



**UNIVERSITY OF PIRAEUS**

**DEPARTMENT OF DIGITAL SYSTEMS**

**Postgraduate Program : "Digital Systems Security"**

**ACADEMIC YEAR 2019-2020**

**<An investigation into blockchain's forensic artifacts>**

**Antigoni Zisi**

MTE1716

**Supervisor**

Christoforos Dadoyan

**Piraeus, January 2020**



## Table of Contents

<b>1. Introduction</b>	6
<b>2. Anonymity in the Bitcoin System</b>	7
2.1 How Bitcoin works	7
2.2 Interesting Bitcoin Features	9
2.3 The Transaction Network $T$	10
2.4 The User Network $U$	11
2.5 Anonymity Analysis	13
2.5.1 Integrating off-Network Information	14
2.5.2 Analysis and Visualization of the User Network	15
2.5.3 Findings	16
2.5.4 Flow Analysis	19
2.6 Research's conclusion	20
<b>3. Cryptocurrencies and Illicit Activities</b>	20
3.1 Money Laundering in the Crypto World	21
3.2 Cryptomarkets and Drug Trafficking	23
3.3 Extortion, Attacks and Terrorism Financing	24
3.4 Ponzi Schemes/Pump and Dump	25
<b>4. Anonymity over Tor</b>	25
4.1 Cryptos and Tor	26
4.2 Bitcoins services	27
4.3 How Tor works	29
4.4 Possible Attacks	32
4.4.1 Exploitation	32
4.4.2 Defeating onion peers	34
4.4.5 Sybil Attacks on Bitcoin	36
4.4.6 Attacks Description	36
4.5 Conclusion	37
<b>5. Bitcoin Forensics</b>	37
5.1 Wallets and Transactions	37
5.2 Bitcoin artifacts	40
5.3 Investigation	41
5.3.1 Building a case	41
5.3.2 Bitcoin forensic artifact examination	43
5.3.3 Hardware Setup	43

5.3.4 Tools	44
5.4 Results	45
5.4.1 Overview	45
5.4.2 Collection and analysis of evidence	48
5.5 Conclusion	54
<b>6. Memory Investigation of Bitcoin Clients</b>	<b>55</b>
6.1 Bitcoin Core and Electrum	55
6.2 Memory Images	58
6.3 Findings for Bitcoin Core	60
6.4 Findings for Electrum	63
6.5 Conclusion	65
<b>7. Ethereum and Smart Contracts</b>	<b>66</b>
7.1 Ethereum Virtual Machine	66
7.2 Ethereum Honeypots	67
7.3 Manticore	68
7.3.1 What is Manticore?	68
7.3.2 Architecture	69
7.3.3 Core Engine	70
7.3.4 Native Execution Module	71
7.3.5 Ethereum Execution Module	72
7.3.6 Auxiliary Modules	73
7.4 Usage	73
7.5 Ethereum Smart Contract Analysis Evaluation	75
7.6 How Manticore can be used for smart contract investigation	76
<b>8 Conclusion</b>	<b>78</b>
<b>9 References</b>	<b>79</b>

## Table of Figures

Figure 1: Sub-network from the Transaction Network <b>T</b>	11
Figure 2: Sub-network of the User Network <b>U</b>	12
Figure 3: Vertex representation of public keys	15
Figure 4: Visualization of identified users	16
Figure 5: Thief Visualization	17
Figure 6: Thief-victim Sub-network	18
Figure 7: Tool representation	20
Figure 8: Tor Guards	26
Figure 9: Chain of signatures of signatures associated with a Bitcoin transaction as it progresses from one owner to the next	35
Figure 10: Transaction Ledger	36
Figure 11: Screenshot of the folder structure of Multibit installed on the test system.	42
Figure 12: Screenshot of the folder structure of Bitcoin-Qt installed on the test system	43
Figure 13: Screenshot depicting the transactions conducted within the Multibit wallet on the test system.	44
Figure 14: Screenshot depicting the transactions conducted within the Bitcoin-Qt wallet	44
Figure 15: Screenshot depicting the two separate dates for the unencrypted rolling backups of the Suspect's	47
Figure 16: Some Multibit artifacts recovered during the hard drive analysis Evidentiary Artifact Location of Artifact	48
Figure 17: Bitcoin Core Wallet	53
Figure 18: Application data in Bitcoin Core and Electrum.	54
Figure 19: Memory Images per Application	55
Figure 20: Honeypot Phases	64
Figure 21: Manticore Logo	65
Figure 22: Manticore's Architecture	66
Figure 23: The State Life Cycle	67
Figure 24: Ethereum Contract Code Coverage	71

## 1. Introduction

Cryptocurrencies have been a significant topic during the past 10 years. Relying on the blockchain technology to gain decentralization, transparency and immutability, this internet-based medium of exchange uses cryptographic functions to conduct financial transactions. However, they have entered the digital world serving as an innocent (until proven otherwise) means of payment and at the same time, a gate to illicit activities such as money laundering, information disclosure, crime coverage, etc.

Digital forensics is a branch of forensic science encompassing the recovery and investigation of material found in digital devices, often in relation to computer crime. The term digital forensics covers investigation of all devices capable of storing digital data. The typical forensic process encompasses the seizure, forensic imaging (acquisition) and analysis of digital media and the production of a report into collected evidence.

In this thesis, the combination of the two aforementioned concepts is analyzed and research is made on their artifacts as well as the impact of those in the so-called accountability of the blockchain network.

In chapter 2, a brief but indicative explanation of the general perspective of the blockchain network is introduced as far as key concepts like the user anonymity. The results of public keys' tracking are analyzed in regards to the user anonymity and the most significant findings are explained.

Chapter 3 deals with the concept of illicit activities in the cryptocurrency world alongside chapter 4 which refers to the "interdependencies" of the cryptocurrencies and Tor.

Chapter 5 goes into a thorough description of artifacts provided from a case investigation. Forensics examination is performed in the memory, disk, file locations and paths of some transactions and the key findings are analyzed regarding their readability.

The next chapter, uses the examples of two known Bitcoin Clients, Bitcoin Core and Electrum, in order to expose the results of forensics research into the aforementioned artifacts.

Finally, chapter 7 describes the Ethereum Platform, the smart contracts perspective, and introduces the idea of Manticore, a tool for forensics investigation. Chapter 8 concludes with the previous chapters' results and sets some food for thought for future consideration.

## 2. Anonymity in the Bitcoin System

Bitcoin has the confusing property that while the ownership of money is implicitly anonymous, its flow is globally visible. Bitcoin identities are thus, pseudo-anonymous: while not explicitly tied to real-world individuals or organizations, all transactions are completely transparent. This unusual combination of features has given rise to considerable confusion about the nature and consequences of the anonymity that Bitcoin provides. In particular, there is concern that the combination of scalable, irrevocable, anonymous payments would prove highly attractive for criminals engaged in fraud or money laundering. Using the dissolution of a large Silk Road [1] wallet and notable Bitcoin thefts [2] as case studies, we demonstrate that an agency with subpoena power would be well placed to identify who is paying money to whom.

### 2.1 How Bitcoin works

Bitcoin is a decentralized electronic currency, introduced by (the individual or the team) Satoshi Nakamoto in 2008 and deployed on January 3 2009. A chain of transactions from one owner to the next, where owners are identified by a public key (ECDSA signature scheme [3]) that serves as a pseudonym. For example, users can use any number of public keys and their activity using one set of public keys is not inherently tied to their activity using another set or to their real-world identity. In each transaction, the previous owner signs, using the secret signing key corresponding to his public key, a hash of the transaction in which he received the bitcoins (a SHA-256 hash [4]) and the public key of the next owner. This transaction is then added to the set of transactions that constitutes the blockchain. Since each of these transactions refers to the previous transaction (e.g. while sending bitcoins, the current owner must specify where they came from), the transactions form a chain. To verify the validity of a bitcoin, a user can check the validity of each of the signatures in this chain.

A considerable issue that can arise is double spending [5]. To prevent double spending, it is necessary for each user in the system to be aware of all the transactions. Double spending can then be identified when a user attempts to transfer a bitcoin after he has already done so. To determine which transaction came first, transactions are grouped

into blocks, which serve to timestamp the transactions they contain and vouch for their validity. Blocks are themselves formed into a chain, with each block referencing the previous one and thus further reinforcing the validity of all previous transactions. This process yields a blockchain, which is then publicly available to every user within the system.

Throughout this process, bitcoins are transferred and transactions are broadcasted to all users of the system. But, since Bitcoin is decentralized and there is no central authority minting bitcoins, how are bitcoins generated in the first place? In fact, this happens in the process of forming a block. Each accepted block (each block incorporated into the block chain) is required to be such that, when all the data inside the block is hashed, the hash begins with a certain number of zeroes. To allow users to find this particular collection of data, blocks contain, in addition to a list of transactions, a nonce. Once someone finds a nonce that allows the block to have the correctly formatted hash, the block is then broadcasted in the same peer-to-peer manner as transactions. At inception, each bitcoin block reward was worth 50 BTC. This reward is halved after the discovery of every 210,000 blocks, which takes around four years to complete. As of February 2019, one block reward was worth 12.5 BTC [6]. The amount is expected to hit zero around 2140.

In summary, the shared information within the Bitcoin network works as follows. Firstly, users generate at least one signing keypair and publicize the public key or address to receive bitcoins (as mentioned earlier, users can choose to use a single public key or arbitrarily many). If a user has bitcoins that he wishes to transfer, he broadcasts a transaction, proving that he has the bitcoins and indicating the address of the recipient to his peers, who in turn broadcast it to their peers. Eventually, this transaction reaches a miner who collects the transactions into a block and works on finding the right data/nonce balance to hit the target hash. He also includes in the block a special coin generation transaction that specifies his address for receiving the block reward. Finally, when the miner does find such a block, he broadcasts it to his peers who again broadcast it to their peers. As his reward, the block reward and all the fees for the included transactions are credited to his specified address. When another block has been formed, referencing his block as the previous block, his block can now be considered part of the blockchain [7].



## 2.2 Interesting Bitcoin Features

Three features of the Bitcoin system are of particular interest. Firstly, the entire history of Bitcoin transactions is publicly available. As described in the previous section, this is necessary in order to validate transactions and prevent double-spending in the absence of a central authority. The only way to confirm the absence of a previous transaction is to be aware of all previous transactions. The second feature of interest is that a transaction can have multiple inputs and multiple outputs. An input to a transaction is either the output of a previous transaction or a sum of newly generated Bitcoins and transaction fees. A transaction frequently has either a single input from a previous larger transaction or multiple inputs from previous smaller transactions. Also, a transaction frequently has two outputs: one sending payment and one returning change. Thirdly, the payer and payee(s) of a transaction are identified through public keys from public-private key pairs. However, a user can have multiple public keys. In fact, it is considered good practice for a payee to generate a new public-private keypair for every transaction. Furthermore, a user can take the following steps to better protect their identity:

- He can avoid revealing any identifying information in connection with his public keys.
- He can repeatedly send varying fractions of his Bitcoins to himself using multiple (newly generated) public keys and/or
- He can use a trusted third-party mixer or laundry.

These three features, namely the public availability of Bitcoin transactions, the input-output relationship between transactions and the re-use and co-use of public keys, provide a basis for two distinct network structures: the transaction network (T) and the user network (U). The transaction network represents the flow of Bitcoins between transactions over time. Each vertex represents a transaction and each directed edge between a source and a target represents an output of the transaction corresponding to the source that is an input to the transaction corresponding to the target. Each directed edge also includes a value in Bitcoins and a timestamp. The user network represents the flow of Bitcoins between users over time. Each vertex represents a user and each directed edge between a source and a target represents an input-output pair of a single transaction where the input's public key belongs to the user corresponding to the source and the output's public key

belongs to the user corresponding to the target. Each directed edge also includes a value in Bitcoins and a timestamp.

The below research from Fergal Reid and Martin Harrigan [8] shows how two networks (The Transaction Network and the User Network) derived from Bitcoin's public transaction history can have implications on the user's anonymity.

### 2.3 The Transaction Network $T$

The Transaction Network  $T$  represents the flow of Bitcoins between transactions over time. Each vertex represents a transaction and each directed edge between a source and a target represents an output of the transaction corresponding to the source that is an input to the transaction corresponding to the target. Each directed edge also includes a value in Bitcoins and a timestamp. The below Figure shows an example sub-network of  $T$ .  $t_1$  is a transaction with one input and two outputs. It was added to the blockchain on the 1st May 2011. One of its outputs assigned 1.2 BTC (Bitcoins) to a user identified by the public key  $pk_1$ . Similarly,  $t_2$  is a transaction with two inputs and two outputs, accepted on the 5th May 2011. One of its outputs sent 0.12 BTC to a user identified by a different public key,  $pk_2$ .  $t_3$  is a transaction with two inputs and one output, accepted on the 5th May 2011. Both of its inputs are connected to the two aforementioned outputs of  $t_1$  and  $t_2$ . The only output of  $t_3$  was redeemed by  $t_4$ .

$T$  has 974.520 vertices and 1.558.854 directed edges. The number of vertices is less than the total number of transactions in the dataset because transactions that are not connected to at least one other transaction are omitted. These correspond to newly generated Bitcoins and transactions fees that are not yet redeemed. The network has neither multi-edges (multiple edges between the same pair of vertices in the same direction) nor loops. It is a directed acyclic graph (DAG) since the output of a transaction can never be an input (either directly or indirectly) to the same transaction.

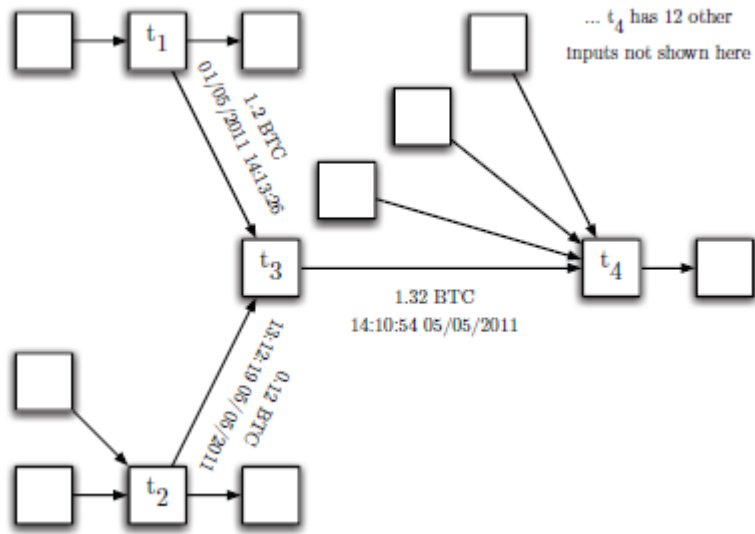


Figure 1: Sub-network from the Transaction Network  $T$

## 2.4 The User Network $U$

The user network  $U$  represents the flow of Bitcoins between users over time. Each vertex represents a user and each directed edge between a source and a target represents an input-output pair of a single transaction where the input's public key belongs to the user corresponding to the source and the output's public key belongs to the user corresponding to the target. Each directed edge also includes a value in Bitcoins and a timestamp.

Suppose  $U$  is, at first, imperfect in the sense that each vertex represents a single public key rather than a user and that each directed edge between a source and a target represents an input-output pair of a single transaction, where the input's public key corresponds to the source and the output's public key corresponds to the target. In order to perfect this network, each subset of vertices whose corresponding public keys belong to a single user need to be contracted. The difficulty is that public keys are Bitcoin's mechanism for ensuring anonymity. In fact, it is considered good practice for a payee to generate a new public private key-pair for every transaction to keep transactions from being linked to a common owner. Therefore, it is impossible to completely perfect the network using the dataset alone. However, as Nakamoto has noted [9], some linking is still unavoidable with multi-input transactions, which necessarily reveal that their inputs were owned by the same

owner. The risk is that if the owner of a key is revealed, linking could reveal other transactions that belonged to the same owner.

This property of transactions was used with multiple inputs to contract subsets of vertices in the imperfect network. An ancillary network is constructed in which each vertex represents a public key. These vertices are connected with undirected edges, where each edge joins a pair of public keys that are both inputs to the same transaction (and are thus controlled by the same user). In the dataset, this network has 1.253.054 vertices (unique public keys) and 4.929.950 edges. More importantly, it has 86.641 non-trivial maximal connected components. Each maximal connected component in this graph corresponds to a user and each component's constituent vertices correspond to that user's public keys.

The outputs of t1 and t2 that were eventually redeemed by t3 were sent to a user whose public key was pk1 and a user whose public key was pk2 respectively. pk1 and pk2 are contracted into a single vertex u1 since they correspond to a pair inputs of a single transaction so, they are in the same maximal connected component of the ancillary network (vertices representing pk1 and pk2 in the dashed grey box in the below Figure). A single user owns both public-keys. The maximal connected component in this case is not simply a clique since it has a diameter of four indicating that there are at least two public keys belonging to that same user that are connected indirectly via three transactions. The sixteen inputs to transaction t4 result in the contraction of a further sixteen public keys into a single vertex u2. The value and timestamp of the flow of Bitcoins from u1 to u2 is derived from the transaction network. After the preprocessing step, U has 881.678 vertices (86.641 non-trivial maximal connected components and 795.037 isolated vertices in the ancillary network) and 1.961.636 directed edges. Unlike T, U has multi-edges, loops and directed cycles.

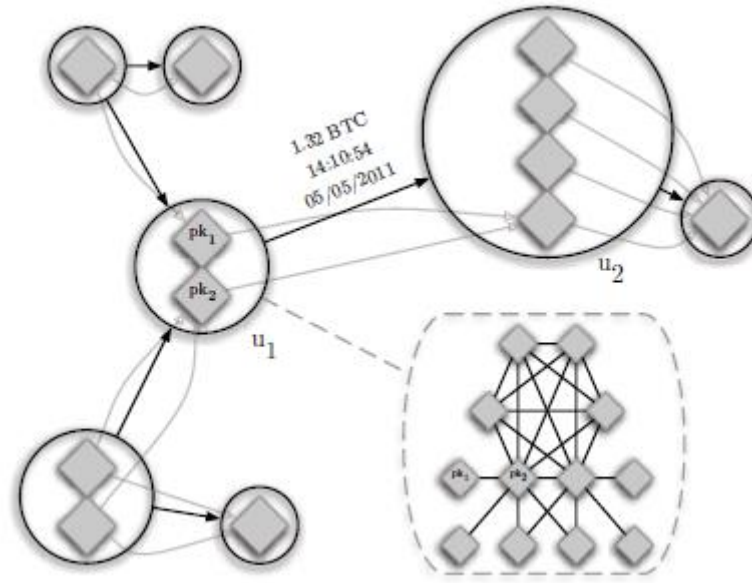


Figure 2: Sub-network of the User Network  $U$

## 2.5 Anonymity Analysis

The analysis reveals that the user network has considerable cyclic structure. The implications of this structure, coupled with other aspects of the Bitcoin system for anonymity should be considered.

There are several ways in which the user network can be used to deduce information about Bitcoin users. Global network properties can be used, such as degree distribution, to identify outliers. Local network properties can be used to examine the context in which a user operates by observing the users with which he interacts with either directly or indirectly. The dynamic nature of the user network also enables us to perform flow and temporal analysis. The significant Bitcoin flows between groups of users over time can be examined. Each of these possibilities is described in more detail and a case study to demonstrate their use in practice is provided in the following sections.

### 2.5.1 Integrating off-Network Information

There is no user directory for the Bitcoin system. However, for the purposes of the research, a partial user directory was built associating Bitcoin users and their known public keys with off-network information.

Many organizations and services such as on-line stores that accept Bitcoins, exchanges, laundry services and mixers have access to identifying information regarding their users, e.g. e-mail addresses, shipping addresses, credit card and bank account details, IP addresses, etc. If any of this information is publicly available or accessible by the law enforcement agencies, then the identities of users involved in related transactions may also be at risk. To illustrate this point, a number of publicly available data sources was considered and their information with the user network was integrated.

The Bitcoin Faucet [10] is a website where users can donate Bitcoins to be redistributed in small amounts to other users. In order to prevent abuse of this service, a history of recent give-aways are published along with the IP addresses of the recipients. When the Bitcoin Faucet does not batch the redistribution, it is possible to associate the IP addresses with the recipient's public-keys. This page can be scraped over time to produce a time-stamped mapping of IP addresses to users.

It was found that the public keys associated with many of the IP addresses that received Bitcoins were contracted with other public keys in the ancillary network, thus revealing IP addresses that are somehow related to previous transactions.

Another source of identifying information is the voluntary disclosure of public keys by users, for example, when posting to the Bitcoin forums [11]. Bitcoin public keys are typically represented as strings approximately thirty-three characters in length and starting with the digit one. They are indexed very well by popular search engines. Many high degree vertices were identified with external information using a search engine alone. Bitcoin Forums were searched where users frequently attach a public key to their signatures. Public keys from Twitter streams and user-generated public directories were also found. It should be noted that large centralized Bitcoin service providers can do the same with their user information.

## 2.5.2 Analysis and Visualization of the User Network

There are several pieces of information directly derived from the user network regarding a particular user:

- The balance held by a single public key can be computed.
- We can also aggregate the balances belonging to public keys that are controlled by a particular user can be aggregated.

The donations, on the aforementioned websites, are relatively small and are forwarded to other public keys periodically. There was also a noticeable spike in donations when the facility was first announced. An important advantage of deriving network structures from the Bitcoin transaction history is the ability to use network visualization and analysis tools to investigate the flow of Bitcoins. For example, the below Figure shows the network structure surrounding the public key in the user network. Several of the vertices contain identifying information so, these users can be linked either directly or indirectly to their donations. The color denotes the volume of Bitcoins as warmer colors have larger volumes flowing through them. The large red vertices represent a Bitcoin mining pool, a centralized Bitcoin wallet service and an unknown entity.

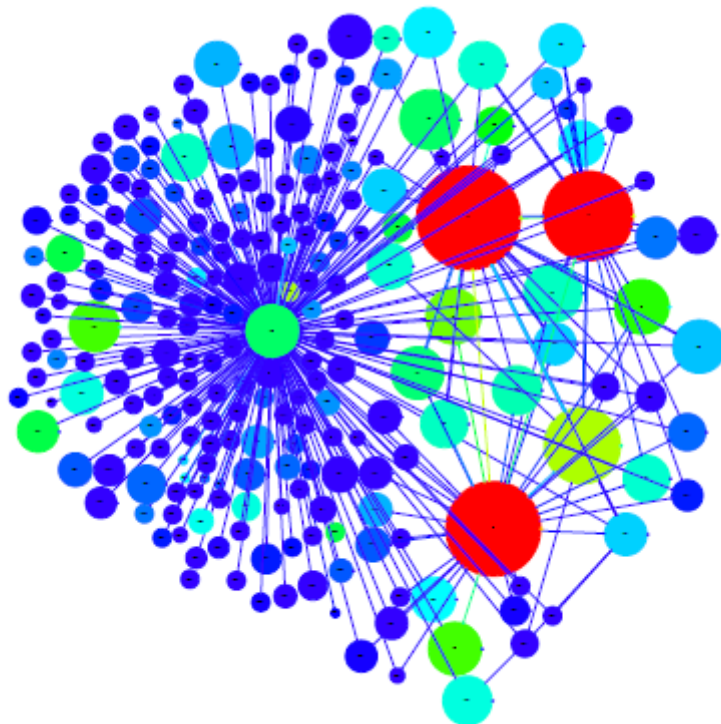
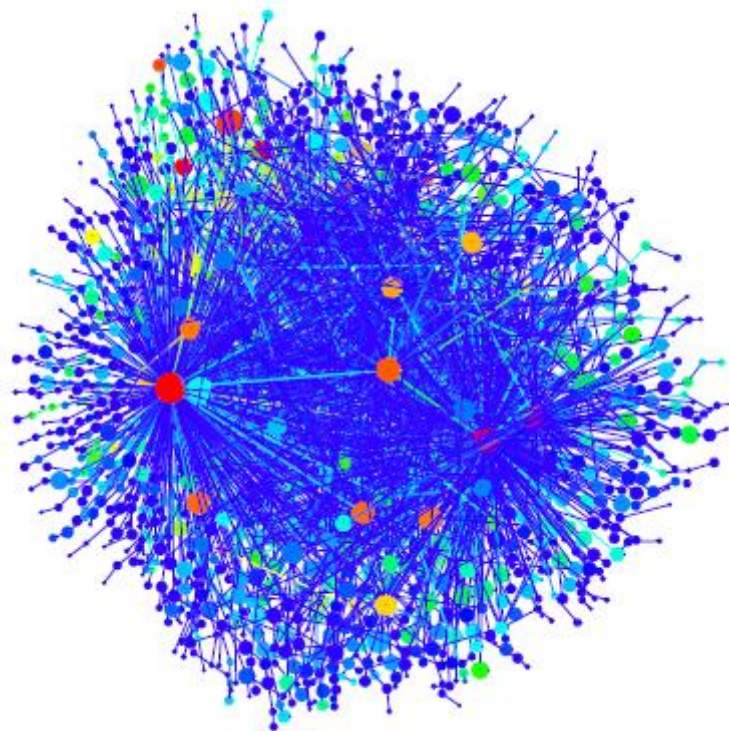


Figure 3: Vertex representation of public keys

### 2.5.3 Findings

Given a number of public keys or users of interest, we can use network structure and context to better understand the flow of Bitcoins between them. For example, all shortest paths between a set of vertices can be examined. Also, the maximum number of Bitcoins that can flow from a source to a destination given the transactions and their 'capacities' can be considered in an interesting time-window. For example, the below Figure shows all shortest paths between the vertices representing the users who were identified using off-network information and the vertex that represents the MyBitcoin service [12] in the user network. More than 60% of the users can be identified in this visualization and direct and indirect relationships between them can be noted.



*Figure 4: Visualization of identified users*

Case Study – Part I: An alleged theft of 25 000 BTC reported in the Bitcoin Forums by a user known as *allinvain* was analyzed at first. The victim reported that a large portion of his



Bitcoins were sent to  $pk_{red}$  [13] on 13/06/2011 at 16:52:23 UTC. The theft occurred shortly after somebody broke into the victim's Slush pool account [14] and changed the payout address to  $pk_{blue}$  [15]. The Bitcoins rightfully belonged to  $pk_{green}$  [16]. At the time of theft, the stolen Bitcoins had a market value of approximately half a million U.S. dollars. This case study was chosen in order to illustrate the potential risks to the anonymity of a user (the thief) who has good reason to remain anonymous.

A restriction to the egocentric network surrounding the thief is made: we include every vertex that is reachable by a path of length at most two ignoring directionality and all edges induced by these vertices. We also remove all loops, multiple edges and edges that are not contained in some biconnected component to avoid clutter. In the below Figure, the red vertex represents the thief who owns the public key  $pk_{red}$  and the green vertex represents the victim who owns the public key  $pk_{green}$ . The theft is the green edge joining the victim to the thief. There are in fact two green edges located nearby in the below Figure but only one directly connects the victim to the thief.

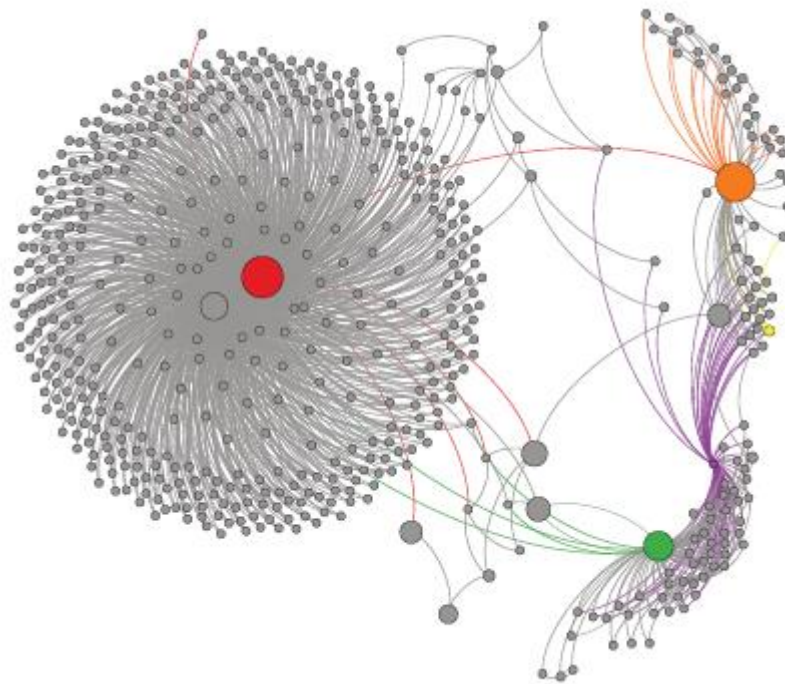


Figure 5: Thief Visualization

Interestingly, the victim and the thief are joined by paths (ignoring directionality) other than the green edge representing the theft. For example, consider the sub-network shown in the below Figure induced by the red, green, purple, yellow and orange vertices.

This sub-network is a cycle. All vertices whose corresponding public-keys belong to the same user are contracted. This permits to attach values in Bitcoins and timestamps to the directed edges. A number of observations can be made. Firstly, we note that the theft of 25.000 BTC was preceded by a smaller theft of 1 BTC. This was later reported by the victim in the Bitcoin forums. Secondly, using off-network data, we have identified some of the other colored vertices: the purple vertex represents the main Slush pool account and the orange vertex represents the computer hacker group known as LulzSec [17]. We observe that the thief sent 0.31337 BTC to LulzSec shortly after the theft but we cannot otherwise associate him with the group. The main Slush pool account sent a total of 441.83 BTC to the victim over a 70-day period. It also sent a total of 0.2 BTC to the yellow vertex over a two-day period. One day before the theft, the yellow vertex also sent 0.120607 BTC to LulzSec. The yellow vertex represents a user who is the owner of at least five public keys. Like the victim, he is a member of the Slush pool and like the thief, he is a one-time donator to LulzSec. This donation, the day before the theft, is his last known activity using these public-keys.

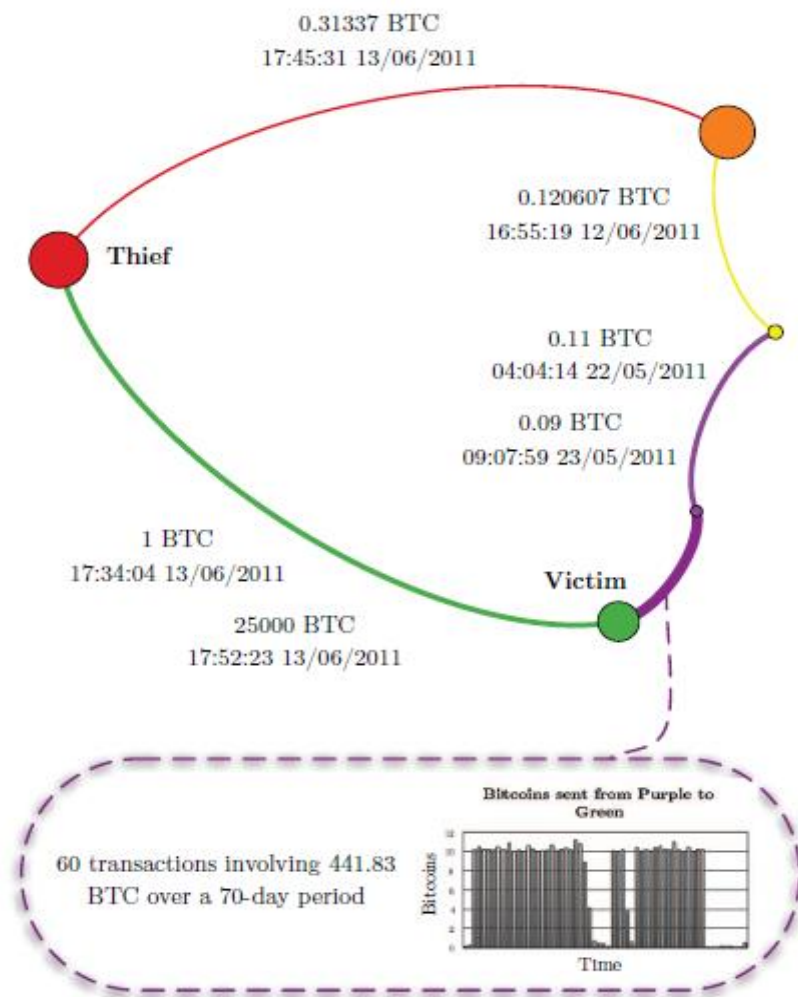


Figure 6: Thief-victim Sub-network

## 2.5.4 Flow Analysis

Significant flows of value through the network over time can be followed. For example, a vertex representing a user receives a large volume of Bitcoins relative to their estimated balance and shortly after, transfers a significant proportion of those Bitcoins to another user, is an interesting occurrence. A special tool was built for this purpose, starting with a chosen vertex or set of vertices, traces significant flows of Bitcoins over time.

Case Study – Part II: To demonstrate the tool, the Bitcoin theft described earlier is considered. The below Figure shows an annotated visualization produced using the specific tool. There are several interesting flows in the aftermath of the theft. The initial theft of a small volume of 1 BTC is immediately followed by the theft of 25.000 BTC. This is represented as a dotted black line between the relevant vertices, magnified in the left inset.

In the left inset, the Bitcoins are shuffled between a small number of accounts and then transferred back to the initial account. After this shuffling step, four significant outflows of Bitcoins that began at 19:49, 20:01, 20:13 and 20:55 have been identified. Of particular interest are the outflows that began at 20:55 (labeled as “1” in both insets) and 20:13 (labeled as “2” in both insets). These outflows pass through several subsequent accounts over a period of several hours. Flow 1 splits at the vertex labeled A in the right inset at 04:05 the day after the theft. Some of its Bitcoins rejoin Flow 2 at the vertex labeled B. This new combined flow is labeled as “3” in the right inset. The remaining Bitcoins from Flow 1 pass through several additional vertices in the next two days. This flow is labeled as “4” in the right inset.

On 16/06/2011 at approximately 13:37 there is an interesting observation. A small number of Bitcoins are transferred from Flow 3 to a heretofore unseen public key  $pk_1$ . Approximately seven minutes later, a small number of Bitcoins are transferred from Flow 3 to another unseen public key  $pk_2$ . Finally, there are two simultaneous transfers from Flow 4 to two more unseen public keys:  $pk_3$  and  $pk_4$ . These four public keys,  $pk_1$ ,  $pk_2$ ,  $pk_3$  and  $pk_4$  – which receive Bitcoins from two separate flows that split from each other two days earlier – are all contracted to the same user in our ancillary network. This user is represented as C in the below Figure.

The flow labeled as Y involves the movement of Bitcoins through thirty unique public keys in a very short period of time. At each step, a small number of Bitcoins (typically 30 BTC

which had a market value of approximately US\$500 at the time of the transactions) are siphoned off. The public keys that receive the small number of Bitcoins are typically represented by small blue vertices due to their low volume and degree [18]. On 20/06/2011 at 12:35, each of these public keys makes a transfer to a public key operated by the MyBitcoin service.

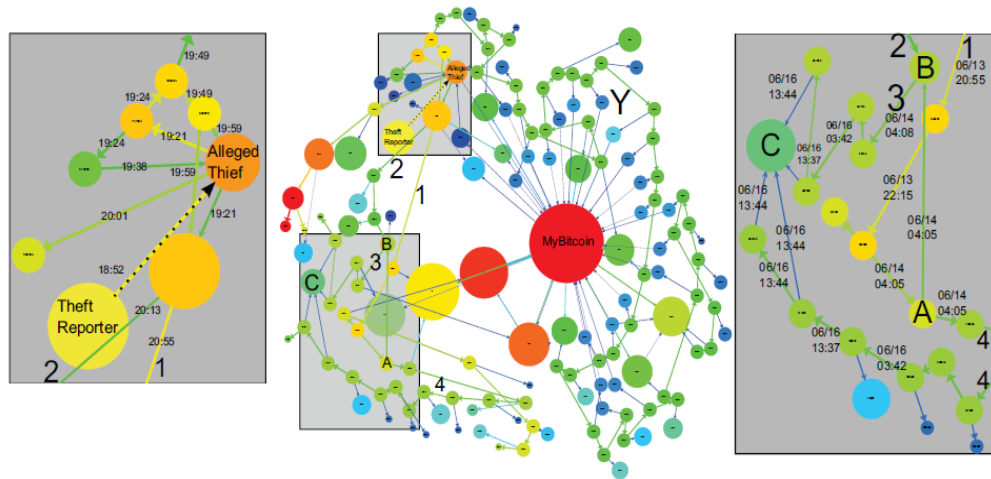


Figure 7: Tool representation

## 2.6 Research's conclusion

Through this research it is shown that it is possible to associate many public keys with each other and with external identifying information. Also, the activity of known users can be observed in detail. Large centralized services such as the exchanges and wallet services are capable of identifying and tracking considerable portions of user activity. It should be noted that even the technical members of the Bitcoin community have cautioned that strong anonymity is not a prominent design goal of the Bitcoin system. However, casual users need to be aware of this, especially when sending Bitcoins to users and organizations they would prefer not to be publicly associated with.

## 3. Cryptocurrencies and Illicit Activities

The security and anonymity offered by cryptocurrency transactions sometimes gives rise to illicit activities, including the sale of illegal goods, drugs and weapons, assassinations,

Ponzi schemes, money laundering, unlawful gambling and identity theft. Illicit activities come in many forms and businesses can practice in more than one illicit transaction at the same time. In simple words, cryptocurrency has enabled criminals to conduct traditional illicit crimes through an almost anonymous payment system, which helps to hide their activities from regulators.

Before further discussing the issue of cryptocurrency anonymity and how this is used for illicit activities, we want to discuss the concept of anonymity per se, and in particular anonymity as privacy and anonymity as a hidden identity. Anonymity as privacy relates mostly to customers who are concerned about potential intrusions in their personal life through institutions gathering lots of information for them. In response to these fears they tend to use cryptographic tools or other anonymizers so that they can browse the web and conduct purchases in an anonymized way. Anonymity as hidden identity relates mostly to criminals who require anonymity to conceal their real identity and avoid being tracked by the authorities [19]. Since no transaction can be absolutely anonymous, but rather very difficult to trace, criminals that use cryptocurrencies risk that their transactions are linked with each other.

Certain illicit activities are reviewed below where cryptocurrencies are commonly used as a facilitator. Even though the examples refer to every cryptocurrency, Bitcoin is currently the preferred denominator [20].

### 3.1 Money Laundering in the Crypto World

So, what can be defined as “money laundering”? Essentially, it is a process in which money is separated from the criminal activities where it was earned and mixed with funds from legitimate enterprises, such as small businesses where cash sales are common and then funneled back to the original criminal enterprise or source.

Bitcoins are famously used to launder money acquired through illicit activities due to the lack of “Know Your Client” (KYC) measures that traditional banking institutions implement. Unlicensed entities do not typically require any such data collection for reporting requirements. Indicative of this popularity [21], a selling exchange for bitcoins called “Local Bitcoins” allows buying or selling bitcoins using cash. The transaction fee on these platforms averages 10-15%, which is significantly higher than the 1-2% for licensed bitcoin exchangers. In essence, the transaction fee allows individuals with quantities of illicit

cash to anonymously convert the cash into bitcoins. Once the transaction is confirmed in the blockchain, the seller receives the currency. No KYC information is typically required for this type of transactions. The buyer can then successfully accomplish the placement phase of money laundering and introduce “clean” cash into the financial system.

Bitcoin ATMs are another way to launder money since they allow bitcoins to be purchased with cash or gift cards. Said ATMs either collect minimal KYC information, that remains unverified or no information at all. The first bitcoin ATM was introduced in 2014 [22]. As of November 2018, there are approximately 4,500 active ATMs in more than 70 countries around the world [23]. These ATMs charge a commission fee of 10-15% and sometimes are as simple to transact as inserting your phone number. While other ATMs require an identification, the information provided is rarely verified, which defeats the point of KYC in the first place.

To make things even easier for aspirational money launderers, bitcoin tumbling services can conceal the true source and destination of bitcoin transfers by dividing the transfer into smaller payments transacted at the same time. Tumblers, also named mixers or laundry services, obfuscate bitcoin transactions between parties to make them less identifiable by law enforcement and other users on the network. Since the blockchain contains bitcoin transactions, users desire to mask their transmissions of bitcoins through tumblers to facilitate money laundering [24]. Tumblers take multiple transactions and join them together for disbursement to payees through multiple senders. New cryptocurrencies, like Monero, have built-in tumbling services which increases the anonymity they offer to their users.

Finally, property exchanges work through intermediaries who purchase items on popular websites (e.g. Amazon). The customer creates a wishlist and then the user lists these items on the Purse marketplace. Customers receive the goods and pay similarly a 10-15% commission fee, as above. Neither Purse nor Amazon conducts KYC or AML programs, which similarly enhances the anonymity of the marketplaces.

A few well-known money laundering cases include the Bitcoin exchange “OKCoin” with hundreds of thousands of US dollars laundered as well as the case of “BitInstant”, where an estimated sum of more than \$1,000,000 was laundered for Silk Road market customers [1], [25]. Moreover, cryptocurrencies have advanced the operations of various malware families such as ransomware, with CryptoLocker [26] and CryptoWall [27] receiving

133,045.9961 BTC and 87,897.8510 BTC respectively. Cryptojacking, with JenkinsMiner [28] earning its operator over \$3,000,000 worth of Monero and crypto-stealing Trojans, such as CryptoShuffler [29] which stole hundreds of thousands of US dollars by targeting the contents of volatile memory.

Cash has traditionally been king for these types of enterprises because it is notoriously difficult to trace and because many criminal enterprises, particularly drug trafficking, theft and similar crimes deal in cash. However, cryptocurrencies are opening pathways for new methods of laundering funds on a scale that has global governments showing concern.

Other types of crimes money laundering “cover-ups”, include customs violations, tax violations and cybercrimes [30], making cryptocurrencies a convenient resource for “cleaning up” these funds.

### 3.2 Cryptomarkets and Drug Trafficking

A cryptomarket is usually defined as a marketplace that hosts multiple sellers or vendors, provides participants with anonymity via its location on the hidden (or dark) web and the uses cryptocurrencies for payment. Cryptomarkets are designed in a way that allows the trafficking of illicit products, predominantly drugs. This type of illicit substance trafficking rose to prominence with the creation of the Silk Road in 2011 [1]. In web interface and usability, the Silk Road very much resembled engines like the ebay and Amazon, based on peer to peer technologies related to encryption processes. In contrast to these popular search engines, Silk Road is part of the darknet [31]. Access to it necessitates a specific communication protocol, such as an onion routing used to hide a computer’s IP address. These cryptomarkets can facilitate the online trafficking of illicit goods through encrypted communications and financial transactions using cryptocurrencies (e.g. bitcoins).

The original Silk Road was launched in 2011 and was shut down by the US Federal Bureau of Investigation (FBI) in 2013. It was considered to provide a high level of security and anonymity, evidenced by the fact that out of hundreds of thousands of participants, only a small fraction was arrested following the take down of the cryptomarket. The website’s creator, the American Ross Ulbricht, was sentenced to life imprisonment without parole for a number of charges, including money laundering. Consequently, Silk Road 2.0 was taken down in 2014 following an international operation by police agencies from 17

different countries. Silk Road 3.0 opened again in mid-2016 revealing the longer-term limit to international police crackdowns. Apart from other factors, the limited experience of law enforcement officials also contributed to a less effective prosecution of the cryptomarket vendors.

In essence, cryptomarkets provide a new distribution channel for illegal substances for which the costs and benefits are still unknown. These new platforms allow for reputation building for vendors who provide illicit products and provide a relatively seamless experience since users avoid face to face interaction, which consequently reduces the chance for violent incidents. These benefits of using these platforms make them an attractive option for users looking to purchase illegal goods.

### 3.3 Extortion, Attacks and Terrorism Financing

Extortion schemes have increased as a result of cryptocurrencies' anonymity. It is desirable for those penetrating extortion due to its relevant anonymity and non-oversight by a central regulator. These crimes are being funded through anonymous payments in cryptocurrency where no physical or electronically traceable handoff occurs. The schemes include traditional extortion such as kidnapping and blackmail to higher tech schemes including malware, ransomware, and DDoS attacks. Another common scheme is to insert malware on a subject's servers that encrypts their data until a ransom is paid. Similarly, hackers can exploit a weakness in the individual's technology until a ransom is paid, or even take control of the distribution of an individual's funds. Another method whereby blockchain is exploited is by injecting arbitrary encoded data chunks (e.g. pictures) in non-standard Bitcoin transactions to infiltrate child exploitation material, then asking for ransomware.

For instance, in November 2015 three Greek banks were reportedly threatened with dire consequences by a group of cybercriminals called the Armada Collective unless they paid 'hundreds of thousands of Euros' in Bitcoin (they asked for 20,000 bitcoin from each bank [32] and, in November 2015, the hackers of the mobile telephone provider TalkTalk sought to extort £80,000, also in Bitcoin, in return for not publishing the company's hacked customer data [33]. These are just a few recent examples of a growing catalogue of criminal activity in which Bitcoin has been nominated as the preferred method of payment.



Also, cryptocurrencies have been used to sponsor nation-state attacks since a number of countries around the world are affected by the existence of contemporary hybrid-war strategies.

Finally, terrorism financing is a connected point to extortions, ransomware and nation-state cyber attacks. Often, the ransomware itself is destined to finance terrorism activities, whereas in other instances terrorist groups communicate via the dark web and exchange funds through bitcoins.

### 3.4 Ponzi Schemes/Pump and Dump

Cryptocurrency is volatile and it fluctuates in a similar way as the stock market- which is sometimes referred to as “pump and dump” [34]. A cryptocurrency Ponzi scheme works exactly like a traditional Ponzi scheme [35]. A falsely inflated rate of return is used to draw in investors since the more money that is invested earlier in the scam allows the scammers to continue to pay off their victims for longer before folding and disappearing. The longer period of operation creates a false sense of reliance for future investment, which subsequently attracts more investors.

Initial Coin Offerings are also used for exit scams [36] since the unregulated environment allows criminals to persuade their victims to buy large numbers of fake coins, subsequently disappearing with millions of dollars. Criminals may also use cryptocurrencies as a high yield investment or a Ponzi scheme.

As cryptocurrencies gain worldwide legitimacy, the aforementioned problems are only going to continue to grow.

## 4. Anonymity over Tor

Another interesting issue that should be investigated is the influence a browser which allows anonymous communication can have on the distribution of Bitcoin’s information. How secure is Bitcoin information via Tor and what can be derived from each transaction?

## 4.1 Cryptos and Tor

As previously stated, Bitcoin is a decentralized virtual currency and a P2P payment system in which coins are generated by miners and double spending [5] is prevented since each peer keeps a local copy of the constantly growing public ledger of all the previous transactions [37]. Though the original Bitcoin paper states that privacy in such a system may still be maintained, the recent findings disprove this. Anonymity and privacy of the plain Bitcoin protocol is also not claimed by the Bitcoin developers.

There are two independent problems:

a) the ability of the attacker to link transactions to the IP address of the user by studying connectivity and traffic of the peers and

b) the linkability of the user's pseudonyms and transactions in the public ledger achieved via transaction flow analysis.

At the same time as Bitcoin increases its user base and moves from mining and hoarding to the actual use as a currency and payment protocol in various on-line applications, there is a growing demand in more privacy among the Bitcoin users. Since IP address leakage is still possible, Bitcoin developers recommend to use third party anonymization tools like Tor or VPNs to solve the problem.

Anoncoin [38], BitTor [39], Torcoin [40], Stealthcoin [41] and others are alternative currencies which offer native support for Tor. There are also several other use cases for Tor in the Bitcoin ecosystem. For mobile payments, the SPV (simple payment verification) method is used for clients which cannot afford to hold the full 20 GB blockchain ledger. Since many Bitcoin clients are vulnerable to spoofing attacks which may result in double-spending, the current trend is to bundle them with Tor by default to avoid spoofing [42] and man-in-the-middle attacks. Tor can also be a solution for services and online shops that want to prevent DoS attacks against their public IP.

However, Tor is not a panacea and not all applications are anonymized equally well when combined with Tor. The biggest effort has been made so far on improving protection of the HTTP(S) protocol on top of Tor. Other protocols are not researched that well. There were several documented cases when application level leaked crucial user-identifying information. Moreover, there is only a limited number of applications which are studied well enough to be considered safe to use with Tor.

## 4.2 Bitcoins services

Bitcoin software does not explicitly divide its functionality between clients and servers. So, Bitcoin peers can be grouped into those which accept incoming connections (servers) and those which don't (clients), such as peers behind Network address translation (NAT) or firewalls. Bitcoin users connecting to the Bitcoin network through Tor or VPN do not accept incoming connections.

There are millions of reachable Bitcoin servers and even more Bitcoin clients. By default, Bitcoin peers (both clients and servers) try to maintain 8 outgoing connections to other peers in the network. If any of the 8 outgoing connections drop, a Bitcoin peer tries to replace them with new connections. A Bitcoin client can only establish a connection to a Bitcoin server. By default, a server can accept up to 117 incoming connections. If this limit is reached, all new connections are dropped.

1) Bitcoin anti-DoS protection: As an anti-DoS protection, Bitcoin peers implement a reputation-based protocol with each node keeping a penalty score for every other Bitcoin peer (identified by its IP address). Whenever a malformed message is sent to the node, the latter increases the penalty score (different messages incur different penalties) of the sender and bans the "misbehaving" IP address for 24 hours when the penalty reaches the value of 100.

2) Bitcoin peers as Tor hidden services: Tor hidden services are service-agnostic in the sense that any TCP-based service can be made available as a Tor hidden service. This is used by Bitcoin which recognizes three types of addresses: IPv4, IPv6, and OnionCat [43]. Onioncat address format is a way to represent an onion address as an IPv6 address: the first 6 bytes of an OnionCat address are fixed and set to FD87:D87E:EB43 and the other 10 bytes are the hex version of the onion address (i.e. base32 decoded onion address after removing the ".onion" part).

3) Bitcoin peer discovery and bootstrapping: Bitcoin implements several mechanisms for peer discovery and bootstrapping. First, each Bitcoin peer keeps a database of IP addresses of peers previously seen in the network. This database survives between Bitcoin client restarts. This is done by dumping the database to the hard drive every 15 minutes and on exit (as we will see later this facilitates setting a cookie on the user's computer). Bitcoin peers periodically broadcast their addresses in the network. In addition, peers can request addresses from each other using GETADDR messages and advertise addresses using ADDR messages.

If Tor is not used, when a Bitcoin client starts, it first tries to populate its address database by resolving 6 hard-coded hostnames. If Tor is used, Bitcoin does not explicitly ask Tor to resolve them but rather asks it to establish connections to these hostnames.

If Tor is not used, the addresses for outgoing connections are taken from the addresses database only. In case Tor is used, every second connection is established to a DNS hostname. These DNS hostnames are called “oneshots” and once the client establishes a connection to such a hostname it requests a bunch of addresses from it and then disconnects and never tries to connect to it again. As a fallback, if no addresses can be found at all, after 60 seconds of running the Bitcoin client uses a list of 600 hard-coded IP addresses.

Bitcoin nodes recognize three types of addresses: IPv4, IPv6, and OnionCat [43]. For each type of addresses, the peer maintains a state variable indicating if the Bitcoin node is capable of using such address type. These state variables become important when using Tor: the only address type which is accepted from other peers is OnionCat type. Curiously, this results in that all IPv4 and IPv6 addresses obtained from oneshots are dropped and the client uses its original database. The opposite case also holds: if Tor is not used, onion addresses are not stored in the address database.

Finally, each address is accompanied by a timestamp which determines its freshness.

4) Choosing outgoing connections: For each address in the address database, a Bitcoin peer maintains statistics which among other things includes when the address was last seen in the network, if a connection to this address was ever established before and the timestamp of such a connection. All addresses in the database are distributed between buckets. There are 256 buckets for “new” addresses (addresses to which the Bitcoin client has never established a connection) and 64 for “tried” addresses (addresses to which there was at least one successful connection). Each bucket can have at most 64 entries (which means that there can be at most 20480 addresses in the database). When a peer establishes outgoing connections, it chooses an address from “tried” buckets with probability  $p = 0.9 - 0.1n$ , where  $n$  is the number of already established outgoing connections. If an address is advertised frequently enough it can be put into up to 4 different “new” buckets. This obviously increases its chances to be selected by a user and to be transferred to a “tried” bucket.

### 4.3 How Tor works

Tor is the most popular low-latency anonymity network based on ideas of onion routing and telescoping path-building design [44]. When a user wants to connect to an Internet server while keeping his IP address in secret from the server, he chooses a path consisting of three Tor relays (called Guard, Middle and Exit), builds a circuit and negotiates symmetric keys with each one of them using the telescoping technique. Before sending a message to the server, the user encrypts it using the negotiated keys. While the message travels along the circuit, each relay strips off its layer of encryption. In this way the message arrives at the final destination in its original form and each party knows only the previous and the next hop.

Tor tries hard to achieve low traffic latency to provide a good user experience, thus sacrificing some anonymity for performance. To keep latency low and network throughput high, Tor relays do not delay incoming messages and do not use padding. This makes Tor susceptible to traffic confirmation attacks: if an attacker is able to sniff both ends of a communication, he is able to confirm that the user communicated with the server. If the first hop of a circuit is chosen at random then the probability that a malicious node will be chosen as the first hop (and thus will know the IP address of the user) converges to one with the number of circuits. Due to this, each user has a set of three Guard nodes. When a user builds a circuit the first hop is chosen from the set of Guard nodes.

The list of all Tor relays is assembled and distributed in the so-called consensus document by nine trusted Tor authorities. For the purpose of traffic balancing, the bandwidth of each relay is measured and reported. A user chooses relays for his circuits with probability proportional to the relays' weights listed in the consensus. Each relay in the consensus is identified by his fingerprint (or ID) which is the SHA-1 hash of its public key.

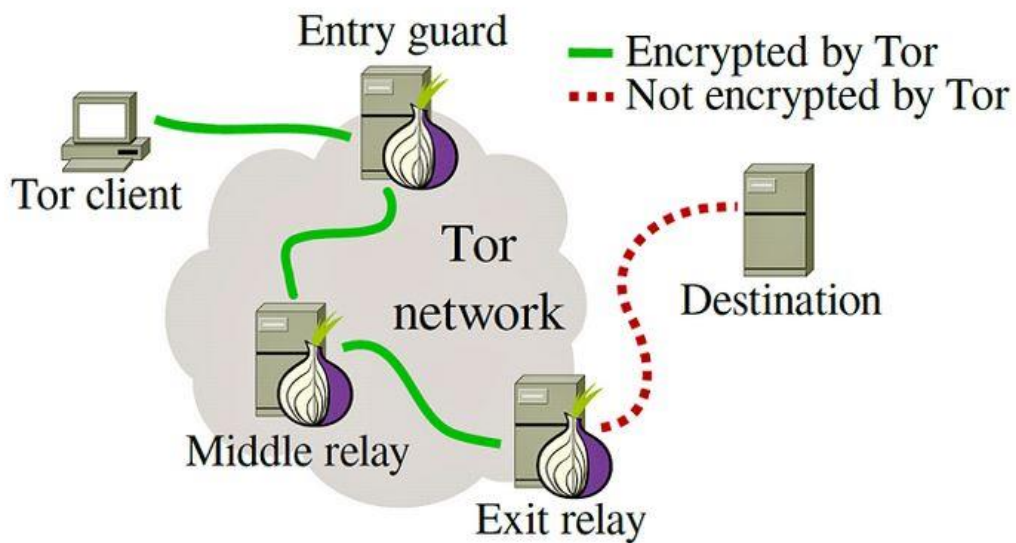


Figure 8: Tor Guards

1) Tor stream timeout policy: Tor provides SOCKS interface for applications willing to connect to the Internet anonymously. Each connection to the SOCKS port by an application is called a stream. For each new stream Tor tries to attach it either to an existing circuit or to a newly built one. It then sends a BEGIN cell down the circuit to the corresponding Exit node asking it to establish a connection to the server requested by the application. In order to improve user's quality of service, if Tor does not receive a reply from the Exit node within 10 or 15 seconds, it drops the circuit and tries another one. If none of the circuits worked for the stream during 2 minutes, Tor gives up on it and sends a SOCKS general failure error message.

2) Tor Exit policy: In order to access a Web resource anonymously through a Tor circuit, the Exit relay (the final relay in the circuit) should allow establishing connections outside the Tor network. This makes Exit relay operators open to numerous abuses. In order to make their life easier, Tor allows them to specify an Exit Policy: a list of IP addresses and ports to which the Exit node is willing to establish connections and which destination are prohibited. When a client establishes a circuit, he chooses only those Exit nodes which allow connections to the corresponding IP addresses and port ranges.

3) Tor Hidden Services: Tor is mostly known for its ability to provide anonymity for clients accessing Internet services. Tor Hidden Services are another feature of Tor which enables responder anonymity: a service can be contacted by clients without revealing its physical location. In order to achieve this, a client and the hidden service choose at random and connect to a Tor relay (rendezvous point) and forward all the data through it.

In more details:

1) The hidden service generates a public key and chooses at random a small number of Tor relays (typically three) which become its introduction points. The service maintains permanent connection to these relays.

2) It then generates an HS descriptor which contains the public key and the list of introduction points and

3) Publishes it at 6 different Tor relays having HSDir flag. These are called responsible HS directories. The choice of responsible HS directories is deterministic and depends on the hash of the hidden service's public key and current day.

4) Introduction points are instructed by the hidden service to forward connection requests from clients. The base32 encoding of the hash of the hidden service's public key (onion address) is then communicated to clients by conventional means (blog post, e-mail, etc.).

When a client decides to connect to the hidden service, he:

1) Determines the list of the responsible HS directories using the onion address and downloads the HS descriptor.

2) Chooses a rendezvous point at random.

3) Communicates the ID of the rendezvous point to the hidden service's introduction points which then forward it to the hidden service.

When the hidden service receives the ID of the rendezvous point, it establishes a connection to it and the data transfer between the service and the client can start. All communications between the client and the rendezvous point, between the service and the rendezvous point and between the service and the introduction points are established over three-hop circuits. This hides the location of the hidden service and its clients both from each other and from external observer.

The hidden service or a client can determine the fingerprints of the responsible directories as follows. They first take all Tor relays which have HSDir flag in the consensus and sort their fingerprints in lexicographical order. Second, they compute the descriptor ID's of the hidden service which is the SHA-1 hash of a value composed of the following items: public key of the hidden service, current day, and replica (which can be 0 or 1). What is important is that:

- a) The ID changes every 24 hours,
- b) There are two replicas of the ID,
- c) They find the place in the sorted list of the fingerprints for the computed ID and take the next three relays' fingerprints (thus having 6 fingerprints in total since there are two replicas).

#### 4.4 Possible Attacks

The possible issues throughout this whole process are described in the below sections.

##### 4.4.1 Exploitation

By exploiting Bitcoin's anti-DoS protection, a low-resource attacker can force users which decide to connect to the Bitcoin network through Tor to connect exclusively through her Tor Exit nodes or to her Bitcoin peers, totally isolating the client from the rest of the Bitcoin P2P network [45]. This means that combining Tor with Bitcoin may have serious security implications for the users:

- 1) they are exposed to attacks in which an attacker controls which Bitcoin blocks and transactions the users are aware of and
- 2) they do not get the expected level of anonymity.

The main building blocks of the attack are: Bitcoin's reputation-based anti-Dos protection, Tor's stream management policy, the fact that connections between Bitcoin peers are not authenticated. The attack consists of four steps:



1) Injecting a number of Bitcoin peers to the network. In order to comply with Bitcoin's limitation "one peer per IP address", the attacker should obtain a large number of IP addresses. The easiest way would be to rent IP addresses on per hour basis. The market value is 1 cents per hour per IP address. The important note is that the obtained IP addresses will not be involved in any abusive activity (like sending spam or DoS attacks) which makes this part of the attack undetectable.

2) Periodically advertising the newly injected peers in the network so that they are included into the maximum possible number of buckets at the client side. The attacker is interested in that his Bitcoin peers are chosen by Bitcoin clients as frequently as possible. In order to increase the chances for her peers to be included into "tried" buckets, the attacker should advertise the addresses of her peers as frequently as possible. This mechanism would allow the attacker to inject less malicious peers. Note also that address advertisement is not logged by default and thus requires special monitoring to be noticed.

3) Injecting some number of medium-bandwidth Tor Exit relays. During this step the attacker runs a number of Exit Tor nodes. In order to get Exit flag from the Tor authorities, an attacker's Exit node should allow outgoing connections to any two ports out of ports 80, 443, or 6667. Such an open Exit policy might not be what a stealthy attacker wants. Fortunately for the attacker, he can provide incorrect information about his exit policy in his descriptor and thus have Exit flag while in reality providing access to port 8333 only. The attacker can do even better, and dynamically change the exit policy of his relays so that only connections to specific Bitcoin peers are allowed.

4) Making non-attacker's Bitcoin peers ban non-attacker's Tor Exit nodes. In this phase, the attacker exploits the built-in Bitcoin anti-DoS protection. The attacker chooses a non-attacker's Bitcoin peer and a non-attacker's Tor Exit, builds a circuit through this Exit node and sends a malformed message to the chosen Bitcoin peer (e.g. a malformed coinbase transaction which is 60 bytes in size and which causes the immediate ban for 24 hours). As soon as the Bitcoin peer receives such a message, it analyses the sender's IP address which obviously belongs to the Tor Exit node chosen by the attacker. The Bitcoin peer then marks this IP address as misbehaving for 24 hours. If a legitimate client then tries to connect to the same Bitcoin peer over the banned Exit node, his connection will be rejected. The attacker repeats this step for all non-attacker's Bitcoin peers and each non-attacker's Tor Exit node. This results in that a legitimate Bitcoin user is only able to connect

to Bitcoin over Tor if he chooses either one of the attacker's peers or establishes a circuit through an attacker's Exit node.

#### 4.4.2 Defeating onion peers

Bitcoin peers can be made reachable as Tor hidden services. Banning Tor Exit nodes will obviously not prevent Bitcoin clients from connecting to such peers. Nonetheless, observations show that this case can also be defeated by the attacker.

First, the current design of Tor Hidden Services allows a low-resource attacker to DoS a hidden service of his choice (this technique is called black-holing of hidden services). Before a client can contact a hidden service, he needs to download the corresponding descriptor from one of the six responsible hidden service directories. These directories are chosen from the whole set of Tor relays in a deterministic way based on the onion address and current day.

The attacker needs to inject six malicious relays that would become responsible directories. In other words, he needs to find the right public keys with fingerprints which would be in-between the descriptor IDs of the hidden service and the fingerprint of the currently first responsible hidden service directory. This can become a problem though for a large number of hidden services: for each hidden service the attacker needs to run at least 6 Tor relays for at least 25 hours, 2 relays per IP address.

Fortunately, for the attacker, the fraction of Bitcoin peers available as Tor hidden services is quite small which results in:

- 1) a very small probability for a client to choose a peer available as a hidden service and
- 2) thus, making black-holing of existing Bitcoin hidden services practical.

Second, the attacker can at almost no cost inject a large number of Bitcoin peers available as Tor hidden services. It requires running only one bitcoind instance and binding it with as many onion addresses as needed. Thus, users will more likely connect to attacker controlled "onion" peers.

Third, when running Bitcoin without Tor, onion addresses received from peers are silently dropped. Thus, one can only obtain OnionCat addresses by either connecting to an IPv4 or IPv6 reachable peers through a proxy or by specifying an onion address in the command line.

#### 4.4.3 Attack vectors

The technique described in this section allows an attacker to direct all Bitcoin-over-Tor traffic through servers which are under his control.

**Traffic confirmation attack** : First, it becomes much cheaper to mount a successful traffic confirmation attack. In traffic confirmation attacks, the attacker controls a fraction of Guard and Exit nodes. The attacker sees that one of her exit nodes is requested to access a particular (e.g. censored) web-site and the attacker is interested in finding out the user who made this request. The attacker sends a traffic signature down the corresponding circuit. If the attacker was lucky and the user chose one of her Guard nodes, the attacker will see the traffic signature going through this Guard to the target user. This reveals the user's IP address.

The success probability of the attack is computed as the product of two factors:

- 1) the probability for the user to choose an attacker's Guard and
- 2) the probability for the user to choose an attacker's Exit.

**Revealing Guard nodes.** In case the attacker does not control the user's Guard node, he may try to find this Guard. We assume that the attacker controls a fraction of middle nodes. As before, the attacker would send a traffic signature down the circuit and if none of the attacker's middle nodes detects this signature, the attacker drops the circuit. This will force the user to build another circuit. After some number of circuit tries, one of the attacker's middle nodes will finally be chosen. This middle node will know the user's Guard node. The re-identification of the user between different circuits is possible e.g. using the fingerprinting technique. Revealing the guards does not immediately allow an attacker to reveal the location of the user but gives him the next point of attack. Given that guard nodes are valid for more than a month, this may be sufficient to mount a legal attack to recover traffic meta-data for the guard node, depending on the jurisdiction the guard node is located in.

**Linking different bitcoin addresses.** Even without knowing the user's IP, the attacker can link together user's transactions regardless of pseudonyms used.

**Possibility of double spending.** Finally, after successfully mounting the described attack, the attacker controls the connectivity to the Bitcoin network for users which chose to use Tor which increases the success rate of double-spend attacks. In addition, the attacker can defer transactions and blocks and send dead forks. In collusion with a powerful mining pool (for example 10-20% of total Bitcoin mining capacity), the attacker can create fake blocks. This enables additional possibilities for double spending, however to make this relevant the amount should exceed what such miner would be able to mine in the real Bitcoin network. Also, complete alternative Bitcoin reality for all the users who access Bitcoin solely through Tor is possible. This however would come at a cost of 5-10 times slower confirmations, which after some time can be detected by the wallet software.

#### 4.4.5 Sybil Attacks on Bitcoin

As previously mentioned, a client needs to connect directly to one of the attacker's nodes in order to reveal his IP address so that an attacker can de-anonymize his previous transactions done over Tor. Bitcoin as a peer-to-peer network is vulnerable to Sybil attacks and just operating many Bitcoin servers means that a client will sooner or later choose an entry node controlled by the attacker (i.e. in some number of sessions). However, running too many servers can be. Fortunately for the attacker, there are a couple of ways to prevent Bitcoin clients from using non-attacker's Bitcoin servers (and choose an attacker's one instead).

#### 4.4.6 Attacks Description

Below a few further attacks related to user anonymity are described.

##### Exhausting connections limit

As stated earlier, by default a Bitcoin server accepts up to 117 connections. Once this limit is reached, all new incoming connections are dropped. At the same time, a Bitcoin server neither checks if some of these connections come from the same IP address, nor forces clients to provide proof-of-work. As a result, a low-resource attacker can establish many connections to all but his Bitcoin servers and occupy all free connection slots. If a client connects directly to a Bitcoin server connection slots of which are occupied, the

connection will be dropped immediately, thus the client will soon end up connecting to a malicious peer.

#### Port poisoning attack

Peer addresses are of the following form (IP, PORT). However, when a client decides to add a received address to the database, he does not take the port number into account. For example, assume a client receives an address (IP0, PORT1) and there is already an entry in the client's database (IP0, PORT0). In such case the client will keep (IP0, PORT0) and will not store (IP0, PORT1). The attacker can use this fact to flood with clients with addresses of legitimate Bitcoin servers but wrong port numbers. If the attacker is the first to send such addresses, the client will not be able to connect to legitimate nodes.

#### 4.5 Conclusion

The truth about Tor is that it facilitates a growing underground marketplace that sophisticated criminals use to traffic drugs, stolen identities, child pornography and other illicit products and services. Attackers can link together user's transactions regardless of pseudonyms used, control which Bitcoin blocks and transactions are relayed to which user and delay or discard user's transaction and blocks. So, with the "untraceable" cryptocurrencies as the primary means of payment, close cooperation between law enforcement, financial institutions and regulators around the world could be useful, under certain circumstances, to control all this nefarious activity.

## 5. Bitcoin Forensics

Very little research has been dedicated to what specific forensic artifacts are left on a user's system as a result of a Bitcoin transaction. The below research of Michael Doran [46] shows the artifacts of a thorough investigation regarding the memory, the disk and the file locations of Bitcoin's "fingerprint".

### 5.1 Wallets and Transactions

A Bitcoin wallet should be downloaded at first in order to begin conducting Bitcoin transactions. The Bitcoin wallet can show the total balance of all Bitcoins it controls and let a user pay a specified amount to a specific person, just like a physical wallet. Once the wallet is installed and configured, an address is generated which is similar to an e-mail or physical address, since it provides other users a numerical location to which they can send Bitcoins. In addition, the wallet contains a user's private key, which allows the spending of the Bitcoins, which are located in the blockchain.

After concluding the Bitcoin wallet setup, a user is supposed to have Bitcoins in order to conduct transactions. At the moment, there are four methods to acquire Bitcoins:

- 1) as **payment** for goods or services,
- 2) as **purchase** of coins through a Bitcoin exchange,
- 3) as an **exchange** with another user or
- 4) **earning** the coins through competitive mining.

When a user has successfully obtained Bitcoins through purchase, trade or by mining, the user's Bitcoins remain in their worker account until they are transferred to their individual Bitcoin wallet. When a user wishes to conduct a transaction, three pieces of information are required:

- 1) An **input** - the record of which Bitcoin address was used to send Bitcoins to the user.
- 2) An **amount** - the number of Bitcoins that the user is sending to another user.
- 3) An **output** - the address of the recipient of the Bitcoins to be sent.

In order for a person to send the Bitcoins to an intended user and complete a transaction, the person needs to have a Bitcoin address. This address is automatically generated when the Bitcoin wallet software is installed and a private key is generated. A private key serves as a cryptographic signature that validates a user's right to send Bitcoins from a specific wallet. If a user is utilizing a software wallet, the private key is stored on the user's computer, whereas if the user makes use of a web-based wallet, the private key is stored on a separate server. With the addresses of the sender and the recipient, the amount and the private key, the user can then conduct a Bitcoin transaction. The user's private key signs a message with the input, amount and output of Bitcoins before it is sent from their Bitcoin wallet out to the wider Bitcoin network. There the transaction is placed on the transaction block where it is eventually verified by Bitcoin miners.

A series of electronic signatures is required when each owner transfers the coin to the next owner. These signatures are unique to each owner and are created by digitally signing the hash of the previous transaction as well as the public key of the next owner. The signatures will then be added to the end of the coin, which provides a payee with a visual representation of the chain of ownership.

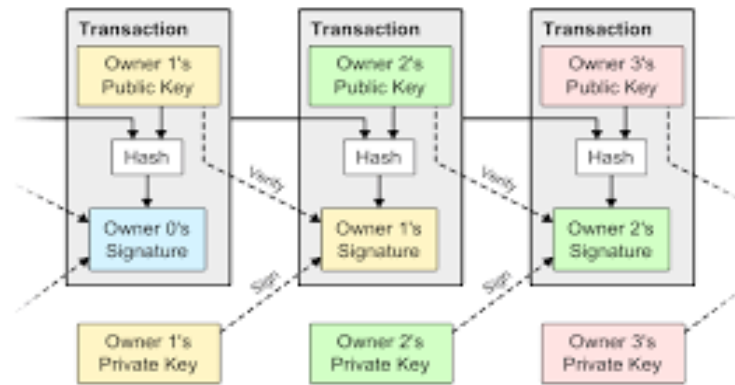


Figure 9: Chain of signatures of signatures associated with a Bitcoin transaction as it progresses from one owner to the next

Since all the Bitcoin information is transparent, information concerning the Bitcoin money supply is readily available on the block chain for anybody to verify and use in real-time. Thus, if an individual wanted to verify any of their transactions or the signatures associated with those specific transactions, they could visit a website such as blockexplorer [45] and conduct a search based on the block number, address, block hash, transaction hash or public key. Let's say that we conduct a search for the Bitcoin address of 17vPdTfLEEtFnpUZK2BUZuGBzyKbBx4iwF, the below six-column ledger would appear.

ab41036adae8ac22f43d43706d9b6478d15d889397b4bbe7d8aa3dc8e068cd6d		mined May 9, 2014 12:31:46 AM	
1GFzQ22XKCyKumMaz8C6MoUXA4CuiD3G6o	2.53379155 BTC	1PvMVpiiRUogoADd3kUUGE6NfcPKcfr1ok	6.41619655 BTC (S)
1CxL4giPs7Z8mkoTAmAFEj1sfzBaJ1YqDh	0.4894 BTC		
17vPdTfLEEtFnpUZK2BUZuGBzyKbBx4iwF	1.49644 BTC		
1LMPftnMm9uHW9ke4WcRSzLue6ZAp31tyg	1.76996 BTC		
12AxJEA5ERSPeTvgjYhZRFdxSMHJRqG9m2	0.126705 BTC		
FEE: 0.0001 BTC		302796 CONFIRMATIONS	6.41619655 BTC

Figure 10: Transaction Ledger

The ledger, in general, provides details regarding the transaction hash, the block that the transaction appeared in (including date and time), the amount, the type, who sent the Bitcoin, who received the Bitcoin and finally the balance available to the Bitcoin address.

## 5.2 Bitcoin artifacts

The Bitcoin wallet, which is downloaded as software, is similar to that of a physical wallet on the Bitcoin network. The user can spend the Bitcoins allocated to it as well as see the total balance of all Bitcoins it controls and pay a specific amount to a specific person, just like a real wallet.

Although there are numerous Bitcoin wallet software applications out on the market, the most notable is Bitcoin-Qt [47] because it is the original Bitcoin P2P open source software created by the creator of Bitcoin, Satoshi Nakamoto. It is not only a Bitcoin wallet, but it also contains the public ledger that lists every Bitcoin transaction in the system.

When examining a hard drive image from a suspect machine that was using Bitcoin-Qt, the folder structure will consist of three folders:

- 1) **Blocks** - This subdirectory contains blockchain data, a “blk.dat” file and a “blocks/index” subdirectory. The “blk.dat” stores actual Bitcoin blocks, in network format, which is dumped to disk in raw format. These enable for the re-scanning of missing transactions in a wallet, reorganizing to a different part of the chain, and serving the block data to other nodes that are synchronizing. The “blocks/index subdirectory” is a database that contains metadata about all known blocks and where to find them on the disk. Without this, finding a block would be very slow.
- 2) **Database** - This subdirectory contains database journaling files and
- 3) **Chainstate** - This subdirectory is a database with a compact representation of all currently unspent transactions and some metadata about where the transactions originated.

Five additional files of Bitcoin-Qt are:

- 1) **lock** - The database-locking file,
- 2) **db.log** – The database file,
- 3) **debug.log** – The Bitcoin’s extensive logging file,
- 4) **peers.dat** -The storage file for peer information to make a reconnection easier. It also utilizes a Bitcoin specific file format, which is unrelated to any other database system and



- 5) **wallet.dat** – The file for storage of keys, transactions, metadata, and options relating to Bitcoin.

Each file folder and file have their specific function and each offers specific forensic artifacts and information that can be utilized in the course of an investigation.

## 5.3 Investigation

How an investigation can be designed and what can be extracted from it?

### 5.3.1 Building a case

The content of a case as well as the knowledge, experience, expertise, thoroughness and the curiosity of the investigator in charge of the case defined its success. In addition to a well-rounded investigator, the success of a digital case rests on a foundational model that provides phases by which the investigator can progress through. The Investigation Process for Digital Forensic Science model [48] is the foundation for a successful digital investigation.

According to this model, there are six key phases during an investigation:

1. Identification
2. Preparation
3. Collection
4. Examination
5. Analysis
6. Presentation

However, the forensic artifacts of Bitcoin are more difficult than the average case due to the technology that Nakamoto implemented to keep the transactions pseudonymous. Because of this anonymity, particular pieces of evidence are more difficult to obtain and interpret. Still, a successful Bitcoin investigation is possible by escalating through the phases of the Investigation Process for Digital Forensic Science. At the moment, the focus centers on the Collection, Examination and the Analysis phases as they pertain to the forensic artifacts of a Bitcoin case.

In the Collection phase, the investigator needs to search for, document and collect any object or data that could potentially contain digital evidence. Since Bitcoin transactions occur via a network connection, an investigator should seize any physical object that can connect to the Internet. These objects include cell phones, PDAs, laptops, tablets, desktop

computers, or iPods. If during the Identification and Preservation phases it is determined that, the suspect's computer is on, it is imperative that the investigator can capture the system's physical memory (RAM). Many types of evidence may be available in volatile memory relating to Bitcoin. These types of evidence include:

- Running Bitcoin processes and services
- System information
- Information about logged in users
- Registry information
- Remnants of chats, communications in social networks and Bitcoin forums
- Recent Bitcoin web browsing activities
- Recent communications via webmail systems involving Bitcoin
- Information from cloud services
- Decryption keys for encrypted volumes mounted at the time of the capture
- Running Bitcoin malware/Trojans

Upon collecting the evidence, either physically or through extraction or imaging, the investigator can now begin the process of examining the data and assigning the level of importance of each individual piece. Although the Bitcoin artifacts reside on the suspect's hard drive and can be recovered using robust forensic tools, Internet Evidence Finder [49] permits the investigator to view just the Bitcoin artifacts.

Two different options are available that enable an investigator to recover the Bitcoin evidentiary artifacts utilizing Internet Evidence Finder. The first option is that the investigator can export the entire Bitcoin file folder from the suspect's drive and have Internet Evidence Finder analyze just that folder for Bitcoin artifacts. The second option is that the investigator can point Internet Evidence Finder at the entire image of the suspect's drive. The program will return not only the Bitcoin artifacts, but also Internet and chat history, e-mail and web searches to name a few.

In either option, the Bitcoin artifacts recovered provide a solid base for an investigator to build a case on. Because a majority of the user's activity involving Bitcoin resides within their respective Bitcoin wallets, a majority of the forensic artifacts are going to be located in the "wallet.dat" file. Internet Evidence Finder will recover the "wallet.dat" file and present the addresses from a Bitcoin wallet, as well as queries to the Bitcoin network from log files created by the Bitcoin client software in a user-friendly format.

In addition to the "wallet.dat" file, the investigator can examine the chain state

subdirectory to view all currently unspent transactions. These transactions, with corresponding addresses, could then be compared to the addresses recovered in the “wallet.dat” file as well as those found on the blockchain. By taking the recovered addresses, queries and information pertaining to the unspent transactions an investigator can slowly begin to piece together a case that could develop leads to not only other potential suspects or victims, but also open doors to other potential investigations.

### 5.3.2 Bitcoin forensic artifact examination

An experiment was designed in order to provide a visual representation of the functionality of Bitcoin and the various forensic artifacts that the software application leaves on a suspect system. The experiment utilized a designated computer with a fresh installation of Windows 7 Professional, Multibit [50], Bitcoin-Qt [47], Bitminter [51] and a basic USB ASIC [52] Bitcoin mining rig [53].

A Bitcoin mining rig is typically a computer system used for mining bitcoins. The rig might be a dedicated miner built specifically for mining, or it could be a computer used to mine only on a part-time basis. An ASIC, or Application Specific Integrated Chip, is a microchip designed for a special application, such as Bitcoin mining. After ensuring that the default settings are set with the installation of both the Multibit and Bitcoin-Qt wallet software applications and creating an account with the Bitminter mining pool, the system mined Bitcoins for approximately a month, during which time various transactions were made in order to place evidentiary artifacts inside of the Bitcoin wallets.

At the conclusion of the testing process, an image of the system’s RAM and hard drive were examined with EnCase 6.19.9 [54] and Internet Evidence Finder 6.1 [55]. The goal of the examination is to see the interaction between the Bitcoin mining software and wallet, with the operating system, registry and RAM.

### 5.3.3 Hardware Setup

The below listed items are the specific items of hardware that were used during the experiment. The 120 GB hard drive was wiped and a fresh installation of Windows 7 was installed to ensure a clean experimental environment. The individual ASIC Mining drives were individually plugged in to the USB hub that was then plugged in to the Gateway laptop via USB connection.

- Gateway laptop ML6720 with power supply
- 120 GB Western Digital hard drive

- (4) USB ASIC Mining drives
- 7 port USB hub
- USB powered cooling fan
- 32 GB USB thumb drive

#### 5.3.4 Tools

Throughout the experiment, several tools were utilized in order to maintain a running Bitcoin mining computer as well as populate the system's file system and registry with Bitcoin evidentiary artifacts. Each of the tools utilized in the experiment had a specific purpose and were chosen based on their platform design and ease of use.

- **Bitminter** is a Bitcoin mining pool that enables a user to mine for Bitcoins. It provides the user with a graphical user interface that enables the user to control every facet of their Bitcoin mining experience.
- **Multibit** is a lightweight "thin client" Bitcoin wallet for Windows, MacOS and Linux based on bitcoinj [56], which is an open source Bitcoin client library built using Java and the Bitcoin network protocol. Its main advantages include support for opening multiple wallet simultaneously, and not requiring the user to download the entire block chain.
- **Bitcoin-Qt** is the Bitcoin wallet software developed by Wladimir J. van der Laan, which is based on the original source code of Satoshi Nakamoto. It is a desktop wallet system and contains the public ledger that lists every Bitcoin transaction in the system.
- **Tableau Imager 3.1.2** [57] is a forensic imaging tool used to acquire a bit-for-bit copy of a piece of media. It supports Encase .E01, .DD, and .DMG file formats and can customize the destination path and file name conventions with the use of variables including date/time, drive serial number and model number. It also has error recovery and reporting and conducts the calculation of MD5 and SHA-1 hash values at the conclusion of the imaging process.
- **EnCase 6.19.7** is a forensic program designed for forensic examiners and trained investigators who are conducting full forensic examinations on any type of digital media. EnCase allows the forensic examiner to acquire data rapidly from various device types and perform an in-depth forensic analysis of the media. At the conclusion of an analysis, it provides the forensic examiner with the ability

to produce comprehensive reports as well as maintain the integrity of the evidence in a format that is presentable and accepted by the courts.

- **Internet Evidence Finder 6.2.3** is a forensic program designed for forensic examiners and trained investigators who are conducting full forensic examinations of Windows and Mac computers. Specifically, Internet Evidence Finder recovers data from social networking sites, instant messenger chats, P2P file sharing apps, mobile backups, webmail, web browser history, pictures and videos.
- **Winen.exe** [58] is a RAM acquisition tool that ships with the forensic software EnCase. It can run as a command line tool or from a configuration file. The tool collects RAM and places the collected information into an .E01 file that can be stored on an external drive.

## 5.4 Results

The testing environment was the first to be prepared. This phase included a clean install of the host operating system, with all drivers and updates installed.

### 5.4.1 Overview

The test system was a Gateway ML6720 laptop computer running Windows 7 Professional Service Pack 1, Build 7601, 32 bits. The processor was an Intel Pentium T2310 running at 1.46 GHz. The system had 1 GB RAM and the system time zone was set to Central Standard Time and was verified through the use of an Apple iPhone utilizing Sprint's cellular network.

The second phase of testing involved configuring the test system to mine and interact with Bitcoins. The first step in this process was to install the Bitcoin wallets that would house the Bitcoin transactions, addresses and private keys utilized during the testing. The Multibit Bitcoin wallet application was downloaded in the test system's Internet Explorer web browser and saved in the Downloads folder of the test system. The Multibit application was located in the Downloads folder and installed by double-clicking on the "multibit-0.5.16-windowssetup.exe" file. This action installed the application with the default settings in the following location "C:\Program Files\MultiBit-0.5.16." Upon successful installation of the Multibit Bitcoin wallet, the application was opened and the Bitcoin address associated with the wallet was "1FdhjMV8s2kzfAdU6TXVS35xkCGcbxAiM6."

In addition to verifying the address of the Multibit wallet, the folder structure of

the installation was documented for reference when conducting further examination with EnCase and Internet Evidence Finder. The below figure depicts the folder structure of the Multibit wallet application on the test system.

Name	Date modified	Type	Size
log	1/22/2014 9:11 PM	File folder	
multibit-data	1/22/2014 9:58 PM	File folder	
multibit.checkpoints	1/22/2014 9:11 PM	CHECKPOINTS File	13 KB
multibit.info	3/2/2014 9:21 AM	INFO File	1 KB
multibit.properties	2/28/2014 4:42 PM	PROPERTIES File	1 KB
multibit.spvchain	1/22/2014 9:11 PM	SPVCHAIN File	626 KB
multibit.wallet	3/2/2014 9:21 AM	WALLET File	3 KB

Figure 11: Screenshot of the folder structure of Multibit installed on the test system.

In order to gain an understanding of the various artifacts resulting from different Bitcoin wallets, Bitcoin-Qt was downloaded and installed via Internet Explorer to the test system as an additional wallet software application. The Bitcoin-Qt application, “bitcoin-0.8.6-win32-setup.exe,” was installed with the default settings in the following location “C:\Program Files\Bitcoin-Qt-0.8.6.” After noting the Bitcoin address associated with the wallet as “14igLoRYLjmqc9H5ZSxWqBvdNT3Ro1QeUJ,” was then labeled as “Suspect” and saved within the Bitcoin-Qt wallet.

To verify the address of the Bitcoin-Qt wallet, the folder structure of the installation was documented for reference when conducting further examination with EnCase and Internet Evidence Finder. The below figure depicts the folder structure of the Bitcoin-Qt wallet application on the test system.

Name	Date modified	Type	Size
blocks	2/28/2014 6:38 AM	File folder	
chainstate	3/2/2014 9:24 AM	File folder	
database	3/2/2014 8:09 AM	File folder	
.lock	2/24/2014 7:19 PM	LOCK File	0 KB
db	2/24/2014 7:19 PM	Text Document	0 KB
debug	3/2/2014 8:05 AM	Text Document	1,090 KB
peers.dat	3/2/2014 9:24 AM	DAT File	970 KB
wallet.dat	3/2/2014 8:29 AM	DAT File	96 KB

Figure 12: Screenshot of the folder structure of Bitcoin-Qt installed on the test system

After installing and configuring both of the Bitcoin wallets, an account was created utilizing the G-Mail address of “forensicminer@gmail.com” at website

"https://bitminter.com." The account was created in order to run the software application from the test system and to store pertinent information such as Bitcoin addresses and worker identities. Upon signing on to the Bitminter mining pool for the first time, all of the default account settings were left. However, a "worker" was created and given the identifier of "1" serving as the sole worker performing mining from the test system in the Bitminter mining pool. The Bitminter application was not installed on to the test system; rather, the application was run from the website by clicking on the "Engine Start" button.

With the Multibit and Bitcoin-Qt wallets installed and the Bitminter account created, the Bitcoin mining rig was configured. For the testing environment, the rig consisted of four ASIC Block Erupters plugged in to a seven-port USB hub. An ASIC Block Erupter is a tool utilized to mine Bitcoins that uses an Application Specific Integrated Chip and mines at 330 mega hashes a second (MH/s).

All four of the ASIC miners were attached to a USB port on the hub, in addition to a USB cooling fan. The fan kept the ASIC miners cool, increasing performance and preventing damage due to the overwhelming amount of work that each miner was doing. Upon plugging the hub in for the first time, the test system did not initially recognize the ASIC miners because of the lack of the "CP210x USB to UART Bridge VCP Drivers." The "CP210x USB to UART Bridge VCP Drivers" were downloaded via the test system's Internet Explorer web browser from the Silicon Labs website [59]. Upon successful installation, the "Devices and Printers" section of the test system's Control Panel was verified in order to determine that the ASIC miners appeared on the test system as four separate entries, each with the name of "CP2102 USB to UART Bridge Controller."

Internet Explorer was used to visit the website <https://bitminter.com>, logging in with the username of "forensicminer@gmail.com" and clicking on the "Start Engine" button that launched the Bitminter control panel and showed the miners actively working.

While the test system was actively mining for Bitcoins and the Bitminter account had accrued enough Bitcoins in order to conduct a transaction, three separate transactions were made from the "forensicminer" Bitminter account to the address of the Multibit wallet as well as the Bitcoin-Qt wallet. The transactions occurred on separate dates and times and the respective wallet application logged each. The below figure depicts the transactions conducted within the Multibit wallet on the dates of January 25, February 8, and February 28, 2014.

Status	Date	Description	Amount (BTC)	Amount (\$)
✓	28 Feb 2014 16:10	Received with 1FdHjMV8s2kzfAdU6TXVS35xkCGcbxAiM6	0.0001	0.01
✓	28 Feb 2014 07:25	Sent to 14igLoRYLjmqc9H5ZSxWqBvdNT3Ro1QeUJ	-0.0002	-0.03
✓	08 Feb 2014 21:31	Received with 1Bt9tP4CU1r9QXtr1X3rZob7SznM8ShE4v	0.0001	0.01
✓	25 Jan 2014 13:57	Received with 1FdHjMV8s2kzfAdU6TXVS35xkCGcbxAiM6	0.0001	0.01

Figure 13: Screenshot depicting the transactions conducted within the Multibit wallet on the test system.

Date	Type	Address	Amount
2/28/2014 16:06	Sent to	Ⓜ Suspect	-0.0002
2/28/2014 11:41	Received with	Ⓜ (14igLoRYLjmqc9H5ZSxWqBvdNT3Ro1QeUJ)	0.0001
2/28/2014 07:25	Received with	Ⓜ (14igLoRYLjmqc9H5ZSxWqBvdNT3Ro1QeUJ)	0.0001

Figure 14: Screenshot depicting the transactions conducted within the Bitcoin-Qt wallet

It is important to note that the address listed as “Suspect” is actually the address of the Multibit wallet, “1FdHjMV8s2kzfAdU6TXVS35xkCGcbxAiM6.”

#### 5.4.2 Collection and analysis of evidence

##### RAM Capture

At the conclusion of the testing period, a 32 GB USB thumb drive was formatted as NTFS and labelled “RAM” for easy identification. The “winen.exe” program was loaded on to the thumb drive and attached to the system that assigned drive letter “F:\.” The program was run by right clicking on the “winen.exe” file and selecting “Run as Administrator.” A series of values in the winen.exe control panel served to tell the program the file path to save the “.E01” image file, the evidence number, examiner name and whether or not to compress the image file.

After approximately 15 minutes, the memory acquisition completed successfully and the contents of the “RAM” thumb drive had created a new file, “Test1.E01.” The thumb drive was properly ejected and stored for later analysis. The test system was shutdown properly and the hard drive was removed.

##### Hard Drive Imaging



The test system hard drive was connected to a Digital Intelligence UltraBay 3D Hardware Write-Blocker and a physical image of the hard drive was conducted.

When the imaging process was completed, the log file generated by Tableau Imager displayed no acquisition errors with an MD5 acquisition hash of B9CCFE1092693E9194AE617262CE3375. From this point forward, all analyses pertaining to the Bitcoin artifacts were done from the digital copy to preserve the original disk's integrity.

### Forensic Analysis

The analysis began with the RAM capture file and progressed through to the system image file. The goal of the analysis was to seek out and recover any evidentiary artifacts pertaining to the Multibit and Bitcoin-Qt wallets. The analysis of the RAM capture was conducted utilizing EnCase 6.19.7 and included keyword searches pertaining to various key Bitcoin terms and artifacts. The analysis of the test image file was completed utilizing EnCase as well as Internet Evidence Finder 6.1 and included searches for various key Bitcoin artifacts, analysis of log files and Internet activity.

### RAM Forensics

The "Test.E01" image file was imported into EnCase, and a new case entitled "Test System Examination" was created. The following search terms were entered in to the Keyword function of EnCase:

- Multibit
- Bitcoin
- 1FdjhMV8s2kzfAdU6TXVS35xkCGcbxAiM6
- 14igLoRYLjmqc9H5ZSxWqBvdNT3Ro1QeUJ
- Bitcoin-Qt
- Bitminter
- forensicminer

Examination of the search results revealed multiple locations in "Program Files," "User files" and the registry where the Multibit and Bitcoin-Qt application files were stored. One of the locations indicated was C:\Users\Suspect\AppData\Roaming."

Further examination of the search results revealed the transactions that had been conducted during the course of the testing phase. There were no specific dates and times associated with either transaction. However, the two addresses associated with the Multibit and Bitcoin-Qt addresses were in clear text.

## Hard Drive Forensics

A new examination case entitled “Bitcoin Test System” was created and the disk “Test System Image.E01” file was loaded as the evidence. The examination strategy consisted of conducting an analysis of the Multibit and Bitcoin-Qt files and the analysis of the Internet activity utilizing Internet Evidence Finder.

**Multibit.** The root directory, “C:\,” contained the bulk majority of the files and folders used by the system and the user. Based on the results obtained from the analysis of the RAM capture, the Multibit application was found to be located in: “C:\Users\Suspect\AppData\Roaming\Multibit.”

Examination of the files in detail resulted in the following information:

- The presence of the Multibit wallet, which had the default name of “multibit.wallet.” This particular file is the main wallet file that contains the user’s private keys and transactions.
- The presence of a rolling backup of the “multibit.wallet” file. This file was located in the subfolder of “rolling-backup” and was entitled “multibit- 20140222190122.wallet.” The series of numbers following the name of the wallet served as the time stamp when the backup was created. In this case, the timestamp of the backup was 02/22/2014 at 19:01:22 (7:01:22 PM). The backup files are created by the respective user and the primary purpose of them is to recover from any sudden loss of power that prevents a clean wallet save.
- Two backups of the wallet file. One of the backups stores the data for encrypted wallets and the other for unencrypted wallets. These files are in the format “YYYYMMDDHHMMSS.wallet” and “YYYYMMDDHHMMSS.info.” The Multibit wallet is backed up to these directories each time that the user opens a wallet, adds or changes the password, adds a receiving address or imports private keys.

There were also two separate entries on two separate dates for unencrypted rolling backups within the Suspect’s Multibit wallet located in the subfolder of “wallet-unencbackup.” The figure below shows the backup files as they appear in EnCase.


	Name	Filter	In Report	File Ext
<input type="checkbox"/> 1	 multibit-20140122211104.wallet			wallet
<input type="checkbox"/> 2	 multibit-20140122211104.info			info
<input type="checkbox"/> 3	 multibit-20140208212323.wallet			wallet
<input type="checkbox"/> 4	 multibit-20140208212323.info			info

Figure 15: Screenshot depicting the two separate dates for the unencrypted rolling backups of the Suspect’s

Each of those entries had timestamps attached to them providing evidence of when Multibit had generated the backups:

- multibit-20140122211104.wallet (01/22/2014 at 21:11:04 hours)
- multibit-20140122211104.info (01/22/2014 at 21:11:04 hours)
- multibit-20140208212323.wallet (02/08/2014 at 21:23:23 hours)
- multibit-20140208212323.info (02/08/2014 at 21:23:23 hours)

Examination of each of the “.wallet” files revealed unreadable data; however, examination of the two “.info” files revealed information pertaining to the wallet version, where the wallet backup was stored, and the specific addresses associated with the wallet file.

Examination of each of the remaining files within the Multibit file folder revealed the following:

- multibit.properties – this file is the MultiBit configuration file that contains the location and name of the wallet, the username and the configurations set forth by the user upon installation.
- multibit.checkpoints – the MultiBit checkpoints file enables the Multibit program from downloading the entire blockchain.
- multibit.info – in addition to the multibit.properties file, this is another location that stores the name of the wallet.

<b>Evidentiary Artifact</b>	<b>Location of Artifact</b>
Multibit program	C:\Users\XXXX\AppData\Roaming\Multibit
Multibit wallet (multibit.wallet)	C:\Users\XXXX\AppData\Roaming\Multibit
Multibit-20140222190122.wallet (Rolling Backup)	C:\Users\XXXX\AppData\Roaming\Multibit
multibit-20140122211104.wallet	C:\Users\XXXX\AppData\Roaming\Multibit\wallet-unenc-backup
multibit-20140122211104.info	C:\Users\XXXX\AppData\Roaming\Multibit\wallet-unenc-backup

Figure 16: Some Multibit artifacts recovered during the hard drive analysis Evidentiary Artifact Location of Artifact

**Bitcoin-Qt.** Examination of the Bitcoin-Qt wallet application began by navigating to the location of the Bitcoin-Qt wallet installation obtained from the analysis of the RAM capture, “C:\Users\Suspect\AppData\Roaming\Bitcoin.” Examination of the contents of the

file folder, revealed the presence of two subfolders “blocks” and “chainstate.” Within the “blocks” subfolder, there was an additional subfolder entitled “index.” The “blocks” and “index” subdirectory contain metadata about all known blocks, and provides the location of them on the user’s disk.

Specifically, the “blocks” subfolder contained 271 individual files. Of those files, there were 240 files with the file extension of “.dat.” Of those 240 files, there were 120 files numbered in sequence starting at “blk00000.dat” and ending with “blk00119.dat.” Those files store actual Bitcoin blocks in network format and are only needed for rescanning missing transactions in a wallet, reorganizing to a different part of the chain, and serving the block data to other nodes that are synchronizing. The remaining 120 files had names that were also numbered in sequence; however, they started at “rev00000.dat” and ended with “rev00119.dat.” The “rev.dat” files contain “undo data.” The user is able to see blocks as patches to the chain state and see the undo data as reverse patches. These files are necessary for rolling back the chainstate, which is necessary in the case of reorganizations when one chain becomes longer than the one currently being worked on.

Examination of the other files located within the Bitcoin-Qt file folder in detail revealed the following information:

- The presence of the “wallet.dat” file which contains the user’s private keys and transactions. Further examination of this file revealed large amounts of unreadable Base64 text.
- The presence of the “peers.dat” file which stores peer information to make a reconnect easier. Further examination of this file revealed large amounts of unreadable Base64 text.
- The presence of the “db.log” file which also stores peer information to make a reconnect easier. Further examination of this file revealed it was an empty file.
- The presence of the “Lock” file which is the Bitcoin database-locking file. Examination of this file revealed it an empty file.
- The presence of the “debug.log” file that is the extensive logging file of Bitcoin-Qt. Further examination of this file revealed a large amount of logging data that included dates and times as well as Bitcoin transaction addresses. The entire log file was exported to the desktop of the forensic workstation and given the file name of “Test System Bitcoin Log.” Initial examination of the log revealed a standard log file in readable format containing dates, times, blocks and IP addresses. A search of the log file utilizing the test system’s IP address of 108.XXX.XX.XX resulted in several hits within the log file.

Further examination of each log file entry revealed that each was from the blockchain and contained a date and time stamp, message version, the specific blocks within the block chain, as well as the IP address of the test system and the peer network. The following illustrates the breakdown of one of the log entries from the blockchain:

- Date/Time: 03/02/2014 at 18:23:39 hours (6:23:39 PM)
- Message Version: Satoshi: 0.8.6 version 7001 (Version of Bitcoin-Qt installed on test system)
- Blocks: 257627
- US: 108.XXX.XX.XX (IP address of the test system)
- Them= 131.XXX.XX.XX (IP address of the connected peer) A query of the peer IP address through [www.iplocation.com](http://www.iplocation.com), revealed it to be located in Ontario, Canada.

The above log file entry is a result of a series of messages transmitted and received by the peers of the Bitcoin network. When connecting to the Bitcoin network, everyone broadcasts an “addr” message containing his or her own IP address every 24 hours. Nodes relay these messages to the peers and they are stored if the address is new. Through this system, everyone has a reasonably clear picture of which IPs are connected to the network at that particular moment. The peers will request the full transaction with a “getdata” message that is a request for a single block or transaction. If the peers consider the transaction valid after receiving it, they will in turn broadcast the transaction to all of their peers with an “inv” message.

#### Internet Evidence Finder Forensics

The “Test System.E01” image file was loaded into Internet Evidence Finder 6.2.3 and “Internet Explorer” and “Bitcoin” were selected as the evidentiary artifacts that the program would seek out. At the conclusion of the processing, a section under the “IEF Refined Results” labeled “Peer to Peer” was populated with two entries for “Bitcoin Addresses.” Further examination of those results revealed two addresses, “1FdhjMV8s2kzfAdU6TXVS35xkCGcbxAiM6” and “14igLoRYLjmqc9H5ZSxWqBvdNT3Ro1QeUJ.” In viewing the source of the evidence, it was determined that the above listed addresses originated from the “wallet.dat” file, located at “C:\Users\Suspect\AppData\Roaming\Bitcoin.”

After right clicking on each of the addresses within Internet Evidence Finder and selecting "Query Bitcoin Block Chain," an Internet Explorer window opened and information pertaining to each of the addresses such as the Public Key and the Public Key hash, as well as the sent and received transactions with each of the addresses was visible on the blockexplorer website [45]. The following query of the Bitcoin address "1FdhjMV8s2kzfAdU6TXVS35xkCGcbxAiM6" revealed the following:

- First seen: Block 282447, 01/25/2014 at 20:10:00 hrs. (This is the first block that the address was used in)

- Received transactions: 2
- Received BTC: 0.0002
- Sent transactions: 1
- Sent BTC: 0.0001
- Hash160 (This hash value is the hash of the public key):

"a082bc485913a5d5fffa79e824daa02bebac36a1"

- Public key:  
"02738b96756e7c101f44098665d64dd41e3a6f9b08b7130db71161be77bf978451"

The following query of the Bitcoin address "14igLoRYLjmqc9H5ZSxWqBvdNT3Ro1QeUJ" revealed the following:

- First seen: Block 288294, 02/28/2014 at 13:39:12 hrs. (This is the first block that that the address was used in)

- Received transactions: 2
- Received BTC: 0.0002
- Sent transactions: 2
- Sent BTC: 0.0002
- Hash160: 28ca45b6c41a17c31c551632c6f9412d705c46df
- Public key:

03d2e19dcabe7e5557e204ba6865355f82062f67794ccb1f450778f05954a215a0

Further examination of the findings from Internet Evidence Finder revealed no evidentiary artifacts from the Multibit wallet or Bitminter mining applications.

## 5.5 Conclusion

Bitcoin cryptocurrency is a relatively new technology and very little research has been dedicated to what specific forensic artifacts are left on a user's system as a result of Bitcoin, what those artifacts mean and how to recover them in order to build a successful

case involving Bitcoin. This research sought to provide a history of Bitcoin cryptocurrency and through the use of a test environment, ascertain what specific Bitcoin artifacts are recoverable from a user's system with Bitcoin wallet and mining applications installed and actively used. The examination of the data collected after the testing phase provided evidence validating the installation of the Multibit and Bitcoin-Qt wallet applications on the test machine, as well as confirms the creation of Bitcoin transactions generated by the wallet applications. In addition, the analysis provided evidentiary artifacts relating to the Bitminter mining software and the interaction of each Bitcoin application with the operating system, registry, and RAM. The analysis of the RAM was a success in that it returned a multitude of results that matched the Bitcoin wallet addresses, transactions and Bitcoin applications on the test system.

## 6. Memory Investigation of Bitcoin Clients

Evidence relating to the use of Bitcoins can potentially be located in different locations, such as the Blockchain, the client software and the network protocol. However, existing Bitcoin forensics appear to focus only on the Blockchain so far [60]. The below described research was focused on digital evidence present in the memory.

Memory-resident data of an application can be analyzed in a structured and an unstructured manner. The structured approach focuses on the OS perspective of the memory (processes, files, tables, etc.), whereas the unstructured approach regards the memory dump as a collection of values that can be crawled using tools such as strings and grep. Bitcoin applications function as a storage for Bitcoin keys (Bitcoin wallet) and can contain data of forensic interest, such as public and private keys, addresses, user labels and transaction details.

### 6.1 Bitcoin Core and Electrum

The specific research was focused on two major Bitcoin clients, Bitcoin Core [61] and Electrum [62] which support application-specific functions and could be of great significance to a forensic investigator.

A Bitcoin client generally has, amongst others, three major functions:

- 1) to store keys and user data securely,

2) to initiate Bitcoin transaction from the wallet and

3) to request for Bitcoins.

An example of one such function supported by both Bitcoin Core and Electrum is the option to sign and verify human-readable messages.

In Bitcoin Core, keys and user data are stored in a Berkeley database [63]. This database is written in C and stores the key material as binary data. By default, this database is not encrypted, but the user can choose to do so via the menu option “Settings >Encrypt Wallet”. Apart from the Bitcoin private and public keys in the wallet, the database stores various user data, such as contacts - basically an association between a label and a public address - and the transaction history of the wallet. Bitcoin Core also enables the user to make a backup of a wallet file to a location on the disk.

The user can send Bitcoins from the wallet via the “Send” tab. To do so, the user enters the Bitcoin address, label and amount for the transaction in the screen and presses “Send”. Optionally, the user can add additional addresses to the same transaction or override the default settings for determining the transaction fee. Note that the label is not part of the public Bitcoin ledger, but is stored in the wallet for bookkeeping purposes.

To request a payment to the wallet, the user has to enter the label, the amount and message in the “Receive” tab in Bitcoin Core and press “Request Payment”. In turn, the application generates the request for a Bitcoin address available in the wallet. This address corresponds to a public and private key pair in the wallet. The request is represented as a URI string and as a QR code, which then can be shared with other Bitcoin users for them to initiate the payment requested.



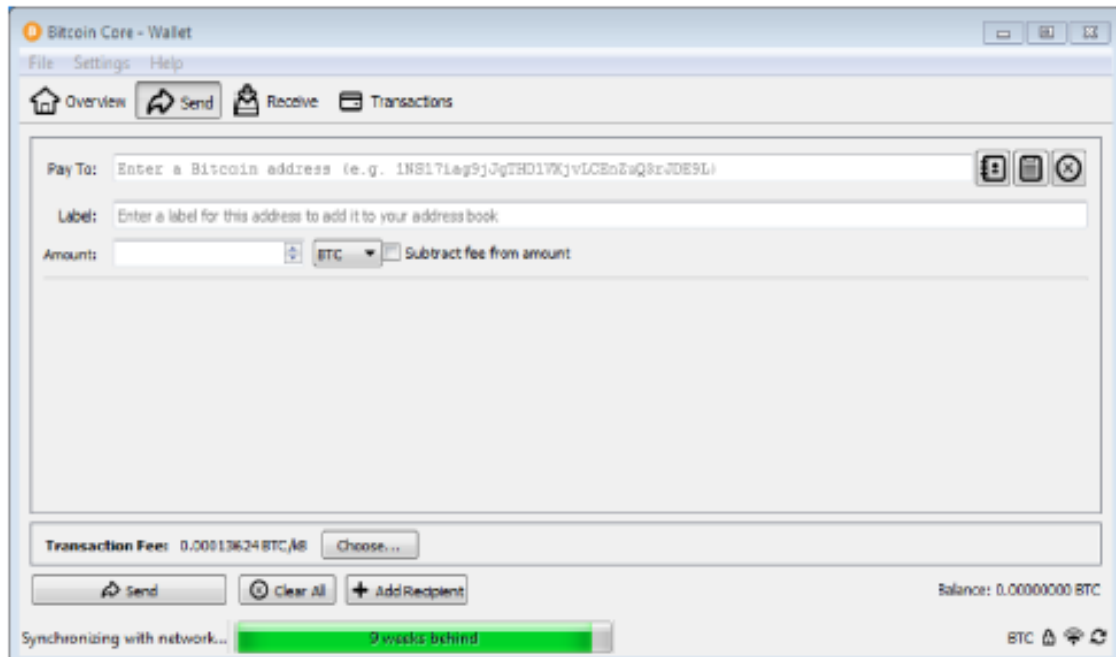


Figure 17: Bitcoin Core Wallet

The below figure shows all the Application Data that are present in the process memory of Bitcoin Core and Electrum. However, even though all of them are of forensic relevance, not all can be easily traced back in the process memory.

Application	Function	Data
Bitcoin Core	Secure Storage	Public and Private keys, transaction data (address, label, transaction-id, amount, fee and timestamps), contacts, passphrase, backup location (s)
	Send Bitcoins	Address, label, transaction-id, amount, fee, timestamp
	Receive Bitcoins	Public and private keys, address, label, message
Electrum	Secure Storage	Master public key, master private key, derived public and private keys, transaction data (address, label, transaction-id, amount, fee and timestamp), contacts, passphrase, seed, backup location(s)
	Send Bitcoins	Address, label, transaction-id, amount, fee, timestamp
	Receive Bitcoins	Public and private key, label, amount, expiry date

Figure 18: Application data in Bitcoin Core and Electrum.

## 6.2 Memory Images

Memory images are acquired from a virtual machine running in various states. The virtual machine runs the Microsoft Windows 7 Enterprise SP1 (64-bit) operating system in a licensed VMWare Fusion Professional Edition v6.0.65 environment. The VM is configured with 1 GB of RAM. The operating system was fully patched in all machines. Additionally, all virtual machines had VMware Tools installed. In the VM, Electrum v2.6.2 was installed and brought to a specific state. Bitcoin Core v0.11.1 based on QT v5.5.0 was used in the Bitcoin Core scenarios. Memory snapshots were acquired by suspending the virtual machine and copying the file with the .vmem extension from the virtual machine's folder.

Bitcoin Core has been investigated both in an unencrypted and in an encrypted state. Since Electrum v2.6.2 only supports a wallet with encrypted private data, this application cannot be analyzed in an unencrypted state. During the creation of the memory images, all user data (e.g. labels, passphrases) and application data (e.g. addresses, public and private key values) were documented in detail.

No	Application	State	Memory Image
1	Bitcoin Core	Unused Client (benchmark)	Unused client with initialized, unencrypted wallet. The wallet is fully synchronized with blockchain.
2	Bitcoin Core	Used Client (unencrypted)	Based on scenario 1, after usage. The wallet is not encrypted. Entries of sending and receiving addresses (contacts) are created with user-specified labels. The wallet has several incoming transactions. An on-disk backup of the wallet is made.
3	Bitcoin Core	Used Client (encrypted)	Based on scenario 2, but the wallet is now encrypted.
4	Bitcoin Core	Used Client after reboot	Based on scenario 3. The virtual machine is rebooted and an outgoing transaction is initiated.
5	Electrum	Unused Client (benchmark)	Unused client with initialized, encrypted wallet. A seed has been generated for the wallet and a passphrase is set. The wallet is fully synchronized with the Electrum server.
6	Electrum	Used Client (encrypted)	Based on scenario 5, after usage. Entries of sending and receiving addresses (contacts) are created with user-specified labels. The wallet has several incoming transactions. An on-disk backup of the wallet is made.
7	Electrum	Used Client after reboot	Based on scenario 6. The virtual machine is rebooted and an outgoing transaction is initiated.

Figure 19: Memory Images per Application

During memory analysis, these values were traced back in the memory images to determine their format, location and context. Incoming transactions to the wallets under investigation come from an external wallet, whereas all outgoing transaction return to this

same external wallet. Furthermore, all labels defined in the scenarios follow a predefined format, namely: MAGIC\_CATEGORY\_RANDOM\_MAGIC

In this format, MAGIC is a hex value set to f0r3ns1c to facilitate tracing in memory. CATEGORY indicates for which purpose the label was created (e.g. the value “req” for a payment request). In turn, RANDOM consists of a variable-length alphanumerical random string which ensures the uniqueness of the label.

In the next stage, the memory images are analyzed using Volatility v2.5 [64] and standard Linux command-line tools in a virtual machine running Kali GNU/Linux v2.0 (64-bit) operating system. Memory snapshots were stored on the host system and available to the analysis machine via an HGFS mount point. The memory images were analyzed in two different ways.

Most importantly, all application data known to be processed by the application as specified in the previous step were traced back in memory. This analysis is similar to the unstructured approach but Volatility makes it possible to attribute values found in memory to a particular process. For this analysis, the yarascan plugin in Volatility was used. Then the modified yarascan plugin, which includes detailed metadata of the VAD area in which a particular value was found in its output. For completeness, the full memory image was searched, rather than only the (private) process memory of the Bitcoin application. Binary values, such as public and private keys, were searched both as string and as binary value.

All known user strings found in memory were analyzed for their format, their location including their VAD properties and their immediate context. Based on these results, it is possible to determine which application data is likely to be memory-resident and how it might be retrieved from memory when their values are not previously known (as is the case in a regular forensic investigation).

Apart from the unstructured analysis described above, a more structured analysis was performed on the Bitcoin application process in each memory image. In particular, the memory-mapped files, registry keys and connections of the application were examined using standard Volatility plugins. This is to ensure that no important forensic clue is missed. Also, if it is possible to extract Bitcoin data files (i.e. wallet files and log files) from memory, then these files would be processed as standard on-disk artefact. Thus, further detailed memory analysis becomes unnecessary.

### 6.3 Findings for Bitcoin Core

### Private Keys

All known private keys were located in binary format in the process memory when the wallet was unencrypted. When the wallet was encrypted, no private key could be traced back in memory, not even directly after a transaction was initiated. Furthermore, no occurrence of private keys in Wallet Import Format (WIF) was found. This observation is not surprising, as private keys are stored in binary format in the wallet database and WIF-formatted keys are only created when exporting keys.

### Public keys

All known public keys in binary format were located in the process memory in all images. Addresses were only calculated from the binary public key in the wallet when a label is associated with them. Not all public keys and addresses occurred equally often. However, no correlation was found between particular categories (send address, receiving address or payment request) or the usage in transactions on the one hand and the number of occurrences of the associated public keys and Addresses.

### Labels

Known labels have been found in all memory images. All different labels were found, four labels for each category (i.e. message, receiving address, payment request and send address). In any case, all labels appeared more than once in process memory. However, because labels are not known upfront and do not follow a fixed pattern, they can be hard to locate for an investigator. This could only be done based on the context of the user labels.

### Transaction IDs

Both search hits corresponded to the Transaction ID of the last transaction initiated from the client were found.

### Passphrase

The passphrase used for wallet encryption, was not encountered in process memory.

### File locations

The full path and location of a backup file was present in memory. When scanning for all file paths in the process memory, many other file paths were present as well. Hence, an investigator should scan through these manually to determine whether they are linked to a backup file (e.g. based on file name or file location, e.g. a thumb drive or user folder).

#### *Format*

The public and private keys in binary format were also present as binary data in memory. All other values, namely addresses, transaction-IDs, labels and file locations, appeared as string values in memory.

#### *Location*

All values were found in private (nonshared) read/write regions in process memory. No correlation was found between the memory location and VAD on the one hand and the type of data on the other hands. Therefore, no conclusions could be drawn on the co-existence of particular (types of) values in process memory.

#### *Context*

##### Private keys

Private keys can be easily retrieved from memory by carving sequences of 32 bytes directly following this fingerprint. Doing so revealed the existence of multiple unknown private keys. This is expected, as not all private keys in the wallet are involved in the user actions during image creation and included in the search item list.

##### Public keys and Addresses

Analysis of the fingerprints showed that these are in fact Berkeley database tags for public keys in the wallet.dat file. Comparison with the data in the respective wallet files confirmed that the memory regions in which the values were uncovered were in fact memory-resident parts of the wallet file. Hence, it is very likely that the same information can also be extracted from the wallet file. Also, public keys in binary format have a xed length6 and public key addresses have a predictable length and format. All such values following the fingerprints mentioned above will yield the correct results.

##### Labels

No fingerprint could be identified to systematically extract labels from process memory with one exception. For a given public address having the name" tag, its corresponding label consisted of the first human-readable string preceding this address.

#### Transaction IDs

Only two transaction IDs were traced back in memory. Information could be better retrieved by analysis of the memory-resident debug.log file.

#### File Locations

No conclusions could be drawn from the context of the memory locations of the hits on file location data (paths and file names).

#### Files

The application files, wallet file (wallet.dat) and the application log file (debug.log), were specifically important.

The wallet file is the key store containing the bitcoin keys and all user data (as described earlier). In an encrypted wallet, the private keys are only readable when the correct passphrase is known. The debug.log file, on the other hand, can be interesting due to two reasons: it shows the transactions initiated by the wallet with their date and time and, possibly more important, it shows the full file location of any backup created by the user. Hence, it is possible to identify wallet backups on user locations and attribute these to the use of the bitcoin client.

#### Registry Keys

Registry keys used by the Bitcoin Core client were identified using the handles plugin. The presence of these keys serves as an indicator for the presence of an active instance of the Bitcoin Core client.

#### Connections

IP addresses of neighboring bitcoin peers were visible. These connections can be identified based on the TCP port number 8333. Such connections can serve as an indicator for the presence of an active bitcoin client, but further forensic usefulness is yet unclear.

## 6.4 Findings for Electrum

### Private keys

No private key or checksum of private keys was located in process memory.

### Public keys and Addresses

Distinct known private keys were found back in binary format with their corresponding public key address. However, no correlation could be detected between the usage of a particular key (e.g. in a payment request) and the number of occurrences.

### Labels

All known labels were present in the process memory.

### Transaction IDs

All known transaction IDs were found at various times in process memory.

### Passphrase

The passphrase for the wallet was not encountered in process memory.

### *Format*

Almost all data appeared as string values in memory. However, public keys associated with key pairs involved in the wallet's transactions were an exception. More specifically, the public key of the input transaction(s) for outgoing transactions initiated just before the image was made occurred in the binary format.

### *Location*

All values were found in private (non-shared) read/write regions in process memory. There was one exception: one occurrence of the file name of the wallet backup file (not the full path) was found in a memory-mapped DLL file (shlwapi.dll). So, no conclusion could be drawn on the co-existence of particular (types of) values in process memory.



### *Context*

Analysis of the direct context in which artefacts appeared showed that most were present as part of JSON-formatted data. A closer analysis revealed that those occurrences were part of memory-resident data from the Electrum wallet file.

### *Files*

While examining the handle tables from the Electrum processes, no reference to wallet files or other relevant Electrum files was found. Hence, extracting these files using Volatility's dump files plugin was not possible. However, analysis of the memory-resident MFT table revealed the entries for all these files. In addition to the metadata available in the MF table, file content of the config, contacts and recent\_servers files were found in the \$DATA section of MFT records due to their small file size. No reference to wallet backup files was found in the handle tables in any of the images.

### *Registry Keys*

No application-specific registry key was in use by the Electrum application.

### *Connections*

The Electrum application opens multiple connections TCP port 50002 and one connection on TCP port 443, to the seed server. These connections are visible in the memory image with the netscan plugin and can serve as an indicator for the presence of the Electrum application.

## 6.5 Conclusion

All in all, data of forensic interest can be extracted from memory by scanning the process memory for fingerprints identified in this research or by searching fixed patterns with regular expressions (e.g. Bitcoin addresses or file paths). Despite the potential to use the located evidence to attribute Bitcoin transactions to the user of the computer, it is unlikely the evidence located will directly result in the seizure of the assets stored in the wallet. It must be noted, however, that most data found in memory are also available in application and wallet files on disk, with a few exceptions. As such, process memory analysis is potentially beneficial to a forensic investigation, particularly when application and wallet files are not available.

## 7. Ethereum and Smart Contracts

Ethereum is an open source, public, blockchain-based distributed computing platform and operating system featuring a smart contract functionality. It supports a modified version of Nakamoto's (Bitcoin's) consensus via transaction-based state transitions. The generated cryptocurrency by the Ethereum platform is Ether (ETH) and is used to compensate mining nodes for computations performed [65]. Each Ethereum account has an Ether balance and Ether may be transferred from one account to another.

The notion of smart contracts has been introduced by Nick Szabo in 1997 [66]. He described the concept of a trustless system consisting of self-executing computer programs that would facilitate the digital verification and enforcement of contract clauses contained in legal contracts. However, this concept only became a reality with the release of Ethereum in 2015. Ethereum smart contracts are different from traditional programs in several aspects. For example, as the code is stored on the blockchain, it becomes immutable and its execution is guaranteed by the blockchain. Nevertheless, smart contracts may be destroyed, if they contain the necessary code to handle their destruction. Once destroyed, a contract can no longer be invoked and its funds are transferred to another address. Smart contracts are usually developed using a dedicated high-level programming language that compiles into low-level bytecode. The bytecode of a smart contract is then deployed to the blockchain through a transaction. Once successfully deployed, a smart contract is identified by a 160-bit address. Despite a large variety of programming languages such as Vyper [67], LLL [68] and Bamboo [69]) Solidity [70] remains the most prominent programming language for developing smart contracts in Ethereum. Solidity's syntax resembles a mixture of C and JavaScript. It comes with a multitude of unique concepts that are specific to smart contracts, such as the transfer of funds or the capability to call other contracts.

### 7.1 Ethereum Virtual Machine

The Ethereum blockchain consists of a network of mutually distrusting nodes that together form a decentralized public ledger. This ledger allows users to create and invoke smart contracts by submitting transactions to the network. These transactions are processed by miners. Miners execute smart contracts during the verification of blocks, using a

dedicated virtual machine denoted as the Ethereum Virtual Machine [71]. The EVM is a stack-based, register-less virtual machine, running low-level bytecode, that is represented by an instruction set of opcodes. To guarantee termination of a contract and thus prevent miners to be stuck in endless loops of execution, the concept of gas has been introduced. It associates costs to the execution of every single instruction. When issuing a transaction, the sender has to specify the amount of gas that he or she is willing to pay to the miner for the execution of the smart contract. The execution of a smart contract results in a modification of the world states, a data structure stored on the blockchain mapping an address  $a$  to an account state  $\sigma[a]$ . The account state of a smart contract consists of two main parts: a balance  $\sigma[a]_b$ , that holds the amount of ether owned by the contract, and storage  $\sigma[a]_s$ , which holds the persistent data of the contract. Storage is organized as a key-value store and is the only way for a smart contract to retain state across executions. Besides the world state  $\sigma$ , the EVM also holds a transaction execution environment  $I$ , which contains the address of the smart contract that is being executed  $I_a$ , the transaction input data  $I_d$ , the transaction sender  $I_s$  and the transaction value  $I_v$ . The EVM can essentially be seen as a transaction-based state machine, that takes as input  $\sigma$  and  $I$ , and outputs a modified world state  $\sigma'$ .

## 7.2 Ethereum Honeypots

A honeypot is a smart contract that pretends to leak its funds to an arbitrary user (victim), provided that the user sends additional funds to it. However, the funds provided by the user will be trapped and at most the honeypot creator (attacker) will be able to retrieve them. The below Figure depicts the different actors and phases of a honeypot.

A honeypot generally operates in three phases:

1. The attacker deploys a seemingly vulnerable contract and places a bait in the form of funds;
2. The victim attempts to exploit the contract by transferring at least the required amount of funds and fails;
3. The attacker withdraws the bait together with the funds that the victim lost in the attempt of exploitation.

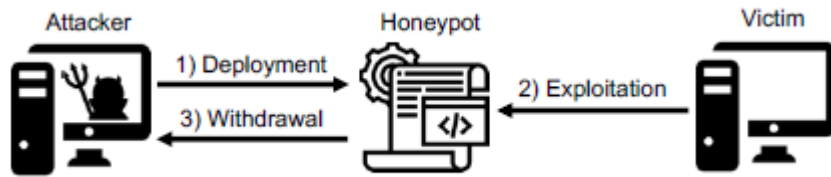


Figure 20: Honeypot Phases

An attacker does not require special capabilities to set up a honeypot. In fact, an attacker has the same capabilities as a regular Ethereum user. He solely requires the necessary funds to deploy the smart contract and place a bait.

## 7.3 Manticore

Honeypots have two participants, the creator of the honeypot and the user whose funds are trapped by the honeypot. In general, they are a type of fraud that combines security issues with scams. They rely on the blockchain itself and can be detected by forensics tools which aim to analyze abnormalities in their overall function. Such a tool is Manticore, an open-source dynamic symbolic execution framework which analyzes binaries and Ethereum smart contracts.

### 7.3.1 What is Manticore?

Dynamic symbolic execution is a program analysis technique that explores a state space with a high degree of semantic awareness [72]. For paths that are explored by the analysis, dynamic symbolic execution identifies a set of path predicates: constraints on the program's input. These are used to generate program inputs that will cause the associated paths to execute. This approach produces no false positives in the sense that all identified program states can be triggered during concrete execution. For example, if the analysis finds a memory safety violation, it is guaranteed to be reproducible.

Symbolic execution has been extensively researched in a security context [73], but industry has been slow to adopt the technique because of the limited availability of flexible, user-friendly, tools. Furthermore, existing frameworks are tightly coupled to traditional execution models, which makes symbolic execution research challenging for alternative execution environments, such as the Ethereum platform.

Manticore is a symbolic execution framework for analyzing binaries and smart contracts. Trail of Bits has used this tool internally in numerous code assessments [74] [79], and in program analysis research, including the DARPA Cyber Grand Challenge (CGC) [75].



*Figure 21: Manticore Logo*

### 7.3.2 Architecture

Manticore's design is highly flexible and supports both traditional computing environments (x86/64, ARM) and exotic ones, such as the Ethereum platform. To our knowledge, it is the only symbolic execution framework that caters to such different environments. It is also simple, extensible and as self-contained as possible, avoiding unwarranted external dependencies [76]. The primary components are the Core Engine and Native and Ethereum Execution Modules. Secondary components include the Satisfiability Modulo Theories (SMT-LIB) module, Event System, and API.

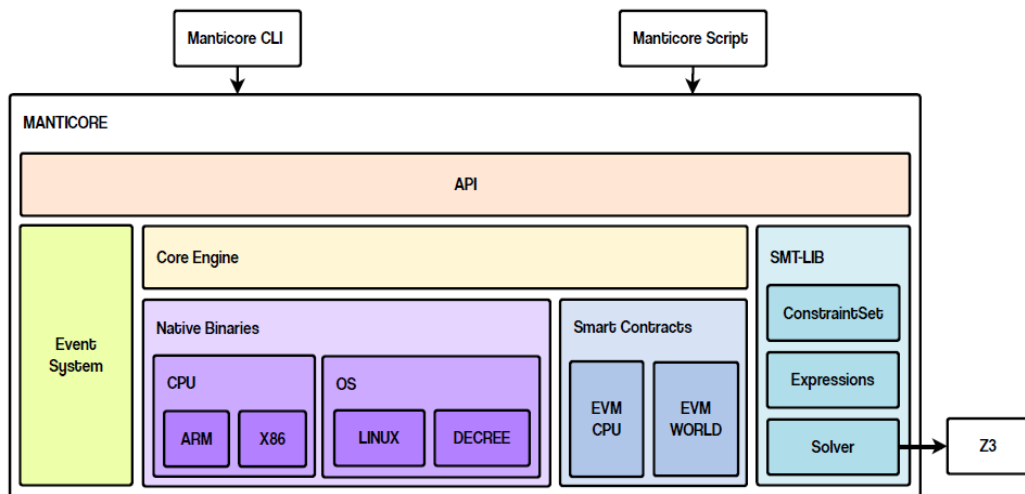


Figure 22: Manticore's Architecture

### 7.3.3 Core Engine

The Core Engine is the source of Manticore's flexibility. It implements a generic platform-agnostic symbolic execution engine that makes few assumptions about the underlying execution model.

This Core Engine operates and manages program states according to the State Life Cycle shown in the above Figure. Program states are abstract objects that represent the state of a program at a point in execution. These objects expose an execution interface that the Core Engine invokes to trigger one atomic unit of program execution. For native binaries and Ethereum, this is one instruction. During execution, states can interrupt back to the Core Engine to signal that a life cycle event needs to be handled.

The State Life Cycle, shown in the below Figure, defines three states:

- 1)Ready,
- 2)Busy and
- 3)Terminated,

as well as two events:

- 1)Termination and
- 2)Concretization.

The Core Engine repeatedly selects a Ready state and executes it (transitioning it to Busy). An executing Busy state can either transition back to Ready or signal a Life Cycle event for the Core to handle.

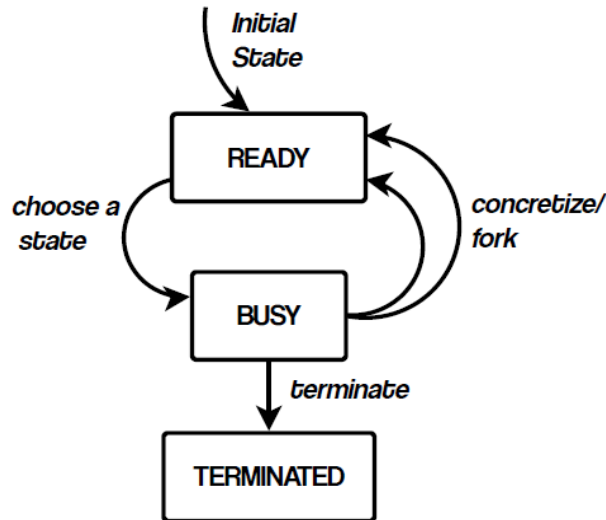


Figure 23: The State Life Cycle

The Termination event occurs when a state reaches an end, typically on program exit or a memory access violation, which transitions the state to Terminated. Concretization happens when a state signals that a symbolic object should be converted into one or more concrete values, subject to the current constraints on the State. For each concrete value, one new "child" State is created and marked Ready. The most common case of Concretization, called "forking", occurs when a program counter register becomes symbolic and is concretized to possible concrete values. This causes new states to be generated for each new program path.

State exploration can be customized using various policies, which implement a variety of heuristics for Ready state selection and Concretization. The Core Engine was designed for parallelism and supports multiple processes for state queue processing [77].

#### 7.3.4 Native Execution Module

The native binary symbolic execution module abstracts hardware execution to implement the high-level execution interface that the Core Engine expects, via symbolic emulation of the CPU, memory and operating system interfaces. Currently, the native execution module emulates Linux on x86, x86 64, ARMv7, and AArch64 as well as DECREE on x86.

1) CPU Emulation: The symbolic CPU emulation is straightforward and follows the ISA specification, with the caveat that emulated registers and instructions must support both concrete and symbolic values. Implementing symbolic support for an instruction typically involves building symbolic expression trees, as opposed to performing computation directly.

2) Memory Emulation: Manticore has a simple virtual address space emulation with interfaces for reading, writing and managing memory mappings. Different policies for handling symbolic memory accesses are implemented. These include fully symbolic and concretized memory models.

3) Operating System Emulation: Manticore includes OS support for the Linux and DECREE operating systems, emulating the system call (syscall) interface, interfaces related to a process address space (e.g. auxiliary vectors, thread local storage) and miscellaneous state setup (e.g. binary loading). Syscalls must handle symbolic inputs, yet few can be reasonably modeled symbolically. Manticore therefore concretizes system call arguments and forwards such calls to the real OS.

### 7.3.5 Ethereum Execution Module

Manticore supports Ethereum smart contracts, which are applications compiled according to the Ethereum Virtual Machine (EVM) specification that run on the Ethereum blockchain. There are many differences between EVM and traditional execution. A few examples include a "gas" cost for executing instructions, radically different memory and persistent storage models, and execution state rollbacks. Despite these differences, adding Ethereum support did not require substantial architectural changes to Manticore, since the Core Engine is completely decoupled from all execution platform details.

1) Ethereum Symbolic Execution: Smart contracts receive input as network transactions consisting of a value and a data buffer. The transaction data buffer



contains information about which function should be executed in a contract, and its arguments.

Symbolic execution of smart contracts involves symbolic transactions, where both value and data are symbolic. Symbolic transactions are applied to all Ready states, which cause the symbolic execution of one transaction. Symbolic transactions can be repeatedly executed to generically explore the state space of a contract.

Manticore's emulated environment for smart contract execution supports an arbitrary number of interacting contracts. It is capable of tracking not only a single contract's state, but a full Ethereum "world", with multiple interacting contracts.

### 7.3.6 Auxiliary Modules

Manticore also has auxiliary modules like the SMT-LIB module that supplies a custom symbolic expression object model and an SMT solver interface. Different solvers can be used seamlessly, since Manticore interacts with solvers via the SMT-LIB language.

The Event System decouples Manticore as a whole from external instrumentation-based analyses. Arbitrary points within Manticore can broadcast various symbolic execution events (e.g. memory reads/writes, state forking, concretization) that can be handled by an event subscriber outside of Manticore, such as an API client. This provides the foundation for Manticore's plugin system allowing users to create modular, event-based analyses.

The API module of Manticore interacts with the Core Engine, SMT-LIB module, and Event System and implements the various external programming interfaces for Manticore.

## 7.4 Usage

Manticore has a command-line interface and an API that works for both binaries and smart contracts. An example command is as follows:

```
$ manticore target_binary ++ +++.txt --data AA --procs 10
```

The arguments “target ++ +++.txt” instruct Manticore to execute and analyze the target\_binary with two arguments. The first is a 2-byte string of symbolic data. The second is a mixed symbolic/concrete string with five bytes of symbolic data followed by the concrete bytes “.txt. –data” specifies concrete bytes to prefix the stdin input stream, which by default contain 256 symbolic bytes. “–procs” allocates 10 cores to the analysis. Manticore’s output is a directory containing generated inputs and information about each discovered state, as shown below:

```
$ ls mcore_x2gncpcq/

test_00000000.argv      test_00000000.input

test_00000000.messages  test_00000000.smt

test_00000000.stdin    test_00000000.trace

...
```

For example, “test\_00000000.stdin” can be piped directly to the stdin of the program during concrete execution to trigger the execution state corresponding to “test\_00000000”.

Manticore also has a Python API for advanced users to customize their analysis using various forms of instrumentation. The API allows users to execute callbacks when a certain state is reached. The callback can access the corresponding State object, which allows complete control over the emulated state [80]. CPU registers, memory and operating system state can be read, written, filled with symbolic bytes or concretized. Moreover, states can be pruned, custom constraints can be applied, and satisfiability queries can be sent to the solver. Writing code using the hook API is relatively straightforward, e.g.:

```
from manticore.native import Manticore

m = Manticore.linux('./target')

@m.hook(0x400ca0) def hook(state):

    # Disregard state if RDX can be equal 0x44

    # (RDX could be symbolic)

    if state.can_be_true(state.cpu.RDX == 0x44)

    state.abandon()
```

```

input_buf = state.new_symbolic_buffer(32)

# Apply arbitrary precondition on input

buffer

state.constrain(input_buf[0] != ord('A'))

# Write symbolic buffer at address RBX

state.cpu.write_bytes(state.cpu.RBX,

input_buf)

```

## 7.5 Ethereum Smart Contract Analysis Evaluation

Manticore was evaluated based on a corpus of 100 Ethereum smart contracts taken directly from the Ethereum blockchain. An analysis was made that repeatedly executed symbolic transactions against a contract and tracked the number of states discovered and coverage of the contract code.

Here are the results of running this analysis, with a timeout of 90 minutes per contract. Manticore produced an average coverage of 65.64%, with an approximately equal median [74]. The mean total number of (symbolic) states reached was 207.71, with median 52, showing that there were a number of outliers where more states were discovered. Coverage ranged from 0% to 100% showing that there were indeed some contracts that caused Manticore to entirely fail, while there were many others that Manticore was able to completely explore.

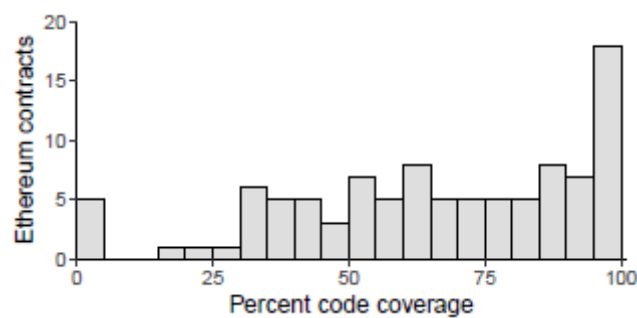


Figure 24: Ethereum Contract Code Coverage

## 7.6 How Manticore can be used for smart contract investigation

An effective way to maximize code coverage in software tests is through dynamic symbolic execution. As described in the previous chapter, Manticore is a symbolic execution tool for analysis of smart contracts and binaries. It can:

- detect potential overflow and underflow conditions on “ADD”, “MUL” and “SUB” instructions
- detect potential uses of uninitialised memory or storage
- calculate code coverage
- generate inputs which could trigger unique code paths (Solidity source code needed)
- offer a Python API for analysis of EVM bytecodes
- Auto-generate inputs for triggering different unique code paths
- Trace inputs that crashed the program
- Record instruction-level execution trace
- Expose its analysis engine via Python API
- Automatically generates inputs that trigger unique code paths
- Discovers inputs that crash programs via memory safety violations
- Records an instruction-level trace of execution for each generated input
- Exposes programmatic access to its analysis engine via a Python API

Manticore's flexible architecture allows it to support both traditional and exotic execution environments and its API allows users to customize their analysis [75]. Here, we discuss Manticore's architecture and demonstrate the capabilities we have used to find bugs and verify the correctness of code for our commercial clients.

Manticore includes built-in "detectors" for certain properties of Ethereum smart contracts. Used in this way, Manticore acts like a linter that reports on these conditions as they are observed while exploring the state space of a smart contract. These detectors may or may not apply to the contract being explored, may falsely detect issues, or may fail to report a true issue. These detectors are a default set of properties that we expect *most* contracts will share. It is always best to reason about the application-specific properties of your contract, and then build analyses to verify them.

Manticore uses symbolic execution to simulate complex multi-contract and multi-transaction attacks against EVM bytecode. Once your app is functional, write Manticore

tests to discover hidden, unexpected, or dangerous states that it can enter. Manticore enumerates the execution states of your contract and verifies critical functionality [76].

If your contract doesn't require initialization parameters, then you can use the command line to easily explore all the possible executions of your smart contract as an attacker or the contract owner:

```
manticore contract.sol --contract ContractName --txaccount [attacker|owner]
```

Manticore will generate a list of all the reachable states (including assertion failures and reverts) and the inputs that cause them. It will also automatically flag certain types of issues, like integer overflows and use of uninitialized memory.

Using the Manticore API to review more advanced contracts is simple [78]:

1. Initialize your contract with the proper values.
2. Define symbolic transactions to explore potential states.
3. Review the list of resulting transactions for undesirable states.

All in all, what should be noted is that through Manticore's use, potential unwanted code errors can be checked and avoided [81]. In case there is a suspicion of planned malicious activities concerning the smart contract world and in particular the Ethereum Virtual Machine, this tool can serve as a medium for thorough code analysis and adverse outcome prevention.

## 8 Conclusion

The rise of cryptocurrencies' value on the market and the growing popularity around the world open a number of challenges and concerns for business and industrial economics. Cryptocurrencies have largely changed the way we perceive the global economy, the value of money, anonymity, online privacy and online services altogether. While their uses can be wide and blockchain can also be used in a number of sectors, cryptocurrencies can also be misused. They are an infant notion that can be easily manipulated while its increased privacy-by-design can backfire and feed in a number of illicit activities ranging from market manipulation to drug trafficking and terrorism financing.

Investigating Cryptocurrencies provides cyber and financial investigators with the necessary background, techniques and methodologies to break through the blockchain "lockdown" and investigate crimes involving cryptocurrency transactions. The specific thesis described, based on lots of already-proven material, that anonymity in the blockchain network is not always granted. Based on evidence and through forensics investigation, artifacts can provide information which under the right perspective, can be quite useful. Either for criminal or simple use, cryptocurrencies can leave their track on a variety of records. The only concern is which people will be able to take advantage of this and whether they will use it for the greater good (e.g. a cyber crime investigation) or for personal ambiguous motives.

It was also stated that even the technical members of the Bitcoin system do not always consider anonymity as a key priority to their overall prominent design and do not seem quite concerned in order to preserve it.

As far as the smart contract world and platforms as Ethereum are concerned, symbolic execution tools can be used in order to prevent potential malicious actions that intend to harm transactions and temper their execution.

Taking everything into account, it is believed that the strive for anonymisation within the blockchain infrastructure will continue. When all is said and done, the next years will determine whether cryptocurrencies will evolve into a mainstream payment tool that can also be used to revolutionize procedures like investments, real estate transfers, voting etc or whether they will end up being associated with illicit activities and fraudulent schemes.

## 9 References

1. Silk Road [https://en.wikipedia.org/wiki/Silk\\_Road\\_\(marketplace\)](https://en.wikipedia.org/wiki/Silk_Road_(marketplace))
2. Theft cases <https://www.vox.com/recode/2019/5/8/18537073/binance-hack-bitcoin-stolen-blockchain-security-safu>
3. ECDSA [https://en.bitcoin.it/wiki/Elliptic\\_Curve\\_Digital\\_Signature\\_Algorithm](https://en.bitcoin.it/wiki/Elliptic_Curve_Digital_Signature_Algorithm)
4. SHA-256 <https://en.bitcoin.it/wiki/SHA-256>
5. Double spending <https://coinsutra.com/bitcoin-double-spending/>
6. Block Reward <https://www.investopedia.com/terms/b/block-reward.asp>
7. Sarah Meiklejohn, Marjori Pomarole, Grant Jordan, Kirill Levchenko, Damon McCoy, Geoffrey M. Voelker, Stefan Savage, University of California, San Diego George Mason University, *A Fistful of Bitcoins: Characterizing Payments Among Men with No Names*
8. Fergal Reid and Martin Harrigan, Cliques Research Cluster, University College Dublin, Ireland, *An Analysis of Anonymity in the Bitcoin System*, 2011 IEEE International Conference on Privacy, Security, Risk, and Trust, and IEEE International Conference on Social Computing
9. Satoshi Nakamoto, "Bitcoin: A Peer-to-Peer Electronic Cash System", 2008
10. Bitcoin Faucet <https://freebitco.in/site/bitcoin-faucet/>
11. Bitcoin Forum <https://forum.bitcoin.com/>
12. MyBitcoin service <http://www.mybitcoin.com>
13. Pkred address  
<https://www.blockchain.com/btc/address/1KPTdMb6p7H3YCwsyFqrEmKGmsHqe1Q3jg>
14. Slush Pool <https://slushpool.com/>
15. Pkblue address  
<https://www.blockchain.com/btc/address/15iUDqk6nLmav3B1xUHPQivDpfMruVsu9f>
16. Pkgreen address  
<https://www.blockchain.com/btc/address/1J18yk7D353z3gRVcdbS7PV5Q8h5w6oWVG>
17. LulzSec <https://twitter.com/LulzSec/status/76388576832651265>
18. <https://www.investopedia.com/terms/b/blockchain.asp>
19. <http://bitcoin.org/bitcoin.pdf>
20. <https://investingalerts.com/2019/07/15/the-effect-of-cryptocurrency-on-money-laundering/>
21. Yaya J. Fanusie and Tom Robinson, *Bitcoin Laundering: An Analysis of illicit Flows into Digital Currency Services*, January 12, 2018
22. [https://en.wikipedia.org/wiki/Bitcoin\\_ATM](https://en.wikipedia.org/wiki/Bitcoin_ATM)
23. <https://www.statista.com/statistics/343127/number-bitcoin-atms/>
24. <https://www.elliptic.co/our-thinking/bitcoin-money-laundering>
25. <https://info.elliptic.co/whitepaper-fdd-bitcoin-laundering>
26. <https://en.wikipedia.org/wiki/CryptoLocker>
27. <https://www.symantec.com/security-center/writeup/2014-061923-2824-99>
28. <https://research.checkpoint.com/2018/jenkins-miner-one-biggest-mining-operations-ever-discovered/>
29. <http://cryptoshuffler.com/>
30. Dr. Nikolaos Theodorakis, "The Use of Cryptocurrencies for Illicit Activities and Relevant Legislative Initiatives", Stanford Law School

31. <https://en.wikipedia.org/wiki/Darknet>
32. <https://blog.nsfocusglobal.com/categories/armada-collective-ddos-attack/>
33. <https://www.bbc.com/news/business-34743185>
34. [https://en.wikipedia.org/wiki/Pump\\_and\\_dump](https://en.wikipedia.org/wiki/Pump_and_dump)
35. <https://www.investopedia.com/terms/p/ponzischeme.asp>
36. <https://thenextweb.com/hardfork/2019/07/16/whats-a-cryptocurrency-exit-scam-and-how-do-i-spot-one/>
37. Alex Biryukov and Ivan Pustogarov, University of Luxembourg, "Bitcoin over Tor isn't a good idea", 2015 IEEE Symposium on Security and Privacy
38. <https://coinmarketcap.com/currencies/anoncoin/>
39. <https://www.coindesk.com/bittorrents-master-plan-to-bring-a-tron-powered-crypto-token-to-the-masses>
40. <https://coinmarketcap.com/currencies/torcoin-tor/>
41. <https://coinmarketcap.com/currencies/stealth/>
42. Spoofing attack <https://www.malwarebytes.com/spoofing/>
43. OnionCat [https://www.whonix.org/wiki/Onion\\_Services](https://www.whonix.org/wiki/Onion_Services)
44. Tor <https://fossbytes.com/everything-tor-tor-tor-works/>
45. Ledger <https://blockexplorer.com/>
46. Michael Doran, "A Forensic Look at Bitcoin Cryptocurrency", SANS Institute Information Security Reading Room
47. Bitcoin-Qt <https://en.bitcoinwiki.org/wiki/Bitcoin-Qt>
48. Steven Rigby, BYU-Idaho, Rexburg, ID USA and Marcus K. Rogers, "Forensics Digital Model The General Digital Forensics Model", Purdue University, West Lafayette, USA
49. Internet Evidence Finder <https://www.teeltech.com/mobile-device-forensic-tools/magnet-forensics/magnet-internet-evidence-finder-ief/>
50. Multibit <https://en.bitcoinwiki.org/wiki/Multibit>
51. Bitminter <https://bitminter.com/>
52. ASIC <https://www.techopedia.com/definition/2357/application-specific-integrated-circuit-asic>
53. Bitcoin Mining Rig <https://www.investopedia.com/tech/usb-bitcoin-mining/>
54. EnCase <https://www.guidancesoftware.com/encase-forensic>
55. Internet Evidence Finder (2) <https://www.forensicfocus.com/c/aid=59/webinars/2013/internet-evidence-finder-ief-advanced-edition-v61/>
56. bitcoinj <https://en.bitcoin.it/wiki/Bitcoinj>
57. Tableau Imager <https://digital-forensics.sans.org/blog/2010/02/16/tableau-imager-first-look>
58. Winen.exe <http://forensiczone.blogspot.com/2008/06/winenexe-ram-imaging-tool-included-in.html>
59. Silicon Labs <https://www.silabs.com/>
60. Luuc Van Der Horst, Kim-Kwang Raymond Choo and Nhien-An Le-Khac, "Process Memory Investigation of the Bitcoin Clients Electrum and Bitcoin Core", IEEE
61. Bitcoin Core <https://bitcoin.org/en/bitcoin-core/>
62. Electrum <https://electrum.org/#home>
63. Berkeley DB [https://en.wikipedia.org/wiki/Berkeley\\_DB](https://en.wikipedia.org/wiki/Berkeley_DB)
64. Volatility <https://www.volatilityfoundation.org/25>
65. Ethereum <https://en.wikipedia.org/wiki/Ethereum>
66. Smart Contracts [https://en.wikipedia.org/wiki/Smart\\_contract](https://en.wikipedia.org/wiki/Smart_contract)
67. Vyper Programming Language <https://vyper.readthedocs.io/en/v0.1.0-beta.13/>
68. LLL Programming Language [https://lll-docs.readthedocs.io/en/latest/lll\\_introduction.html](https://lll-docs.readthedocs.io/en/latest/lll_introduction.html)



69. Bamboo Programming Language <https://github.com/pirapira/bamboo>
70. Solidity Programming Language <https://solidity.readthedocs.io/en/v0.5.12/>
71. Ethereum Virtual Machine <https://www.mycryptopedia.com/ethereum-virtual-machine-explained/>
72. Robert S. Boyer, Bernard Elspas and Karl N. Levitt, *A formal system for testing and debugging programs by symbolic execution*, In Proceedings of the International Conference on Reliable Software
73. Dynamic Symbolic Execution [https://link.springer.com/chapter/10.1007/978-3-642-45422-6\\_1](https://link.springer.com/chapter/10.1007/978-3-642-45422-6_1)
74. Trail of Bits <https://www.trailofbits.com/>
75. DARPA <https://blog.trailofbits.com/category/cyber-grand-challenge/>
76. Christof Ferreira Torres, Mathis Steichen and Radu State, *"The Art of The Scam : Demystifying Honeypots in Ethereum Smart Contracts"*, SnT, University of Luxembourg
77. Manticore Mark Mossberg, Felipe Manzano, Eric Hennenfent, Alex Groce, Gustavo Grieco, Josselin Feist, Trent Brunson, Artem Dinaburg, *"Manticore: A User-Friendly Symbolic Execution Framework for Binaries and Smart Contracts"*, Trail of Bits, New York City, USA
78. <https://pentesttools.net/manticore-symbolic-execution-tool-for-analysis-of-binaries-and-smart-contracts/>
79. <https://github.com/trailofbits/manticore>
80. <https://medium.com/haloblock/introduction-to-manticore-a-symbolic-analysis-tool-for-smart-contract-9de08dae4e1e>
81. <https://www.cyberpunk.rs/symbolic-execution-tool-manticore>