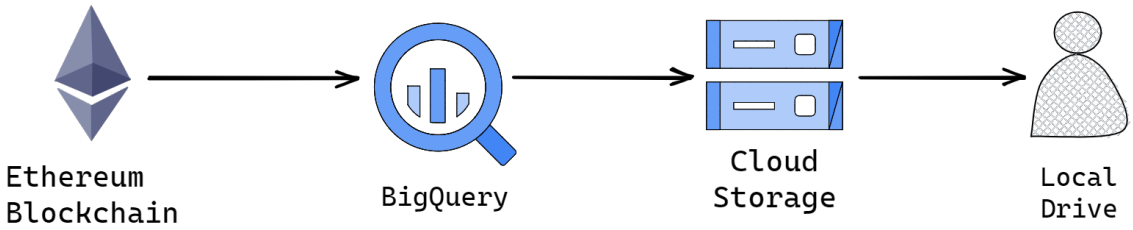

Figure 5.1: *Shows how the transaction data travels before reaching the drives where it will be analysed.*

The download of raw blockchain data from Google BigQuery's public Ethereum datasets was done with the help of Google's `google-cloud-biqquery` python package. This package was used through a facade class in order to simplify the act of exporting an entire BiqQuery table directly to a Google Storage Bucket. The facade was then used inside of a Jupyter notebook, responsible for downloading data locally for development, and also inside the script responsible for downloading data to cloud instances. This flow of data is illustrated in Figure 5.1.

5.3.2 Sorting

Unfortunately, the traces dataset from Google BiqQuery was only sorted by block number. This would not be enough for our blacklisting algorithms, which also require the data to be sorted by *transaction index* and *trace addresses*. The transaction index is an incremental number indicating the order in which the miner processed the transaction while the trace address is a comma-separated list representing the smart contract call stack at the moment of the trace. This meant that we would have to sort the 3TB dataset on our own after exporting it from BigQuery.

We decided to sort the data inside the Google Cloud Storage bucket before moving it to our local drive. The traces dataset ended up being split into more than 23,000 different CSV files with a total size of 3TB. Sorting the approximately 5 billion rows of traces these files contained proved to be challenging, especially since the Google Cloud instance we had access to was limited to only 2TB of HDD storage.

The initial idea was to use a combination of the sorting functionality built into *pandas*, and Python's *heapq merge*. After some test runs, the entire run was estimated to take at least several weeks. This was not an option since an issue with the sort could mean that weeks of work could go to waste. Instead, another solution was found

where the `sort`[3] command, which is part of the `GNU coreutils` was used. This command uses *external k-way merge sort*, making it possible to sort data that cannot fit in memory. Another benefit of the `sort` command is the built-in parallelism, which by default uses all available cores.

5.3.3 Data pruning

Before using the `sort` command, the 3TB dataset had to be reduced by 33% in order to fit within the 2TB cloud instance. After a trial run, it was found that the amount of storage space required for the `sort` command to sort and merge all CSV files was approximately 2.8 times larger than the dataset at its peak. This meant that the data would have to be reduced in size by an additional 65% to fit with the cloud instance. This added up to a total reduction of about 80%.

The traces dataset originally consisted of 20 data columns. The algorithms developed in this thesis utilise at most seven of these columns. Those are `block_number`, `transaction_index`, `trace_addresses`, `from_address`, `to_address`, `value`, and `status`. By cutting the unused columns, the size of the *traces* dataset could be reduced by close to 80%, making it fit within the 2TB cloud instance.

5.3.4 Pre-processing workflow

After having assembled all of the components necessary for preprocessing we combined them using Python and Bash scripts, as can be seen in Figure 5.2. These scripts were then executed in order to make the dataset ready for our blacklisting algorithms.

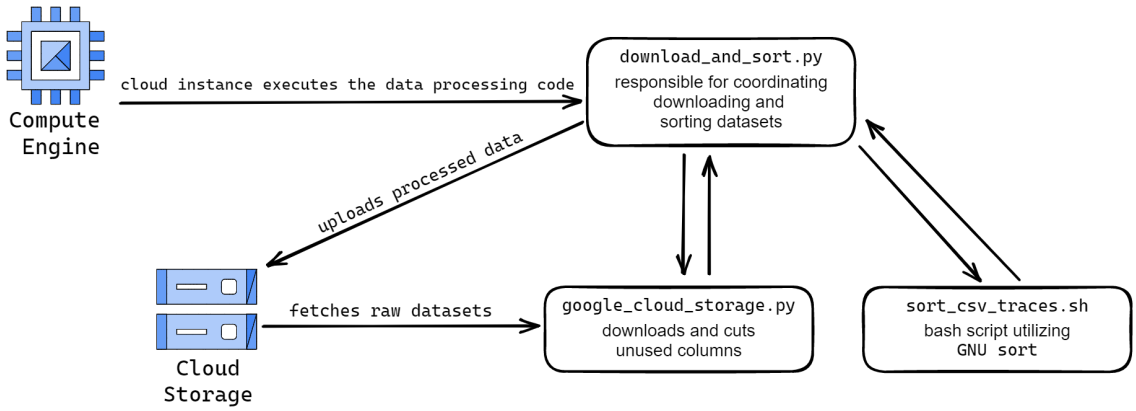

Figure 5.2: *Shows the flow of data between the different preprocessing steps.*

The first step of the preprocessing involved downloading a small number of the raw blobs from BiqQuery. After this, they were pruned from unnecessary columns, reducing the size of the blobs by 80%. The blobs would then be sorted internally, first on *block_number*, then if tied on *transaction_index*, and then finally on *trace_address*, which was enough to place every trace in chronological order. The next step was to upload the blobs to a large cloud storage in order to free up space. This process

[3]https://www.gnu.org/software/coreutils/manual/html_node/sort-invocation.html

was repeated incrementally until all blobs were trimmed and sorted. By making use of multiple cloud instances, this process was parallelised. The final step was to download all of the sorted blobs onto a single cloud instance and perform one last merge-sort of the entire dataset. Due to the previous pruning, the operation could now be done within the 2TB storage limit of the cloud instance. The result of this was one big sorted file, containing all of the traces later used to run the blacklisting algorithms.

5.4 Result storage

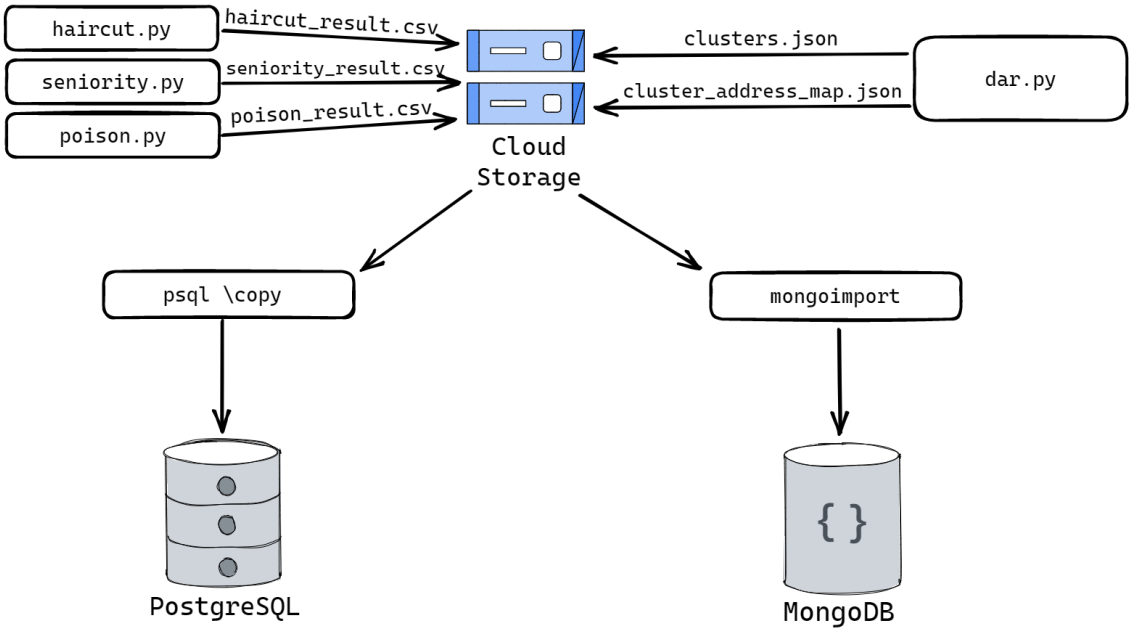

Figure 5.3: *Illustration of how the results reach the databases (the DAR results also go through a number of post-processing steps not illustrated here).*

After the algorithms had been executed, the results were stored in multiple locations. The raw CSV and JSON result files were first stored in a Google Cloud Storage bucket for easy access. As illustrated in figure 5.3, the results were also stored in a *PostgreSQL* database. This database was used to store the results of the blacklisting algorithms, while the results of the clustering algorithm were stored in a *MongoDB* database more suitable to the graph structure of the clustering results.

Storing the massive amount of data produced by the blacklisting algorithms in a relational database was not a trivial task. Inserting the hundreds of millions of rows produced by the algorithms into a fresh *PostgreSQL* instance would have taken days, and any change to the algorithms would have required the entire database to be dropped and recreated with the new results. To avoid this, optimisations were made to the insertion process. The optimisations made to the seniority table can be seen in figure 5.4.

```
1  -- EXAMPLE IMPORT OF SENIORITY RESULTS
2
3  -- SET UNLOGGED AND DROP INDEX
4  ALTER TABLE seniority SET UNLOGGED;
5  DROP INDEX ix_seniority_address;
6  -- PostgreSQL \copy command
7  \copy seniority FROM '/data/blacklist/seniority/seniority-tornado-
       result.csv' delimiter ',' csv header;
8  -- ENABLE LOGGING AGAIN AND IF NEEDED RECREATE INDEX
9  ALTER TABLE seniority SET LOGGED;
10 -- RECREATE INDEX HERE IF NEEDED
```

Figure 5.4: *PostgreSQL code snippet used to heavily improve insertion speed into tables.*

As seen in the code snippet in figure 5.4, the first thing to do before importing is to disable logging and drop any existing indexes on the table. Setting a *PostgreSQL* table to unlogged will disable writes to the *write-ahead log* (WAL). Normally in *PostgreSQL*, all changes to a table will first be described in a log file before being committed. By disabling logging, these writes to the WAL will not happen, which results in a significant performance boost. In this case, the `\copy` ran approximately twice as fast `UNLOGGED` versus `LOGGED`.

However, the most important optimisation for our use case was to disable all existing indexes. No test was conducted to find out exactly how much faster the insertion became. This was because inserting hundreds of millions of rows into a table with an index enabled would have taken too long and was therefore never tested except for one time where it was cancelled after close to 18 hours.

Disk speed was also found to be an important factor when importing large amounts of rows into the database. The *PostgreSQL* instance was therefore hosted locally on an M.2 SSD. Initially, an HDD with 220 MB/s read and write speed was tested, but this was found to be too slow. Even after performing the optimisations described above, the HDD insertion process had not finished after 12 hours, at which point it was cancelled. The M.2 SSD, with read and write speeds of 3480 MB/s and 2700 MB/s, respectively, was able to import the same data in less than 3 hours.

The clustering output was stored in two collections. The first collection, `clusters`, contains the cluster graphs themselves. Each cluster, in turn, contains the addresses found inside the cluster, as well as the edges that represent the relations between them. Each cluster also has an associated key. The second collection, `cluster_address_map`, contains the set of all addresses found within any of the clusters. These are paired with a corresponding key that points to the cluster in the first collection where the address was found. So, to find an address's cluster, one first makes a lookup on the address in the second collection, takes the key, and uses it to make a lookup in the first collection to find its corresponding cluster.

PyMongo, the Python library used for all other interactions with MongoDB from code, could, due to performance reasons, not be used to insert the clusters, as it quickly used all 32 GB of available RAM. *Mongoimport*, a database tool provided by MongoDB, had to be used instead. The mongoimport tool was executed with the `mongoimport -jsonArray` option.

5.5 API

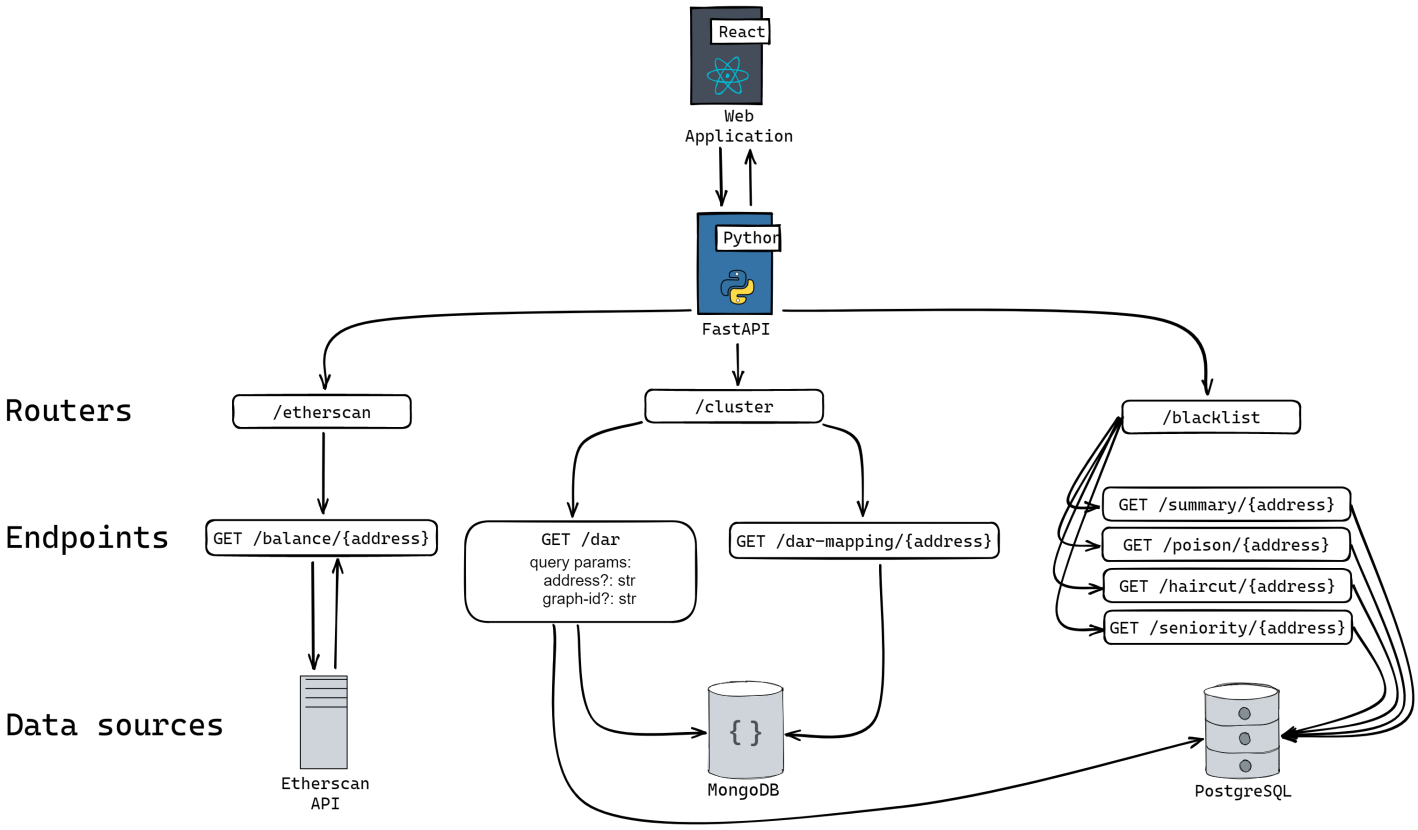

Figure 5.5: *Architecture of the API, including the FastAPI routers, endpoints, and data sources.*

In order to compile and serve the results of the algorithms to the front-end application, a web API was built. This API was constructed using the Python framework *FastAPI*, which is a modern and lightweight web framework for building APIs. The way in which the API is utilised to serve the results to the frontend is illustrated in figure 5.5.

The most crucial role of the API is to compile the results of the different algorithms and draw conclusions from the combined results. One example of a simple conclusion that can only be drawn from the combined results is whether an address is considered to be completely clean. This is when no taint is detected on any of the algorithms for a given address. This is also where the results from the clustering and blacklisting algorithms are combined in order to find out whether any of the addresses detected by the DAR algorithm contains illicit funds. If one of the addresses belonging to a

cluster contains an illicit address, all of the other addresses in that cluster could be marked as suspicious. In other words, the combined results allow the application to move from an address level to an entity level.

The API endpoints are divided into three categories: blacklisting, clustering, and *etherscan*. As seen in Figure 5.5, they each have their own router class containing the endpoints and database connection logic. The clustering router is connected to the *MongoDB* database, while the blacklisting router is connected to the *PostgreSQL* database. In the case of the *etherscan* router, its only purpose is to communicate with the *Etherscan API* in order to fetch the current balance of an address.

Each blacklisting endpoint accepts an `address` argument, which is the address that will be looked up in the databases. The blacklisting endpoints allow you to either look up the address in the results of a particular algorithm or to receive a summary of the address' status in all blacklisting algorithms.

Clustering data is consumed primarily through the `/dar` endpoint. This endpoint allows you to get a DAR cluster based on either the address of a node in the cluster or on the cluster's ID/key. A cluster's ID can be obtained through the `/dar-mapping` endpoint, which takes an address as input and if that address exists in the `cluster_address_map` collection it will return the ID of the cluster within which it can be found. As can be seen in figure 5.5, the `dar` endpoint also uses blacklisting data from the *PostgreSQL* database. This blacklisting data is used to look up each address in a cluster. If it is decided to be a tainted address, a flag will be set next to that address in the response from the API.

The primary reason for creating a separate Etherscan router was to allow for future use of additional data from the *Etherscan API*, e.g. transaction history or token holdings for a particular address.

5.6 Front-end

The front end was developed as a *single-page application* (SPA) with React. The application was made to be very simple with only a search bar and a couple of buttons. If an address is entered into the search field, the application will fetch the results from the API and display them in a separate view.

This simple design was chosen because it would allow for a quick and easy way to demonstrate the results of the algorithms. It was also kept simple because the primary focus of the project was on the algorithms and data, not on building a complex application with a lot of functionality. The frontend was therefore developed in a very short time and was not given much attention in terms of design or user experience.

The design was constructed using two larger React components: the search view, `Home.tsx`, and the view for displaying all of the results, `AccountView.tsx`. These

two components are, in turn, composed of several smaller React components, such as the search bar, the loading spinner, and the algorithm results. The components are populated with data through calls to the API, as illustrated in figure 5.5.

There was one more complex component that had to be built, the `Graph.tsx` component. This component is responsible for drawing the graph of a DAR cluster. It had to be able to draw an interactive graph possibly with a large number of nodes and edges, and it also had to be able to do so in a way that is performant and does not cause the browser to freeze.

To achieve this, the graph component was created with the help of the *ReactFlow* library. *ReactFlow* is a library for drawing interactive graphs in React. It is built on top of the *D3* library. *D3* is a library for drawing interactive graphs in JavaScript. It is very powerful, but it is also very low-level. *ReactFlow* is a wrapper around *D3* that makes it easier to use in React. It also provides a number of useful features, such as the ability to pan and zoom the graph.

5.7 Analysis

In order to analyse the results of the algorithms, Jupyter notebooks were initially used to explore the data, test different hypotheses and create visualisations. The notebooks accessed the results of the algorithms through a combination of raw CSV files, direct database connections, and API calls.

Well-known Python libraries such as *pandas*, *numpy*, *seaborn*, and *matplotlib* were used to analyse the data and produce the graphs displayed in this thesis.

The notebooks were later converted into Python scripts that could be run from the command line. This was done in order to make the scripts more portable and easier to run on cloud instances. Running the scripts on cloud instances was done in order to bypass the RAM limitations of the available physical machines. The scripts were also converted into Python scripts in order to make them easier to run in parallel, which was not a necessity, but is a nice-to-have feature.

6

Results

In this section, the results of both the developed tool and the implemented algorithms will be shown. First, the application will be displayed and explained, then there will be several graphs and charts showing the results of the different algorithms.

6.1 The application

The application was implemented as a *single-page application*. The front end was hosted using *Firebase* on a public domain, while the *PostgreSQL* and *MongoDB* databases, along with the API, were self-hosted with the help of *Docker Compose*.

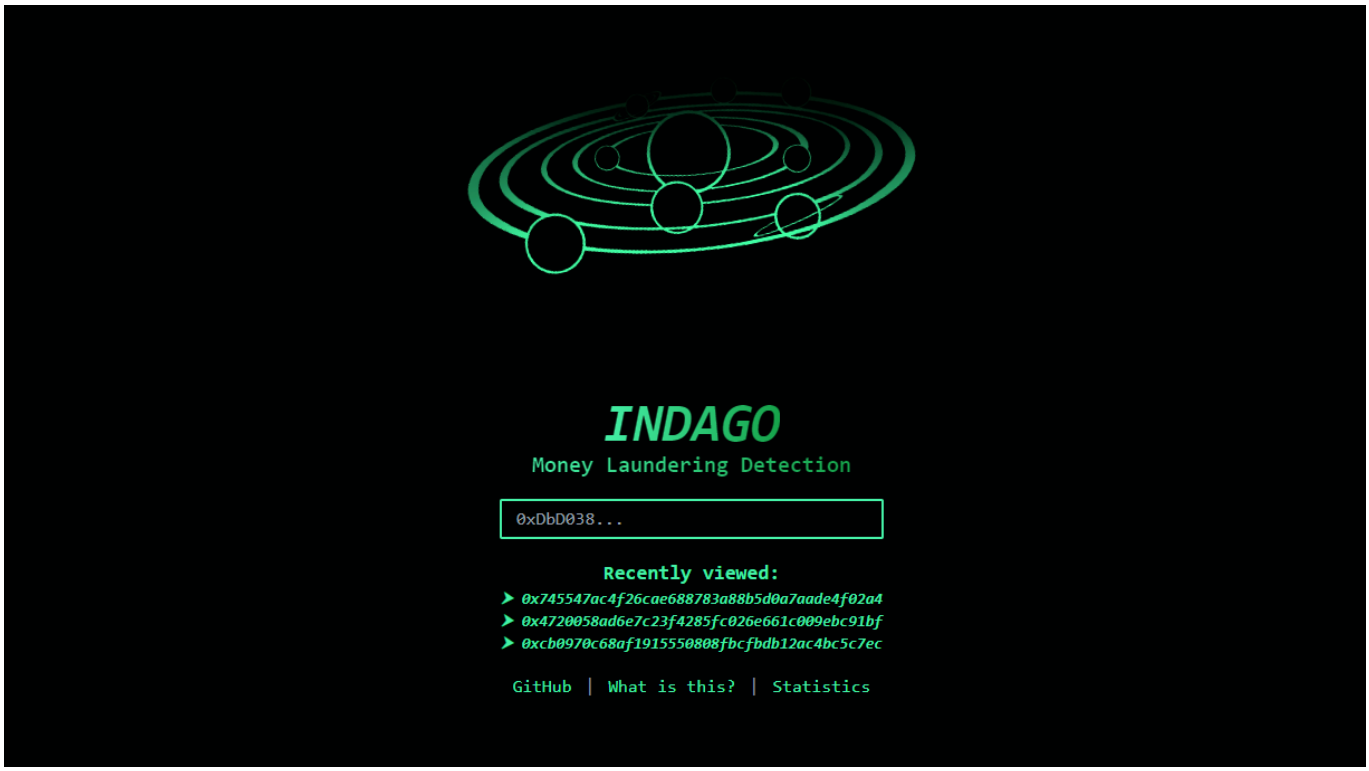

Figure 6.1: *The finished version of the developed application's search screen.*

As can be seen in Figure 6.1 the application was made with a simplistic two-tone

look. The design was inspired by one of the main adversaries, namely Tornado cash [109]. The colour scheme is very similar, although the purpose of the application is the opposite. Since the search bar is the access point through which the user accesses the main feature of the application, it is placed in the centre for easy access. The remaining buttons go to our git repository, an animated view for some of the results shown below, as well as a tribute button to the tool used for developing the website.

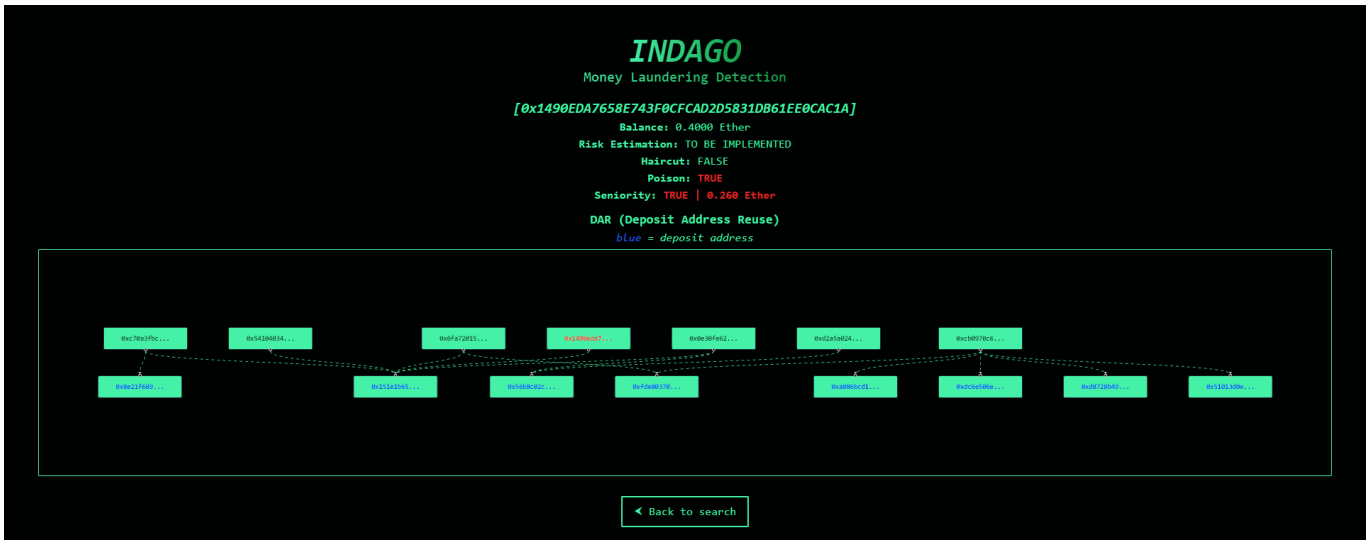

Figure 6.2: *The final version of the details view that is seen after searching for an address in the application.*

In Figure 6.2 the search result view can be seen. See Figures 6.3 and 6.4 for a closer look. First, the address is displayed, under which the balance of the address can be seen. Then follows the three working blacklisting algorithms and their respective tracking results. The red text indicates that the address contains tainted funds. For Poison, only a true or false reading will be displayed, while for the other two algorithms, there will also be a value indicating the portion of tainted funds for haircut and the number of total tainted funds for seniority. Finally, at the bottom, there is also the clustering result view, which appears only if the address is contained within a cluster. The rest of the cluster will also be shown, with links to the other addresses so that the user can easily jump between them.

6.2 Algorithm results

During the execution of the blacklisting algorithms, three metrics were measured in order to get a good idea of how the algorithms would behave over time. These metrics were: the number of addresses blacklisted over time, the time to reach any given block, and the amount of *random-access memory* (RAM) used over time. Data for each of the metrics were collected at intervals of 10,000 blocks. Each algorithm was tested twice, first with the original dataset obtained from *Kaggle* [103], and then with the addresses used by Tornado Cash's smart contracts. The original dataset contains 5,224 addresses, while Tornado Cash only uses 20 addresses for its operations. The

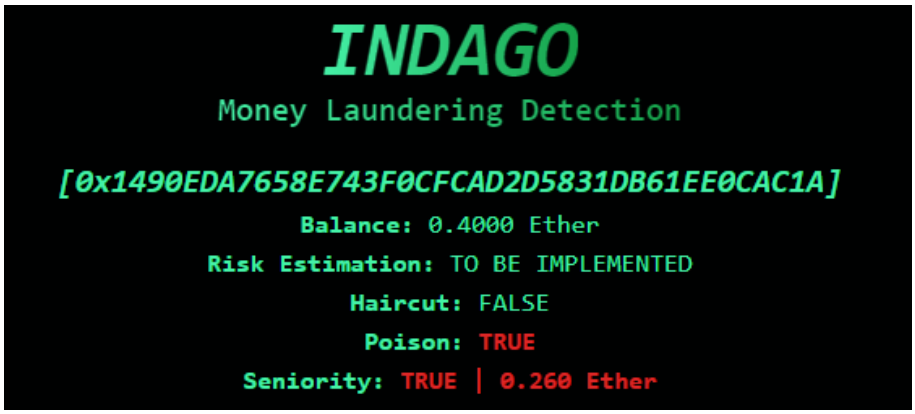

Figure 6.3: *A zoomed-in view of the blacklisting results that are displayed in Figure 6.2.*

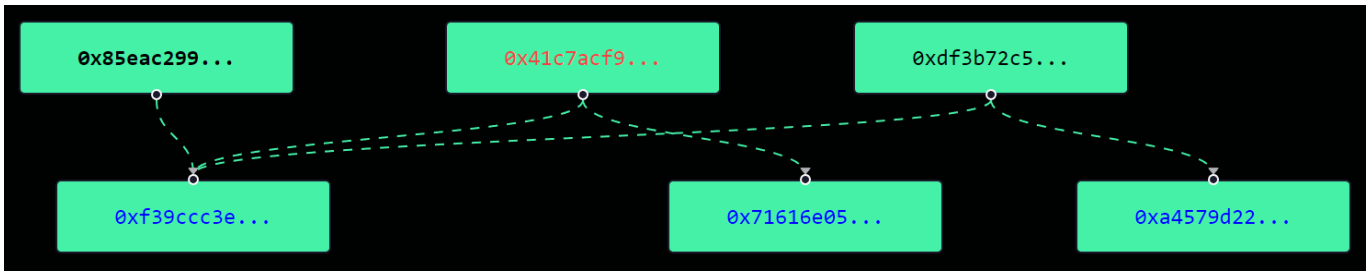

Figure 6.4: *A zoomed-in view of the cluster graph that is displayed in Figure 6.2, but with a smaller cluster.*

number of blacklisted addresses and the time taken to reach each block were both recorded from within the blacklisting scripts, while the RAM usage was recorded by calling `psutil`. The resulting graphs can be seen in Figure 6.5, Figure 6.6, and Figure 6.7.

Sadly, none of the implemented FIFO versions was efficient enough to be able to compute a value for the entire blockchain history. Therefore, all FIFO results will be left out of the results.

It is hard to grasp how many addresses were blacklisted by the algorithms. In Table 6.1 we display the number of blacklisted addresses for the different algorithms in relation to the total number of addresses on Ethereum. As we can see the number of blacklisted addresses makes up a large portion of all addresses on the blockchain, especially poison and haircut, who both blacklist more than half of all addresses on the blockchain. These numbers are for the Kaggle dataset, which means that it has had a lot of time to spread the taint throughout the blockchain. In Table 6.2 we chose to show the number of blacklisted addresses at different taint thresholds for the haircut and seniority blacklisting algorithms. Here, it becomes very clear that haircut defuses the taint to a higher degree than seniority. To note is that the vast majority of tainted addresses for haircut does not even reach above the first threshold.

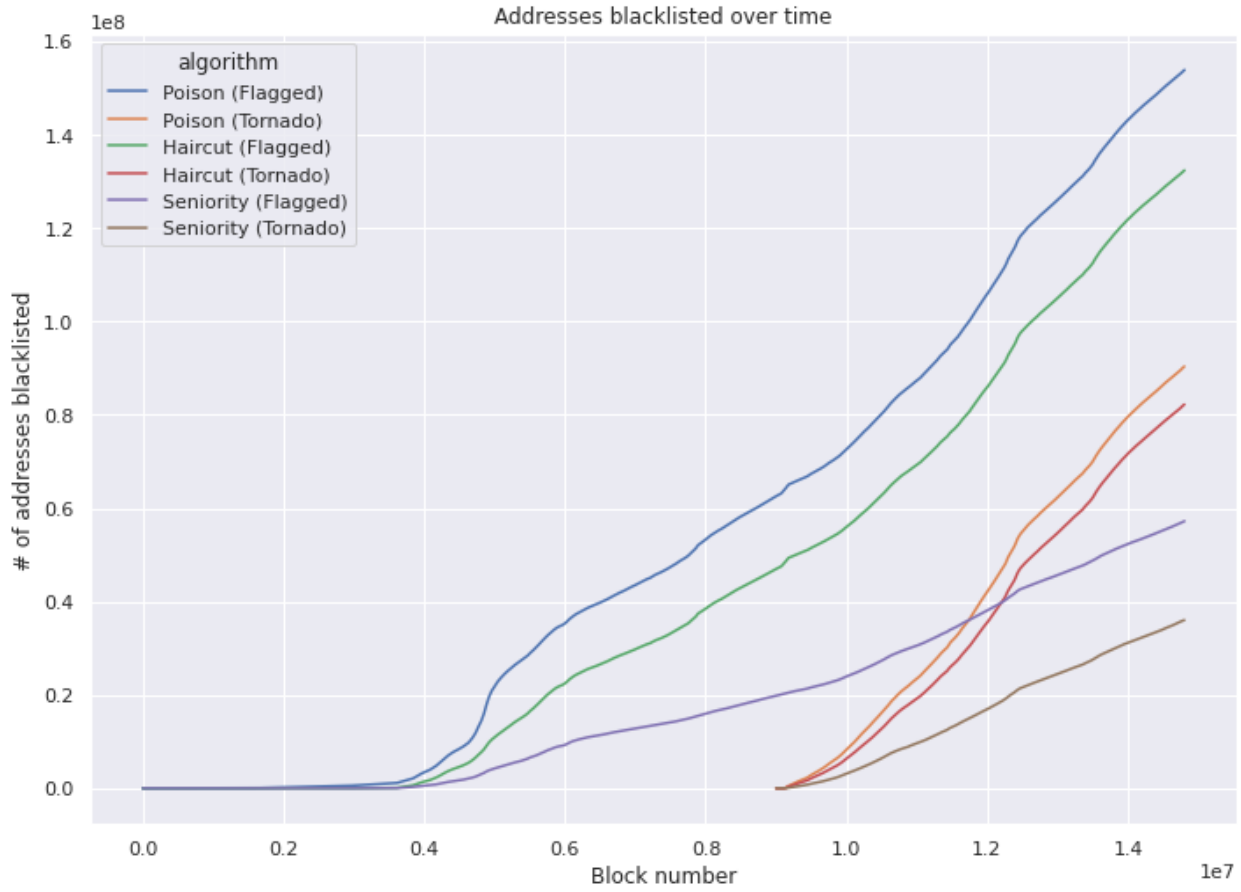

Figure 6.5: *The total number of addresses blacklisted at each block for the different blacklisting algorithms.*

Clustering with the DAR algorithm was performed on all Ethereum transactions, taking approximately 12 hours to complete. In Table 6.3 a number of clustering statistics are presented. As can be seen, close to a million unique clusters were detected containing more than five million unique addresses.

Every address is either a *user* or *deposit* address. Based on the number of addresses classified as *user addresses*, it can be concluded that the majority of the detected addresses are *deposit addresses*.

The size of a cluster can vary greatly as shown by the *min* and *max* sizes. By the μ and median values, as well as in Figure 6.8, it is clear that an overwhelming majority of the clusters are small, with more than 3/4 of the clusters having a $size < 6$. Only looking at Figure 6.8 can make it seem as if almost every detected address resides within a small cluster, while from Figure 6.9 it is apparent that the larger clusters also contain a decent number of the detected addresses. It is important to remember that there is a clear distinction between the number of clusters of a certain size and the number of addresses that reside within clusters of a certain size.

As can be seen in Figure 6.10, the distribution of flagged versus clean addresses

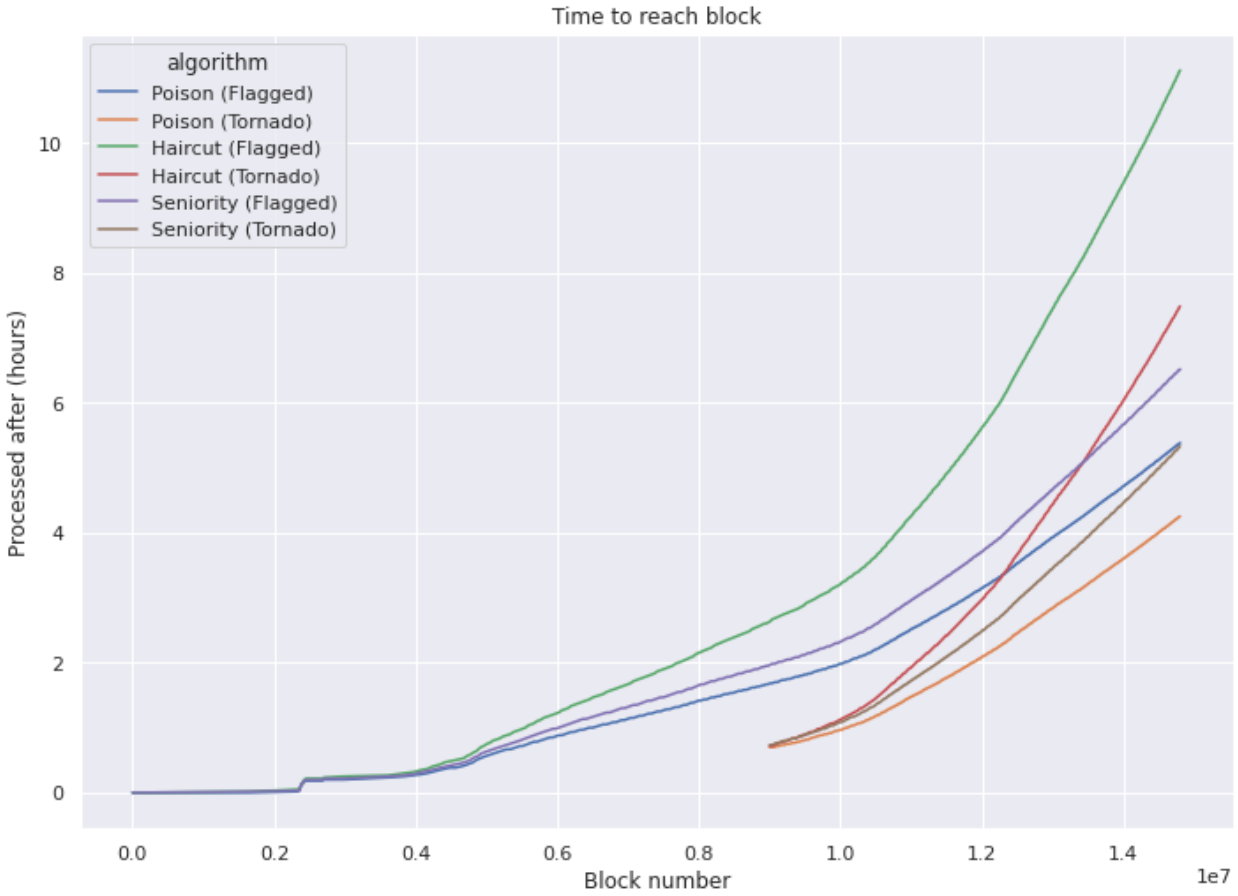

Figure 6.6: *The total time taken to reach each block for the different blacklisting algorithms.*

are close to equal within flagged clusters. This indicates that the combination of blacklisting with clustering helps identify addresses that otherwise would not have been flagged.

The number of new addresses that could be flagged by combining blacklisting and clustering is relatively small. This is because only around 1% of all addresses flagged by the *Seniority* algorithm were located within a cluster, as seen in Figure 6.11.

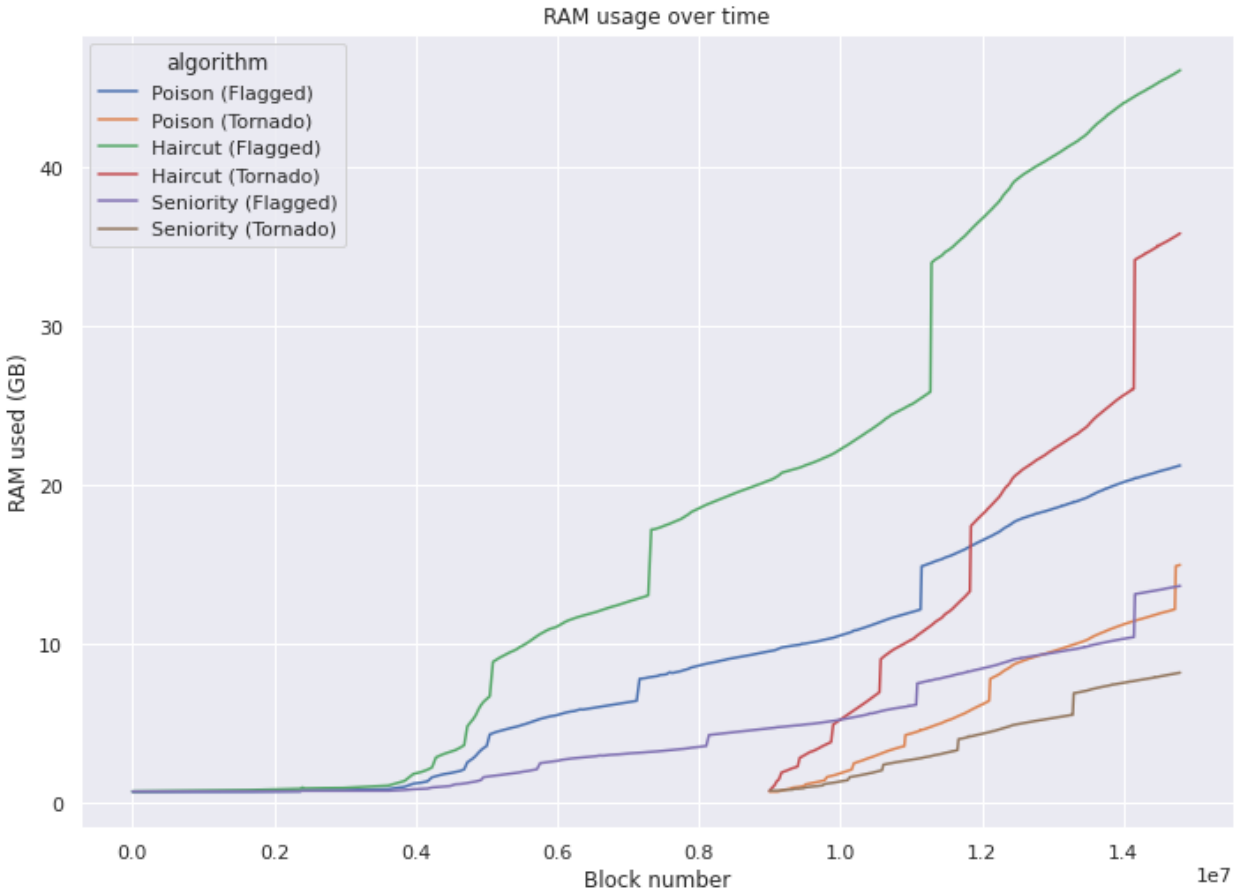

Figure 6.7: *The amount of RAM used by the different blacklisting algorithms.*

Algorithm	Poison	Haircut	Seniority
blacklisted addresses	153,899,138	132,386,234	57,218,543
% of all addresses	78%	67%	29%

Table 6.1: *Shows the number of blacklisted addresses for each of the blacklisting algorithms. Also shows the number of blacklisted addresses as a portion of all Ethereum addresses.*

Algorithm	Haircut	Seniority
$>$ 0.001 ether	9,319,487	42,769,478
$>$ 0.01 ether	2,438,833	18,859,847
$>$ 0.1 ether	677,374	7,264,718
$>$ 1 ether	74,918	2,485,071
$>$ 10 ether	11,481	381,589
$>$ 100 ether	5,976	28,946
$>$ 1000 ether	5,318	7,739

Table 6.2: *Shows the number of blacklisted addresses at different taint thresholds for the haircut and seniority blacklisting algorithms.*

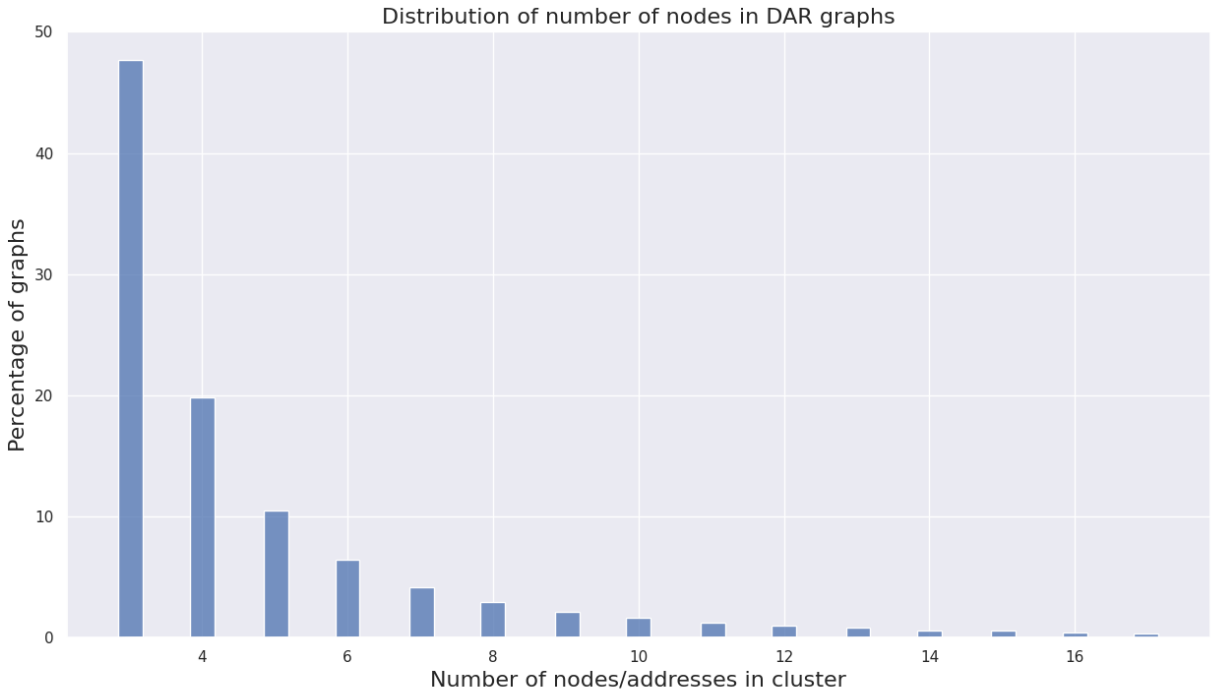

Figure 6.8: *Illustrates the distribution of cluster sizes. There are many clusters that are larger than 17, but they make up such a small percentage that they would barely be visible in the table.*

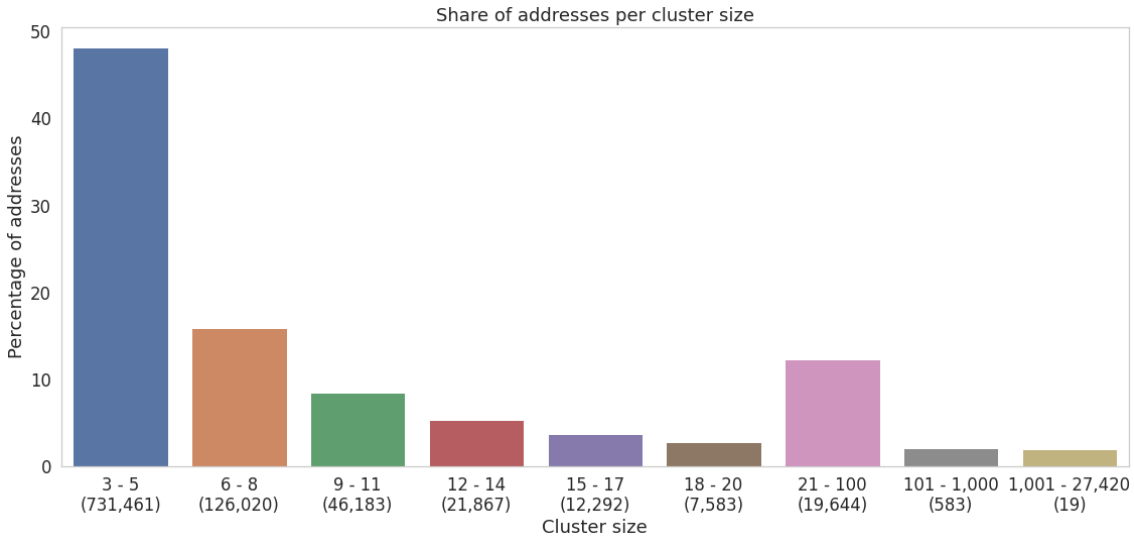

Figure 6.9: *Displays how the detected addresses are spread out between different cluster sizes. The number in parentheses below each x-label is how many clusters are within the range, while the height of each bin represents the total number of addresses within those clusters. Note that the spike that can be seen for addresses in the range 21-100 is due to a jump in the size of the range, and not a sudden increase of addresses.*

Algo.	Total clusters	Uniq. addresses	User addresses	μ (size)	Median (size)	min (size)	max (size)
DAR	965,719	5,361,931	2,447,933	5.6	4.0	3	27,400

Table 6.3: *A collection of statistics related to the clustering results.*

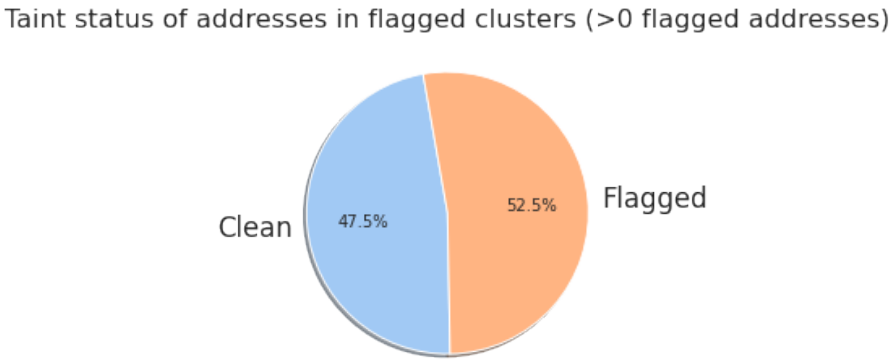

Figure 6.10: *Within clusters containing at least one flagged address, what is the percentage of clean versus flagged addresses.*

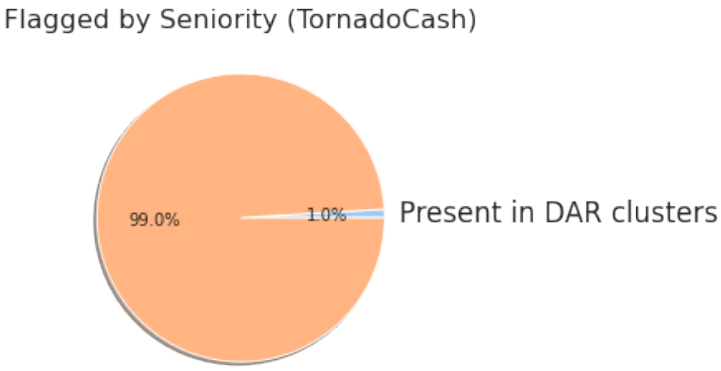

Figure 6.11: *The percentage of the addresses blacklisted by seniority that can be found within a DAR cluster.*

7

Discussion

In this section, the results will be discussed in order to arrive at a conclusion about the performance of the tool and what the result could mean for financial institutions and private individuals trying to avoid illicit assets on Ethereum. First, there will be an individual evaluation of each of the implemented AML strategies. After this, there will be a brief discussion on the generalisability of the results, followed by a discussion about what illicit actors might do to adapt if the tool was put in use. Then we briefly mention Tornado Cash before discussing the cryptocurrency field in relation to other software engineering fields. Finally, there will be a future work section followed by a conclusion.

7.1 Strategy evaluation

Two main strategies, blacklisting and clustering, were developed for the tool. In this section, they will be evaluated both individually and in combination. Since the blacklisting algorithms ended up as the main feature of the tool, each blacklisting algorithm will also be evaluated separately.

7.1.1 Clustering

Clustering could be done in many different ways. For this tool, the main method of clustering was to use deposit address reuse in order to detect exchanges or other entities that reuse addresses when gathering funds to one central address. The use of this strategy has been successful to the extent that it was able to generate a lot of clusters, though the inherent value of these as a way of tracking illicit funds is not very apparent, at least not as standalone data.

As can be seen in Figure 6.8 most of the clusters were small. The number of clusters with more than 10 addresses is less than 8%. This seems logical since most entities moving cryptocurrencies are probably private individuals who invest small amounts and who have only moved currency between exchanges a handful of times, if ever.

Although most clusters are small, there is still a sizable portion of addresses in larger clusters, as can be seen in Figure 6.9. These larger clusters are possibly more

sophisticated entities that are moving funds with more elaborate schemes. This could be day traders taking advantage of disparities in price between exchanges, or hedge fund level investors who have to distribute purchases and sales between several exchanges in order to find enough liquidity. It would be interesting to compare the size of transactions for different cluster sizes in order to see if the larger clusters tend to transfer with larger amounts. This would certainly support the hypothesis that the larger clusters mostly belong to single large entities that transact at a different magnitude when compared to the average private investor.

In regards to tracking illicit funds, the clusters do not have much of a purpose, since they do not provide any direct information about the entities who are in control of the clusters, or at least not when taken in isolation. What they do bring is the ability to amplify the results of other methods. As can be seen in figure 6.11, about 1% of the addresses blacklisted by Seniority turn up in the clusters created by DAR. This might not sound like much, but it still turns out to be roughly 342,981 addresses inside 238,536 unique clusters. What this means is that the remaining 310,491 addresses in the found clusters can be added to the blacklist. This number would probably be much larger if clustering was performed incrementally at different points in time during the original seniority blacklisting.

These extra addresses could also be used for risk assessment when a VASP is asked to accept funds from a customer. Having addresses in the same cluster as other blacklisted addresses might not be enough to bar a customer at the door, but if they also score high on the VASP's individual risk assessment, the combination of the two could be enough to sound the alarm. If such a user was found guilty of money laundering at a later date, this could save the VASP from having to pay large fines, as we have seen in [4].

7.1.2 Poison

As the first and simplest blacklisting algorithm in this study, poison was never expected to perform very well. It is very aggressive when adding addresses to the blacklist, which makes it largely unusable. Poison has one crucial flaw, which is that it does not pay attention to the number of tainted funds sent between addresses. If an illicit actor sends only a small fraction of a tainted ether to an exchange address, all of the cryptocurrencies entering and leaving the exchange will, from then on, be considered 100% tainted. Poison could possibly be used as a simple indicator that should prompt further investigation, but should otherwise not be used to make hard decisions on whether or not to accept cryptocurrency funds.

With that said, there are a few plausible applications of poison where its speed lends it an upper hand. Since it is the fastest blacklisting algorithm out of the ones tested in this thesis, it could be used in situations where only a small amount of computing power is available. If the choice is between having poison blacklisting and having no blacklisting at all, poison is arguably the better choice.

It could also be used to track from a single point of taint in order to, as quickly as possible, reach all potentially tainted addresses. Imagine if law enforcement just discovered that a transaction that occurred on the blockchain a few months ago was involved in the purchase of a large amount of illegal drugs. The police may not have time to wait for a complex FIFO-based tracking algorithm to take days to finish computing. An arrest might only be possible within the next 24 hours. In such a situation, the faster poison algorithm might be able to provide enough information to enable an arrest before the window of opportunity closes.

Since it has proven difficult to perform tracking on more than one thread, it is conceivable that several tracking algorithms could be used at the same time without decreasing the performance of the main tracking algorithm. In such a situation, poison could be used as a separate pre-computation, meant to give a faster partial result that could be used while the full result is still computing. In the example above, law enforcement could use poison to gather preliminary data in order to secure enough information to warrant an arrest, while a more advanced algorithm could be used in the following court case.

7.1.3 Haircut

Haircut blacklisted about the same number of addresses as poison but otherwise performed very differently. The key difference between the two is in the degree to which they blacklist an individual address. Haircut is more nuanced than poison since it keeps track of the portion of funds on an account that is blacklisted. As can be seen in Table 6.2 the vast majority of addresses blacklisted by haircut only have a few fractions of a tainted ether. It seems logical that such an address should be treated very differently from an address containing thousands of tainted ether. Without the more fine-grained control that haircut offers, these two addresses would be treated with the same level of caution, even though one is clearly worse.

Another issue with poison that haircut solves is that of trying to keep the amount of taint on the blockchain constant over time. Poison has a tendency to exponentially grow the number of tainted funds as time goes on. It is inevitable that tainted funds are mixed with other funds on the blockchain. When this happens, poison will label all of the untainted funds as tainted. As this is repeated over time and taint spreads to a growing number of addresses, it gradually increases the total amount of taint on the blockchain. This could eventually lead to a significant portion of all ether being blacklisted. When a large or even dominant portion of the currency on a blockchain is blacklisted, the usefulness of the blacklist disappears. Haircut solves this issue by diluting the taint whenever clean and tainted funds are mixed. Even though haircut blacklists almost as many addresses as poison, most of those addresses only have a fraction of their contents blacklisted.

Keeping the number of blacklisted funds constant over time is especially important with Ethereum since it does not have a built-in way to reject incoming transactions. Anyone who knows about the blacklist and is in possession of blacklisted funds will

be able to spread the taint to any address on the blockchain. This makes haircuts much more realistic for real-world applications since it takes away the negative effect that tainted funds might have if they land in an otherwise clean account. It makes sure that the clean portion of the ether on an address stays the same, no matter how many tainted funds are added to the address. Assuming that users are not banned from interacting with VASPs when their wallet contains only small amounts of taint, haircut will make sure that their clean coins are protected from actors with malicious intent.

In fact, it is crucial for VASPs to allow a small amount of taint if they decide to use haircut as their main tracking strategy. As can be seen in Figure 6.5, the number of tainted addresses is very large, almost as large as poison, which means that many of these addresses have only a small amount of tainted funds. It seems likely that the vast majority of Ethereum users are innocent and do not know that they have trace amounts of tainted funds in their accounts. If a VASP were to go with a zero-taint policy while using haircut, a large portion of all Ethereum addresses would not be able to interact with them despite their innocence. This would also shut out a large portion of potential customers, which could become a significant financial burden.

Blocking users for having only small portions of taint would also reintroduce one of the largest issues with poison. If VASPs block transactions that contain even trace amounts of taint, they would once again make it possible for malicious actors to sabotage other users by sending them tainted funds. If an account is infected with just a small amount of tainted funds, that taint is close to impossible to get rid of. Since there is no way to separate clean and tainted funds, the only way to decrease the level of taint on an address would be to add more clean funds. This would reduce the portion of taint, but it would also introduce taint to funds that were previously clean. Since anyone can transfer to anyone, it would be impossible to enforce a zero-taint policy without having this potential for sabotage, leaving a hole in security that cannot be patched. Because of this, any financial institution that intends to adopt haircut for tracking on Ethereum will have to accept at least a small amount of taint. To discourage money laundering while receiving tainted funds, the tainted portion of a transaction should be ignored when performing valuations before sales involving tainted ether.

7.1.4 Seniority

Seniority has arguably performed best out of the algorithms tested. It has considerably decreased the number of blacklisted addresses compared to haircut. This is beneficial since it also means that the concentration of taint on the remaining addresses is higher. This is very apparent in Table 6.2 where seniority has considerably higher concentrations of taint than haircut. If a VASP wants to block all tainted addresses, it means that seniority will require fewer blocked addresses while still blocking the same amount of taint, when compared to haircut.

Seniority also outperforms haircut in another aspect, which is time. As can be

seen in Figure 6.6, seniority computes its blacklist in less than half the time of haircut. It is even close to beating poison, despite poison being considerably simpler in its implementation. This gain in speed we believe is because of the relatively small amount of memory seniority used compared to the other algorithms, as can be seen in Figure 6.7. Since all of the tainted funds are transferred first, most of the transactions going out of any given tainted address are either 100% tainted or 100% clean. Haircut, in contrast, makes sure that each transaction gets the exact same portion of taint, which leads to maximum dilution. Due to the way seniority is implemented, it does not have to keep track of the balances on addresses that do not have tainted ether. Since seniority dilutes its taint so little, it has a lot of addresses and transactions it can discard, which in turn leads to large reductions in required memory.

Apart from the apparent benefit of using less memory, it also has the knock-on effect of increasing the speed of the algorithm. Since there is a smaller number of addresses stored, there is also a smaller number of addresses that have to be sorted through every time the taint value of an address has to be updated. Even though dictionaries were used for storing the data, there is still a small time increase for lookups as the dataset gets larger [110]. Since the computations used by the algorithm are fairly simple, and the number of data entries is so large, the looking-up operation could potentially take more time than the actual computations.

Halving the computing time might not mean much when the difference is in a few hours. But if cryptocurrencies increase in popularity and the rate of new transactions increases, the amount of saved time might go from hours to days. Add on top of this the large amounts of extra data that would have to be stored to comply with, for example, the travel rule, as well as the 75% reduction in required memory, and it might be enough for seniority to be favoured above haircut.

The largest issue with seniority though is the lack of motivation behind having blacklisted coins leaving an address first. It seems almost as logical to have them leave last. Both poison and haircut are uniform in the way they spread taint. This could be perceived as more fair since it gives everyone the same opportunity. If only a limited number of users were fully aware of the blacklist rules, these users would have the upper hand if the rules made the spread of taint uneven. They would be able to pick and choose when to interact with other users or smart contracts in order to only get hold of clean funds. Validators could perhaps even shift the ordering of transactions on the blockchain, changing who gets taint and who does not, in order to benefit themselves or others. It is harder to game the system with a fair spread of taint.

7.1.5 FIFO

Before implementation, FIFO seemed to be the most promising blacklisting algorithm out of the ones tested in this thesis. Even though it did not have a fair distribution like poison or haircut, we believed that the queueing of transactions would be

easier to motivate than the arbitrary ordering of transactions in seniority. It would theoretically also preserve the concentration of taint better than haircut, while also keeping the total amount of taint stable, unlike poison. The most exciting part though was that using a special implementation of FIFO, called advanced FIFO, we were able to preserve the origin of tainted funds by labelling each item in the queue with the address from which it had originated. This would have been a large step towards providing the KYC information that would be required if legislation such as the travel rule were to be required for transactions going across the blockchain.

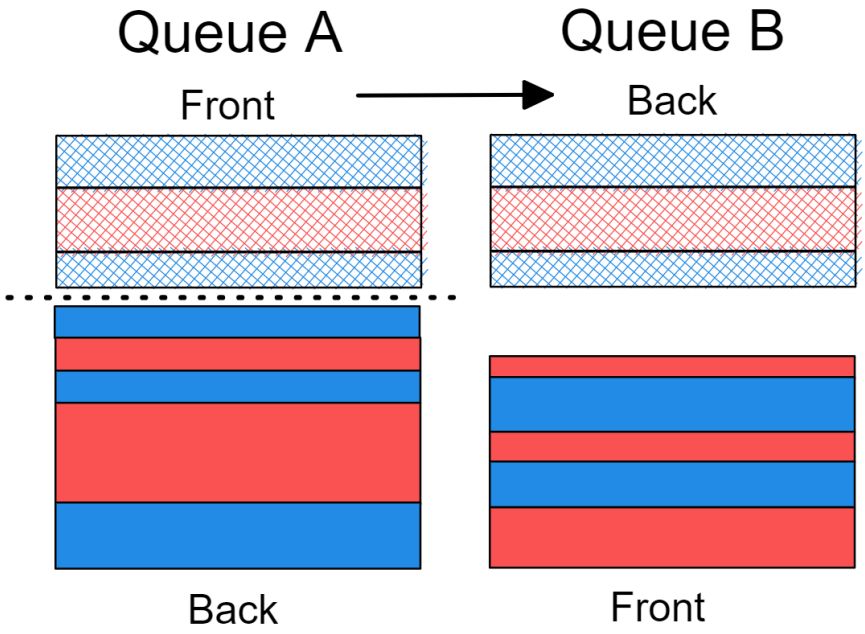

Figure 7.1: *Illustrates the fragmentation that could happen when a transaction is performed from A to B while tracking using the FIFO blacklisting algorithm. Here red is used to show tainted fragments while blue is used to show untainted ones.*

Why none of the FIFO algorithms worked is not fully investigated at this point. We theorise that it has to do with a growing level of fragmentation within each address. Each blacklisted address on FIFO has a queue containing fragments of either entirely tainted or untainted ether. We believe that the slowdown is caused by the number of entries in these queues growing over time. If there is a transaction going from A to B, it is likely that the transferred amount does not equal an exact number of queued items contained in A. This will lead to one new fragment being created as one of the entries in A has to be split in two in order to transact the correct amount to B. See Figure 7.1 for an illustration of this phenomenon. If this is repeated over time, it will lead to a growing number of fragments in the blacklist. Since each fragment takes up the same amount of storage, no matter its value, there will be an increase in storage need as the number of fragments increases. This will in turn increase the number of computations needed, as each transaction will involve more time-consuming look-up operations and involve the transfer of a larger number of isolated fragments.

First, we believed that the creation of new fragments would be balanced out by an equal number of fragments being merged. If two tainted or two untainted

fragments land next to each other in the queue, they can be merged without losing information. The base implementation of FIFO does this with both tainted and untainted fragments, while the advanced version only does it for untainted ones. This we thought would counteract the fragmentation, but we now realise that it will not happen at the same rate as new fragments are created. In the best-case scenario, when both tainted and untainted fragments are merged, this will happen on average only for every other transaction involving tainted addresses. This is because the ordering inside a sequence of fragments that are being moved will be preserved. This means that the only two fragments that get a new neighbour will be the fragment last in the receiver queue and the fragment at the beginning of the sender queue. Out of the four possible combinations of these two fragments, only two will result in a merging of the fragments. Assuming that it is more common to transact a portion of the amount stored on an address than it is to transact all of it, this would create a new fragment for more than half of the transactions involving tainted addresses. This also assumes that the number of times a transaction is made, with a portion of the value stored on the address, where the boundary between two fragments is hit exactly, only happens on rare occasions.

It is possible that further optimisation of the FIFO algorithms could overcome this dip in performance due to fragmentation. All the other algorithms do, after all, also slow down over time, just not at the same rate. As it currently stands though, there is no more time to investigate the FIFO issue further. See section 7.6.5 for a plan on investigating the FIFO issue further.

7.2 Generalizability of results

Despite Ethereum being the second-largest cryptocurrency, and arguably being the most exciting, there is no guarantee that it will remain this way for the foreseeable future. There are many other cryptocurrency projects, heavily inspired by Ethereum, that have the potential to overtake it in the future. If this were to happen, would the results found in this report still apply to other cryptocurrencies? It is hard to say for sure, but due to the heavy inspiration that other projects take from Ethereum, we consider it likely that a potential successor would at least have a similar account and transaction structure. This would in turn mean that all of the algorithms developed here would also be applicable to this potential successor, given a few alterations to the implementation details.

It is also important to remember that even if Ethereum remains the largest Turing-complete cryptocurrency, there will still be other similar cryptocurrencies of significant size. There currently exist several other cryptocurrencies, such as Solana Cardano and Polkadot, that all have a similar account and transaction structure to Ethereum. Even if they never surpass Ethereum, there will still be a need to track illicit activities on these other cryptocurrencies. Although we are not as familiar with these other blockchains as we are with Ethereum, we are still confident that most, if not all, of the strategies developed in this thesis, will be applicable to these other cryptocurrencies.

7.3 How illicit actors might adapt to crypto-AML strategies

It is naive to think that illicit actors will not adapt their behaviour when new tracking tools are introduced. In this section, we will discuss two such adaptations and what the next move would be to counter such behaviour.

7.3.1 Laundering through untracked systems

The most obvious way to counter both the clustering and blacklisting algorithms is to move the funds to where there is no tracking. If there exists a zone in which tracking is not performed, and to which funds can be transferred without resistance, it would be the ideal place to launder illicit funds. By moving assets here, the chain of tracking would be broken since there would be no way to link together transactions entering and leaving the zone. Layer-two solutions, other cryptocurrencies, and Ethereum tokens are all examples of such zones. There is no mechanism in place for stopping funds from entering, and our tool is currently not able to track assets after they have entered.

There are two main reasons why funds cannot be stopped from entering these zones. The first is because they are all connected to Ethereum on the protocol level. Since they are connected directly to the protocol, there is no way for a centralised middleman to stop certain funds from entering or leaving. The second reason is that they are decentralised enough so that they cannot be taken down by anything less than a large international coalition. Tracking is not performed for these zones because the tool has not yet been equipped to do so.

We will discuss these zones further in the future work section. The important part here is that they are all places where funds can be laundered without being detected. The question is what institutions and individuals can do to counter this behaviour. Two obvious methods come to mind. The first is to extend the tool to provide tracking within these zones, and the second is to simply blacklist all the funds that leave the zones. Extending the tool is the ideal situation since it would eliminate the problem without compromises. If tracking can be performed everywhere, there would be no issues when following the flow of funds, and everyone could use every functionality on the blockchain without having to worry about being blacklisted. The issue with this approach is that it might require significant resources to implement a tracking solution for all of these areas, and it might not even be possible in some cases. Certain cryptocurrencies, such as Monero, will not be implemented in a way that makes tracking possible. Blacklisting everything would also not be an ideal solution, since it would block off a significant portion of Ethereum's functionality. In the end, the best way would probably be a hybrid of the two where tracking is implemented in as many places as possible, and the rest are either blacklisted outright or placed on a high alert list of some kind.

7.3.2 Switching to other cryptocurrencies

What if users did not only use other cryptocurrencies to avoid detection but instead completely switched, abandoning Ethereum for another cryptocurrency. This would, for the most part, not be a problem since it would mean that illicit actors are leaving the platform you are currently operating on. Less illicit funds floating around means there is a smaller chance that you or your organisation will acquire these funds by mistake. In such a situation, the most important thing to do would be to be more cautious of funds flowing in from other cryptocurrencies. If some cryptocurrencies have tracking and blacklisting, institutions are likely to only accept transfers from these cryptocurrencies. If there is no way for illicit actors to cash out on their preferred cryptocurrency, they have no choice but to try to funnel their funds over to more reputable cryptocurrencies and try to cash out there instead.

7.4 The size of Tornado Cash

As can be seen in Figure 6.5, the amount of tainted funds that originate from Tornado Cash is significant. It surprised us to find how many tainted funds could be created by a single crypto service. Although most of those funds are probably not illicit in nature, it still speaks to the impact that Tornado Cash had on the Ethereum ecosystem up until its recent sanction by the U.S. Department of the Treasury [111]. It will be interesting to see what the impact of Tornado Cash will be now that it is blocked by such a large portion of Ethereum's validators [112].

7.5 Software Engineering and cryptocurrencies

The last research question of this thesis was to investigate the difference between working with cryptocurrencies and working in other software engineering fields. Even though we only worked with second-hand information from Ethereum, and never developed something directly in conjunction with cryptocurrencies, we still feel we have gained enough experience to deliver some insights into what makes working with cryptocurrencies different.

7.5.1 Quality of information

One major difference is in the amount and quality of information available when browsing the Web. When working in other software engineering fields, there is usually an upstream of online material at hand that is capable of providing solutions to basically any problem. There are courses for you to learn the basics, articles for deeper knowledge, and Stack Overflow threads covering whatever obscure question you might find yourself asking. Access to this is heavily limited when trying to find information about cryptocurrencies.

There is a lot of buzz around cryptocurrencies, but almost all of it is focused on the money-making aspects. Because of this, most of the people searching for information

about cryptocurrencies are beginners with no prior knowledge of programming or software development. People who have no software engineering experience are bound to prefer short, simplified texts that skip out on the technical details. These basic descriptions are perhaps good enough for your average amateur investor, but they are no help for someone trying to develop software in close proximity to cryptocurrencies. It becomes increasingly frustrating when you find several articles covering the same subject, with none of them describing the topic beyond the most basic facts. Even prefixing search terms with phrases such as *"deep dive into"* or *"technical description of"* will still, most of the time, give the same simple descriptions, the only difference being that the title now claims that the article is comprehensive when it is not. On top of this, these articles also contain misleading or straight-up incorrect information to a higher degree than we are used to from other fields.

We expect this problem to disappear or at least decrease over time. It was only a few years ago that cryptocurrencies broke into the mainstream, and there are still not that many experts willing to share information, or developers creating a demand for that information. As cryptocurrencies gain more utility and the number of developers grows, we expect the available technical information covering cryptocurrency to increase.

7.5.2 Large sequential datasets

Another thing fairly unique to blockchain and cryptocurrencies is the large amount of sequential data that is involved. To begin with, the amount of data stored on blockchains is generally fairly large, ranging from a few hundred gigabytes up to tens of terabytes [113]–[115]. On top of this, there is usually also a lot of meta information that is not stored straight on the blockchain. Ethereum's blockchain, for example, currently stores about one terabyte of data, but a complete archival node, containing all previous states of the Ethereum virtual machine, will need close to ten times as much space [72]. This alone, though, would not make cryptocurrencies stand out in comparison to other Software Engineering fields. It is the high sequentiality of the blockchain data that makes it so different.

When transactions and smart contract calls are processed, they all happen in sequence. Each of these steps affects the state of the blockchain, and since every step can also be dependent on any of the previous steps, it is hard to divide the data so that it can be processed in parallel. It is still possible to analyse transactions without taking sequential ordering into account. Simple metrics, such as the number of calls to specific contracts, can still be calculated by analysing every trace individually and summarising the findings. But as soon as the analysis is dependent on the state of the blockchain, and not only on the individual traces, you are bound to run into trouble if you do not process the transactions in the correct order. This is especially apparent when trying to track the flows of funds on a blockchain. If one transaction is processed without the knowledge of a preceding transaction, there will be no data on the origin of those funds, which will break the chain of tracking. This leads to analysis having to be performed on a single thread, which severely limits the possible

complexity of the analysis.

As Baran et al. have shown [107], there are ways around this problem. Although it still significantly increases the complexity of any solution trying to draw conclusions about the flow of information through the blockchain.

7.6 Future work

The scope of this thesis was fairly ambitious, which means that there were many topics that could not be explored in the given time frame. In addition to this, there were also many questions that arose during the course of the thesis. These unexplored topics are listed below.

7.6.1 Tracking in real-time

The initial vision of the tool was to be able to track taint flowing between addresses in real-time. For this to be possible, the tool would have to receive data on new transactions that occur live on the blockchain. This proved to be more challenging than first thought. In order to receive reliable live data, the tool would have to be connected directly to an Ethereum node. Booting up and implementing a solution for streaming data directly from a node was deemed too labour-intensive and was therefore relegated to this future work section.

Real-time tracking is very important for the tool. The current implementation only tracks transactions that have been performed up to a certain point in time. Any moving of funds that has happened after this is invisible to the tool. Illicit actors can currently avoid detection simply by performing a single transaction to a new untainted address. Since the tool has yet to process that transaction, it will not know the origin of the funds, and it will therefore not be able to determine the level of taint on the account. Similarly, it will also not be able to connect that address to any cluster.

For the tool to be ready for real-world use, it will have to be connected to a live data source. If this is done, it will happen after the conclusion of this thesis.

7.6.2 Token support

The algorithms presented in this thesis only handle traditional ether transactions, but ether is not the only currency that can be transferred on the Ethereum blockchain. Ethereum has built-in functionality for creating an infinite number of *tokens*. These are alternative currencies that any Ethereum user can create. They function differently from ether in how they are stored and transferred, and thus require a different implementation for tracking reliably. There are smart contracts on Ethereum that make it possible to swap between ether and its diverse selection of tokens. This makes token tracking necessary to reliably track the flow of funds since illicit actors

would otherwise be able to avoid detection by swapping tainted assets for tokens and then back again in order to obtain untainted ether.

There are two different versions of tokens on Ethereum, one is fungible and the other is not. For fungible token transfers, data can be collected and treated the same as ordinary ether transactions, meaning that the algorithms utilised in this thesis should all be directly applicable given a few adjustments to the setup process.

Non-fungible tokens (NFTs) are, however, a lot trickier due to their uniqueness. There is no guarantee that two NFTs originating from the same smart contract have an equal, or even close to equal, monetary value. As long as all swaps between ether and non-fungible tokens are honest, the value of the ether in the transaction can be used to determine the amount of taint to transfer to the NFT. Issues arise when the value of the NFT is manipulated. This could happen when both the buyer and the seller of the NFT are the same person. In such a situation, the price can be adjusted either up or down depending on the desired outcome. An otherwise worthless NFT can be sold for a large amount of tainted ether, passing the taint over to the NFT. After this, the ether can be cashed out at an exchange and the NFT can be discarded. This manipulation means that any tracking involving NFTs would probably need a more sophisticated system capable of transmitting taint depending on a larger number of factors.

7.6.3 Tracking across cryptocurrencies

Another large problem that needs to be addressed is tracking the flow of funds moving directly between cryptocurrencies. There exist so-called *atomic swaps* that function as self-contained exchanges, making it possible to swap coins across blockchains in a decentralised manner. Since these are decentralised, there is no single entity to hold viable for letting tainted funds flow across cryptocurrency boundaries. Because of this, there is also no way to block specific funds from leaving the blockchain. Since these pathways are open to all, they will be especially attractive for people trying to launder stolen or otherwise tainted coins. As there is no way of stopping funds from leaving the blockchain, the only option for achieving complete coverage would be to extend the tracking across other blockchains. This would cause the required effort to grow rapidly since it would mean that a given tracking solution has to be implemented on every other blockchain connected to the Ethereum blockchain via atomic swaps. As time goes on and the number of available atomic swaps grows, the challenge of achieving complete tracking coverage will continue to increase.

7.6.4 Tracking across layer-two solutions

Due to the high transaction fees on Ethereum, the popularity of so-called *layer-two solutions* has increased. These are more centralised systems that are placed in conjunction with a blockchain. They allow users to deposit funds from the blockchain over to layer-two. On the layer-two users can then perform many of the actions they could do on the blockchain, but with lower transaction fees. One of the most popular

such solutions on Ethereum is Polygon [116]. Polygon has its own blockchain that allows users to interact with smart contracts as if they were using Ethereum.

The problem with layer-twos is that they do not leave any traces on the blockchain they are based on. When users deposit funds to Polygon, they are entered into an Ethereum address together with all the other Polygon funds. When the user wants to retrieve their funds, they are transferred out of the Polygon address back to the user's private address. All that can be seen on the Ethereum blockchain and in the Ethereum traces are the two transactions going back and forth to the Polygon address. There is no way of knowing what the funds were used for on Polygon, or if the funds switched hands during that time. All of the tracking strategies described in this thesis break down when there is a period of time for which funds do not have tracking data. For complete tracking, there would thus have to be some form of separate tracking solution implemented for Polygon and other layer-two solutions. Either that or layer-two solutions would have to be blacklisted by default.

7.6.5 Larger investigation of the FIFO issue

The issues we experienced with FIFO have not been fully investigated yet. We would like to investigate this further, but as there is no more time we will instead provide a plan here on how a potential investigation could be conducted. The heart of the issue is believed to be in the fragmentation that occurs when a partial amount of an address's contents are transferred to another address. The first step would thus be to investigate this further in order to prove or disregard this hypothesis.

We would start with an investigation into the frequency of new fragments forming and if this happens more often than fragments are merged. This could be done by collecting statistics while running FIFO on a sample set of real Ethereum transaction data. If it is observed that the number of fragments increases over time, the next step would be to investigate the relationship between fragment formation and the slowdown of the algorithm. Ideally, the number of fragments would be directly correlated to how long it takes to process a given chunk of transactions.

If a correlation is found we see two alternatives for increasing the processing speed for FIFO. The first and most obvious alternative would be to increase the speed at which FIFO processes transactions. Currently, the algorithms are implemented in Python where the fragments are stored in a Python list. It is possible that changing the implementation to use a more effective datatype, or to switch programming language entirely, might lead to a significant performance increase. The second alternative would be to either decrease the rate at which fragments are formed or to increase the rate at which fragments are merged. Both would require significant changes to how FIFO behave. A customised version of FIFO could have a maximum lifespan where all fragments above a certain age are merged into a single fragment when they are transferred from one address to another. Alternatively, the number of fragments created could be decreased by only tracking for a single or a smaller number of illicit addresses.

If it is found that fragmentation is not the main issue causing FIFO to slow down, further investigation of the different aspects of the FIFO algorithm would have to be conducted. By changing aspects of the algorithm and running the algorithm on a standardised dataset, clues to what causes the slowdown could be found.

7.6.6 Alternative origin tracking schemes

Though FIFO failed, it would also be interesting to investigate alternatives to the advanced FIFO scheme. We believe it might be possible to do a similar implementation for both haircut and seniority, where the origin of funds on each address could be stored as a portion of the tainted funds. The exact origin of each individual wei would not be known, but it could still be enough to satisfy the requirements brought on by the likes of the travel rule.

Haircut is extra interesting in this regard. Since it has a fair spread of taint, it could be the preferred choice if large institutions such as the FATF were to recommend a tracking strategy to become the global standard. If labelling, such as the one implemented for advanced FIFO, could be implemented for haircut, without slowing the algorithm down by more than a couple of magnitudes, it could become the future standard for AML on cryptocurrencies.

7.7 Conclusion

Two different anti-money laundering methods have been implemented and combined into a tool capable of tracking illicit assets on Ethereum. Although the tool is currently not able to track assets in real-time, it shows great potential as it is able to follow the flow of funds throughout the blockchain. Several blacklisting strategies were tested among which seniority and haircut showed the most promise. They were able to blacklist 132 million and 57 million addresses, respectively. Haircut distributed the tainted ether evenly throughout the blockchain while seniority kept it more concentrated to a select number of addresses. Clustering, the second main method, was also tested with successful results. The combination of clustering together with blacklisting was able to reveal 310,491 new potentially illicit addresses that were not visible with only blacklisting.

Bibliography

[1] Igor Makarov and Antoinette Schoar, "Blockchain Analysis of the Bitcoin Market," Oct. 2021. DOI: 10.3386/w29396.

[2] Sean Foley, Jonathan R Karlsen and Tālis J Putniņš, "Sex, Drugs, and Bitcoin: How Much Illegal Activity Is Financed through Cryptocurrencies?" In *The Review of Financial Studies*, Apr. 2019, pp. 1798–1853. DOI: 10.2139/ssrn.3102645.

[3] The European Commission, "EU context of anti-money laundering and countering the financing of terrorism," Accessed: Mar 2022. [Online]. Available: https://finance.ec.europa.eu/financial-crime/eu-context-anti-money-laundering-and-countering-financing-terrorism_en.

[4] Frances Schwartzkopff, "Danske Faces Long Road Back as Fine From Probes Seen Hitting $1 Billion," Accessed: Mar 2022. [Online]. Available: https://www.bloomberg.com/news/articles/2021-03-18/danske-faces-long-road-back-as-fine-seen-hitting-1-billion.

[5] fi.se, "Customer due diligence," Jun. 2023. [Online]. Available: https://www.fi.se/en/bank/money-laundering/process--work-method/customer-due-diligence/.

[6] Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf.

[7] , "blockchain.com Explorer," Accessed: Mar 2022. [Online]. Available: https://www.blockchain.com/explorer.

[8] getmonero.org, "What is Monero (XMR)?," Accessed: Jan 2023. [Online]. Available: https://www.getmonero.org/get-started/what-is-monero/.

[9] theblock.co, "The Block," Accessed: Apr 2022. [Online]. Available: https://www.theblock.co/data/on-chain-metrics/comparison-bitcoin-ethereum.

[10] Vitalik Buterin, "Ethereum Whitepaper," 2014. [Online]. Available: https://ethereum.org/en/whitepaper/.

[11] Cuneyt G. Akcora, Sudhanva Purusotham, Yulia R. Gel, Mitchell Krawiec-Thayer and Murat Kantarcioglu, "How to Not Get Caught When You Launder Money on Blockchain?," Sep. 2020. DOI: 10.48550/arXiv.2010.15082.

[12] Adam Hayes, "What Is a Blockchain?," Sep. 2022. [Online]. Available: https://www.investopedia.com/terms/b/blockchain.asp.

[13] Stuart Haber and W. Scott Stornetta, "How to time-stamp a digital document," in *Advances in Cryptology-CRYPTO' 90*, Berlin, Heidelberg, Germany, 1991, pp. 437–455. DOI: 10.1007/3-540-38424-3_32.

[14] Stuart Haber, W. Scott Stornetta and Dave Bayer, "Improving the Efficiency and Reliability of Digital Time-Stamping," in *Sequences II Methods in Communication, Security, and Computer Science*, New York, New York, USA, 1993, pp. 329–334. DOI: 10.1007/978-1-4613-9323-8_24.

[15] Greg Hall, "The little-known history of blockchain, as told by its inventors," Feb. 2022. [Online]. Available: https://bitcoinassociation.net/the-little-known-history-of-blockchain-as-told-by-its-inventors/.

[16] bitinfocharts.com, "Bitcoin Block Time historical chart," Accessed: July 2022. [Online]. Available: https://bitinfocharts.com/comparison/bitcoin-confirmationtime.html#3y.

[17] ycharts.com, "Ethereum Average Block Time," Accessed: July 2022. [Online]. Available: https://ycharts.com/indicators/ethereum_average_block_time.

[18] ibm.com, "Smart contracts defined," Accessed: Feb 2023. [Online]. Available: https://www.ibm.com/se-en/topics/smart-contracts.

[19] okta.com, "Hashing Algorithm Overview: Types, Methodologies Usage," Feb. 2023. [Online]. Available: https://www.okta.com/identity-101/hashing-algorithms/.

[20] Howard Poston, "Blockchain and hash functions," Mar. 2021. [Online]. Available: https://resources.infosecinstitute.com/topic/blockchain-and-hash-functions/.

[21] Ralph C. Merkle, "A Digital Signature Based on a Conventional Encryption Function," in *Advances in Cryptology — CRYPTO '87*, Berlin, Heidelberg, Germany, 1988, pp. 369–378. DOI: 10.1007/3-540-48184-2_32.

[22] wikipedia.org, "Merkle tree," Accessed: May 2022. [Online]. Available: https://en.wikipedia.org/wiki/Merkle_tree.

[23] pangea.cloud, "Merkle trees," Accessed: Feb 2023. [Online]. Available: https://pangea.cloud/docs/audit/merkle-trees.

[24] Einar Mykletun, Maithili Narasimha and Gene Tsudik, "Providing Authentication and Integrity in Outsourced Databases using Merkle Hash Tree's," 2002. [Online]. Available: http://people.eecs.berkeley.edu/~raluca/cs261-f15/readings/merkleodb.pdf.

[25] David Chaum, "Ecash," Accessed: May 2022. [Online]. Available: https://chaum.com/ecash/.

[26] David Chaum, "Blind Signatures for Untraceable Payments," in *Advances in Cryptology*, Boston, Massachusetts, USA, 1983, pp. 199–203. DOI: 10.1007/978-1-4757-0602-4_18.

[27] Nathan Reiff, "What Was the First Cryptocurrency?," Jul. 2022. [Online]. Available: https://www.investopedia.com/tech/were-there-cryptocurrencies-bitcoin/.

[28] Julia Kagan, "eCash," Mar. 2021. [Online]. Available: https://www.investopedia.com/terms/e/ecash.asp.

[29] stanford.edu, "E-gold," Accessed: May 2022. [Online]. Available: https://cs.stanford.edu/people/eroberts/cs201/projects/2010-11/Bitcoins/e-gold.html.

[30] Kim Zetter, "Bullion and Bandits: The Improbable Rise and Fall of E-Gold," Jun. 2009. [Online]. Available: https://www.wired.com/2009/06/e-gold/.

[31] bitcoin.it, "Hashcash," Apr. 2022. [Online]. Available: https://en.bitcoin.it/wiki/Hashcash.

[32] Adam Back, "Hashcash - A Denial of Service Counter-Measure," Aug. 2002. [Online]. Available: http://www.hashcash.org/papers/hashcash.pdf.

[33] Phillip Moskov, "What Is Bit Gold? The Brainchild of Blockchain Pioneer Nick Szabo," May 2018. [Online]. Available: https://coincentral.com/what-is-bit-gold-the-brainchild-of-blockchain-pioneer-nick-szabo/.

[34] Nick Szabo, "Bit gold," Dec. 2008. [Online]. Available: https://unenumerated.blogspot.com/2005/12/bit-gold.html.

[35] Rakesh Sharma, "Bit Gold," Oct. 2021. [Online]. Available: https://www.investopedia.com/terms/b/bit-gold.asp.

[36] Aaron Van Wirdum, "The genesis files: How hal finney's quest for digital cash led to rpow (and more)," Aug. 2020. [Online]. Available: https://bitcoinmagazine.com/culture/the-genesis-files-how-hal-finneys-quest-for-digital-cash-led-to-rpow-and-more.

[37] Wayne Duggan, "The History of Bitcoin, the First Cryptocurrency," Aug. 2022. [Online]. Available: https://money.usnews.com/investing/articles/the-history-of-bitcoin.

[38] Nick Szabo, "Bitcoin, what took ye so long?," May 2011. [Online]. Available: https://unenumerated.blogspot.com/2011/05/bitcoin-what-took-ye-so-long.html.

[39] financialgym.com, "Bitcoin 101: What is Bitcoin and Why Was it Created?," Jan. 2021. [Online]. Available: https://financialgym.com/blog/2021/1/2/bitcoin-101-what-is-bitcoin-and-why-was-it-created.

[40] bitcoin.zorinaq.com, "Bitcoin price," Accessed: Feb 2023. [Online]. Available: https://bitcoin.zorinaq.com/price/.

[41] Daniel Palmer and Alyssa Hertig, "Who Created Ethereum?," Mar. 2022. [Online]. Available: https://www.coindesk.com/learn/who-created-ethereum/.

[42] moderntreasury.com, "What is a ledger?," Feb. Accessed: Feb 2023. [Online]. Available: https://www.moderntreasury.com/learn/what-is-a-ledger.

[43] Shobhit Seth, "What Is a Cryptocurrency Public Ledger, How It Works, Risks," Aug. 2021. [Online]. Available: https://www.investopedia.com/tech/what-cryptocurrency-public-ledger/.

[44] Vaibhav Saini, "Getting Deep Into Ethereum: How Data Is Stored In Ethereum?," Jul. 2018. [Online]. Available: https://hackernoon.com/getting-deep-into-ethereum-how-data-is-stored-in-ethereum-e3f669d96033.

[45] Jay Kurahashi-Sofue, "What is a blockchain validator?," Feb. Accessed: Feb 2023. [Online]. Available: https://support.avax.network/en/articles/4064704-what-is-a-blockchain-validator.

[46] Paul Wackerow, "Transactions," Feb. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/transactions/.

[47] Bessie Liu, "Ethereum hits 500,000 validator milestone," Jan. 2023. [Online]. Available: https://blockworks.co/news/ethereum-to-reach-500000-validators.

[48] Luke Conway, "Measuring Decentralization: Is Your Crypto Decentralized?," Mar. 2022. [Online]. Available: https://blockworks.co/news/measuring-decentralization-is-your-crypto-decentralized.

[49] aws.amazon.com, "What is Decentralization in Blockchain?," Feb. Accessed: Feb 2023. [Online]. Available: https://aws.amazon.com/blockchain/decentralization-in-blockchain/.

[50] Euny Hong, "How Does Bitcoin Mining Work?," May 2022. [Online]. Available: https://www.investopedia.com/tech/how-does-bitcoin-mining-work/.

[51] Wouter Penard and Tim van Werkhoven, "On the Secure Hash Algorithm family," May 2008. [Online]. Available: https://blog.infocruncher.com/resources/ethereum-whitepaper-annotated/On%5C%20the%5C%20Secure%5C%20Hash%5C%20Algorithm%5C%20family%5C%20(2008).pdf.

[52] Corwin Smith, "Proof-of-stake (pos)," Jan. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/consensus-mechanisms/pos/.

[53] C. Taçoğlu, "Block reward," Feb. 2023. [Online]. Available: https://academy.binance.com/en/glossary/block-reward.

[54] Corwin Smith, "Gas and fees," Feb. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/gas/.

[55] river.com, "Block Subsidy," Feb. Accessed: Feb 2023. [Online]. Available: https://river.com/learn/terms/b/block-subsidy/.

[56] bitcoinmagazine.com, "Bitcoin's fair launch makes it an apex form of property," Nov. 2021. [Online]. Available: https://bitcoinmagazine.com/culture/why-bitcoin-fair-launch-is-important.

[57] Christine Kim, "A Breakdown of Ethereum Supply Distribution Since Genesis," Jun. 2022. [Online]. Available: https://www.galaxy.com/research/insights/breakdown-of-ethereum-supply-distribution-since-genesis/.

[58] learn.bybit.com, "Explained: What Is a Fair Launch Crypto?," Mar. 2023. [Online]. Available: https://learn.bybit.com/crypto/what-is-a-fair-launch-crypto/.

[59] Ren Heinrich, "Crypto Fair Launches — The Better ICOs," Feb. 2023. [Online]. Available: https://medium.com/coinmonks/crypto-fair-launches-the-better-icos-c9a6907fb96a.

[60] river.com, "Understanding Bitcoin Fungibility," Feb. Accessed: Feb 2023. [Online]. Available: https://river.com/learn/bitcoin-fungibility/.

[61] Priya Bansal, "What Is A Satoshi: Bitcoin's Smallest Unit Explained!," Mar. 2022. [Online]. Available: https://www.bankoncube.com/post/what-is-a-satoshi.

[62] Jean Cvllr, "Solidity Tutorial: All About Ether Units," Feb. 2023. [Online]. Available: https://betterprogramming.pub/solidity-tutorial-all-about-ether-units-eaebe55dd4dc.

[63] Pradosh Kumar Mohapatra, "Public Key Cryptography," in *XRDS: Crossroads, The ACM Magazine for Students*, Sep. 2000, pp. 14–22. DOI: 10.1145/351092.351098.

[64] Corwin Smith, "Ethereum accounts," Feb. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/accounts/.

[65] Andreas M. Antonopoulos and Gavin Wood, "Mastering Ethereum," 2018. [Online]. Available: https://www.oreilly.com/library/view/mastering-ethereum/9781491971932/ch04.html.

[66] P. Wackerow, "Introduction to smart contracts," Sep. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/smart-contracts/.

[67] Eiki, "Explaining ethereum contract abi evm bytecode," Jul. 2019. [Online]. Available: https://medium.com/@eiki1212/explaining-ethereum-contract-abi-evm-bytecode-6afa6e917c3b.

[68] Joshua, "Ethereum virtual machine (evm)," Jan. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/evm/.

[69] Paul Wackerow, "Merkle patricia trie," Feb. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/data-structures-and-encoding/patricia-merkle-trie/.

[70] Corwin Smith, "Nodes and clientse," Feb. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/.

[71] medium.com, "Messages and Transactions on Ethereum," Feb. 2020. [Online]. Available: https://cryptocurrency.medium.com/messages-and-transactions-on-ethereum-3c4dadfe7986.

[72] Thomas Jay Rush, "Building Your Own Ethereum Archive Node," Jan. 2022. [Online]. Available: https://tjayrush.medium.com/building-your-own-ethereum-archive-node-72c014affc09.

[73] Jithil P Ponnan, "Erc-20 token standard," Jan. 2023. [Online]. Available: https://ethereum.org/en/developers/docs/standards/tokens/erc-20/.

[74] William Entriken, "ERC-721: Non-Fungible Token Standard," Jan. 2018. [Online]. Available: https://eips.ethereum.org/EIPS/eip-721.

[75] fatf-gafi.org, "Updated Guidance for a Risk-Based Approach to Virtual Assets and Virtual Asset Service Providers," Oct. 2021. [Online]. Available: https://www.fatf-gafi.org/en/publications/fatfrecommendations/documents/guidance-rba-virtual-assets-2021.html.

[76] John Eligon, "Five More Accused in Credit Card Fraud Investigation," Aug. 2009. [Online]. Available: https://www.nytimes.com/2009/09/01/nyregion/01cyber.html.

[77] bitcoin.it, "bitcoin-Privacy," Sep. 2022. [Online]. Available: https://en.bitcoin.it/wiki/Privacy.

[78] Hannah Murphy, "Monero emerges as crypto of choice for cybercriminals," Jun. 2021. [Online]. Available: https://www.ft.com/content/13fb66ed-b4e2-4f5f-926a-7d34dc40d8b6.

[79] Andrew James Lom, Kim Caine and Rachael Browndorf, "New FATF guidance released on virtual assets and virtual asset service providers," Nov. 2021. [Online]. Available: https://www.nortonrosefulbright.com/en/knowledge/publications/024b3d80/new-fatf-guidance-released-on-virtual-assets-and-virtual-asset-service-providers.

[80] Erik Lie, "FATF Travel Rule, A Brief Introduction," 2020. [Online]. Available: https://assets.ctfassets.net/hfgyig42jimx/7ezqnx47HERbjVAofseOnf/792ed1f4c49384a65541aa4456cf037d/Crypto.com_Macro_Report_-_FATF_Travel_Rule.pdf.

[81] Dan Gorton, Handelsbanken, May 2022 [Verbal Account].

[82] fatf-gafi.org, "Virtual Assets Red Flag Indicators of Money Laundering and Terrorist Financing," Sep. 2020. [Online]. Available: https://www.fatf-gafi.org/en/publications/Methodsandtrends/Virtual-assets-red-flag-indicators.html.

[83] fatf-gafi.org, "Who we are," Accessed: June 2022. [Online]. Available: https://www.fatf-gafi.org/en/the-fatf/who-we-are.html.

[84] kyc-chain.com, "Explaining the FATF Travel Rule," Apr. 2020. [Online]. Available: https://kyc-chain.com/explaining-the-fatf-travel-rule/.

[85] Stanley E. Morris, "FinCEN Advisory Issue 7," Oct. 1997. [Online]. Available: https://www.fincen.gov/sites/default/files/advisory/advissu7.pdf.

[86] Bernhard Haslhofer, Roman Karl and Erwin Filtz, "O Bitcoin Where Art Thou? Insight into Large-Scale Transaction Graphs.," Vienna, Austria, 2016. [Online]. Available: https://ceur-ws.org/Vol-1695/paper20.pdf.

[87] Victor Friedhelm, "Address Clustering Heuristics for Ethereum," in *International Conference on Financial Cryptography and Data Security*, Kota Kinabalu, Malaysia, Feb. 2020, pp. 617–633. DOI: 10.1007/978-3-030-51280-4_33.

[88] Ferenc Béres and István A. Seres and András A. Benczúr and Mikerah Quintyne-Collins, "Blockchain is Watching You: Profiling and Deanonymizing Ethereum Users," in *IEEE International Conference on Decentralized Applications and Infrastructures*, United Kingdom, Aug. 2021, pp. 69–78. DOI: 10.1109/DAPPS52256.2021.00013.

[89] Nesreen K. Ahmed, Ryan A. Rossi, John Boaz Lee, Ted Willke, Rong Zhou, Xiangnan Kong and Hoda Eldardiry, "Learning Role-based Graph Embeddings," in *IEEE Transactions on Knowledge and Data Engineering: Volume 34*, Jul. 2014, pp. 2401–2415. DOI: 10.1109/TKDE.2020.3006475.

[90] Benedek Rozemberczki and Rik Sarkar, "Fast Sequence Based Embedding with Diffusion Graphs," in *International Workshop on Complex Networks*, Jan. 2018, pp. 99–107. DOI: 10.1007/978-3-319-73198-8_9.

[91] Mike Wu, Will McTighe, Kaili Wang, Istvan A. Seres, Nick Bax, Manuel Puebla, Mariano Mendez, Federico Carrone, Tomás De Mattey, Herman O. Demaestri, Mariano Nicolini and Pedro Fontana, "Tutela: An Open-Source Tool for Assessing User-Privacy on Ethereum and Tornado Cash," Jan. 2022. DOI: 10.48550/arXiv.2201.06811.

[92] github.com, "Tutela: an Ethereum and Tornado Cash Anonymity Tool," Accessed: May 2023. [Online]. Available: https://github.com/pareto-xyz/tutela-app.

[93] Vitalik Buterin, "Mtgox: What the largest exchange is doing about the linode theft and the implications," May 2012. [Online]. Available: https://bitcoinmagazine.com/markets/mtgox-the-bitcoin-police-what-the-largest-exchange-is-doing-about-the-linode-theft-and-the-implications-1337616444.

[94] 556j, "Mt Gox thinks it's the Fed. Freezes acc based on "tainted" coins," Mar. 2012. [Online]. Available: https://bitcointalk.org/index.php?topic=73385.0.

[95] Malte Möser, Rainer Böhme and Dominic Breuker, "An inquiry into money laundering tools in the Bitcoin ecosystem," in *APWG eCrime Researchers Summit*, San Francisco, CA, USA, Sep. 2013, pp. 1–14. DOI: 10.1109/eCRS.2013.6805780.

[96] Malte Möser, Rainer Böhme and Dominic Breuker, "Towards Risk Scoring of Bitcoin Transactions," in *International Conference on Financial Cryptography and Data Security*, Christ Church, Barbados, Jan. 2014, pp. 16–32. DOI: 10.1007/978-3-662-44774-1_2.

[97] Svetlana Abramova, Pascal Schöttle and Rainer Böhme, "Mixing Coins of Different Quality: A Game-Theoretic Approach," in *International Conference on Financial Cryptography and Data Security*, Sliema, Malta, Apr. 2017, pp. 280–297. DOI: 10.1007/978-3-319-70278-0_18.

[98] Malte Möser and Arvind Narayanan, "Effective Cryptocurrency Regulation Through Blacklisting," 2019. [Online]. Available: https://allquantor.at/blockchainbib/pdf/moser2019effective.pdf.

[99] Zhiyuan Chen, Le Dinh Van Khoa, Ee Na Teoh, Amril Nazir, Ettikan Kandasamy Karuppiah and Kim Sim Lam, "Machine learning techniques for anti-money laundering (AML) solutions in suspicious transaction detection: a review," in *Knowledge and Information Systems*, Feb. 2018, pp. 245–285. DOI: 10.1007/s10115-017-1144-z.

[100] Joana Lorenz, Maria Inês Silva, David Aparício, João Tiago Ascensão and Pedro Bizarro, "Machine learning methods to detect money laundering in the Bitcoin blockchain in the presence of label scarcity," in *Proceedings of the First ACM International Conference on AI in Finance*, New York, New York, USA, Oct. 2021, pp. 1–8. DOI: 10.1145/3383455.3422549.

[101] Mark Weber, Giacomo Domeniconi, Jie Chen, Daniel Karl I. Weidele, Claudio Bellei, Tom Robinson and Charles E. Leiserson, "Anti-Money Laundering in Bitcoin: Experimenting with Graph Convolutional Networks for Financial Forensics," Jul. 2019. DOI: 10.48550/arXiv.1908.02591.

[102] Weili Chen, Zibin Zheng, Edith Ngai, Peilin Zheng and Yuren Zhou, "Exploiting Blockchain Data to Detect Smart Ponzi Schemes on Ethereum," in *IEEE Access*, Mar. 2019, pp. 37 575–37 586. DOI: 10.1109/ACCESS.2019.2905769.

[103] Hamish Hall, "Labelled Ethereum Addresses," Aug. 2020. [Online]. Available: https://www.kaggle.com/datasets/hamishhall/labelled-ethereum-addresses.

[104] Harry Deneley, "Blacklisted / Sanctioned Ethereum Mainnet Addresses," Accessed: May 2023. [Online]. Available: https://dune.com/harrydenley/BlacklistedSanctioned-Addresses.

[105] Gamaliel Padillo, IRMkz, Luca, Will Foster and kolya182, "MyEtherWallet/ethereum-lists," Nov. 2020. [Online]. Available: https://github.com/MyEtherWallet/ethereum-lists/blob/master/src/addresses/addresses-darklist.json.

[106] ycharts.com, "Ethereum Cumulative Unique Addresses (I:ECUA)," Accessed: May 2023. [Online]. Available: https://ycharts.com/indicators/ethereum_cumulative_unique_addresses.

[107] Baran Kılıç, Can Özturan and Alper Sen, "Parallel analysis of Ethereum blockchain transaction data using cluster computing," in *Cluster Comput 25*, Jan. 2022, pp. 1885–1898. DOI: 10.1007/s10586-021-03511-0.

[108] Corwin Smith, "Nodes and clients," Mar. 2022. [Online]. Available: https://ethereum.org/en/developers/docs/nodes-and-clients/.

[109] tornadocash.eth.link, "Tornado Cash homepage," Accessed: May 2023. [Online]. Available: https://tornadocash.eth.link/.

[110] Ibm.com, "Hash lookup," Jan. 2023. [Online]. Available: https://www.ibm.com/docs/en/iotdm/11.3?topic=lf-hash-lookup-function.

[111] treasury.gov, "U.S. Treasury Sanctions Notorious Virtual Currency Mixer Tornado Cash," Aug. 2022. [Online]. Available: https://home.treasury.gov/news/press-releases/jy0916.

[112] Samuel Haig, "More Than Half of Ethereum Network is Excluding U.S.-Sanctioned Wallets," Oct. 2022. [Online]. Available: https://thedefiant.io/flashbots-tornado-sanctions-mev.

[113] ycharts.com, "Bitcoin Blockchain Size (I:BBS)," Accessed: May 2023. [Online]. Available: https://ycharts.com/indicators/bitcoin_blockchain_size.

[114] ycharts.com, "Ethereum Chain Full Sync Data Size (I:ECFSDS)," Accessed: May 2023. [Online]. Available: https://ycharts.com/indicators/ethereum_chain_full_sync_data_size.

[115] Michael Laine, "What's the current size of the Solana blockchain?," Jul. 2022. [Online]. Available: https://solana.stackexchange.com/questions/146/whats-the-current-size-of-the-solana-blockchain.

[116] defillama.com, "Total Value Locked All Chains," Accessed: Feb 2023. [Online]. Available: https://defillama.com/chains.