Cuckoo Filter Approach for an Efficient Tainted Bitcoin Identification

E.W.M.U.I Pitawela

Cuckoo Filter Approach for an Efficient Tainted Bitcoin Identification

E.W.M.U.I Pitawela

Index No: 14001187

Supervisor: Dr. C.I. Keppetiyagama

December 2018

Submitted in partial fulfillment of the requirements of the

B.Sc in Computer Science Final Year Project

(SCS4124)

Declaration

I certify that this dissertation does not incorporate, without acknowledgement, any material previously submitted for a degree or diploma in any university and to the best of my knowledge and belief, it does not contain any material previously published or written by another person or myself except where due reference is made in the text. I also hereby give consent for my dissertation, if accepted, be made available for photocopying and for interlibrary loans, and for the title and abstract to be made available to outside organization

Candidate Name: E.W.M.U.I Pitawela

………………………………………………

Signature of Candidate Date:

This is to certify that this dissertation is based on the work of Ms. E.W.M.U.I Pitawela under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle/Co- Supervisor's Name: Dr. C.I. Keppetiyagama

………………………………………………

Signature of Supervisor Date:

Abstract

Bitcoin has become one of the most popular cryptocurrency and it is the first and foremost cryptocurrency that comes to play as an alternative to the traditional currency. Even though bitcoins have lots of advantages a certain group of people tend to use these bitcoins for illegal purposes. So there comes the need of identifying these bitcoins before their usage.

Probabilistic approach has been taken through this research in order to make this tainted bitcoin identification efficient. Cuckoo filter has a better performance when compared to other probabilistic data structures. This is the very first occasion that Cuckoo filters have been taken for bitcoin and blockchain analysis purposes. The evaluation of the Cuckoo filter is conducted with respect to time and space consumption by utilizing some accepted space and time consumption measuring tools.

Through this research it is successfully showed that tainted bitcoin identification process can be made efficient mainly with respect to time, where the authenticity of a given transaction can be checked in constant time by preserving its accuracy rate with the aid of Cuckoo filters. Additionally, deletion of tainted transactions from the data structure also can be done with Cuckoo filters. Therefore, the proposed tainted bitcoin identification with Cuckoo filter approach, can be considered as a significant contribution to the body of knowledge.

Preface

Probabilistic data structures have been taken for blockchain and bitcoin analysis purposes. Tainted bitcoin identification is the purpose of analyzing blockchain transactions and transaction details. Probabilistic data structures do not provide answers for membership with a 100% accuracy but with a certain accepted accuracy with an acceptable false positive rate.

Utilizing Cuckoo filters as one of the probabilistic data structures for blockchain analysis is a novel approach and it is evaluated with respect to time and space consumption.

An existing Cuckoo filter has been taken for the analysis purposes but the analysis of tainted bitcoin identification through this Cuckoo filter is solely my contribution in conjunction of my supervisor.

Acknowledgement

I would like to express my sincere gratitude to my research supervisor, Dr. C. I. Keppetiyagama, senior lecturer of University of Colombo School of Computing and my research co-supervisor, Dr. A.M.Premachandra, senior lecturer of University of Colombo School of Computing for providing me continuous guidance and supervision throughout the research.

I also take the opportunity to acknowledge the assistance provided by Dr. H. E. M. H. B. Ekanayake as the final year computer science project coordinator.

I appreciate the feedback and motivation provided by my friends to achieve my research goals. This thesis is also dedicated to my loving family who has been an immense support to me throughout this journey of life. It is a great pleasure for me to acknowledge the assistance and contribution of all the people who helped me to successfully complete my research.

Table of Contents

List of Figures

List of Tables

List of Acronyms

SML	Stone Man Loss
AIV	All In Vain
MMT	Mass Mybitcoin Theft
AQP	Approximate Query Processing

Chapter 1

Introduction

1.1 Background and Introduction

Bitcoin has become one of the most popular cryptocurrency and it is the first and foremost cryptocurrency, which comes to play as an alternative to the traditional currency. Bitcoin maintains a high profile through numerous means of social media which eventually make it more popular in the field of cryptocurrency. High price and the availability of various investment options also have become reasons for Bitcoin's popularity.

Each and every Bitcoin transaction should be recorded in the public ledger called Blockchain. Even though the transactions are recorded, the users behind the transactions are not revealed. Hence Bitcoin is pseudo-anonymous[1]. Thus, some people temp to use it for nefarious purposes such as money laundering, stolen funds, contraband deals etc [2], [3] . Other than the pseudo-anonymous feature,the following reasons also have caused for Bitcoin's illegal activities.

- No proper way to retrieve the stolen funds
- No proper mechanism to revert the transaction

In order to discourage these illegal activities precautions should be made before these activities take place. Hence, there is a mechanism to blacklist the bitcoins used for illegal activities by giving a special name called tainted bitcoins. The usage of tainted bitcoins are discouraged as they are less liquid, less valuable and hard to spend. Three heuristics have been defined with the aim of identifying tainted bitcoins namely Poison, Haircut and FIFO[2]–[4].

Table 1.1: Heuristics

Name	Description
Poison	This makes any transaction invalidated if it has at least one dirty predecessor
Haircut	Transactions are devalued proportionally according to the amount of blacklisted bitcoins.
First In First Out (FIFO)	The order of input determines which outputs are affected by blacklisting

BlockSci is an open-source software platform which is implemented for blockchain analysis purposes[5]. BlockSci is versatile and it supports for different blockchains and analysis tasks. It incorporates an in-memory, analytical (rather than transactional) database, making it several hundred times faster than existing tools. BlockSci has its own tools and implementations for clustering, data parsing and for different heuristics. This has implemented the three heuristics and it outputs the details of unspent transactions which have previously combined with tainted bitcoins[6]. BlockSci's core infrastructure is written in C++ and is optimized for speed. To make analysis more convenient, BlockSci has provided Python bindings and Jupyter notebook interface.

1.2 Research Problem and Research Questions

1.2.1 Research Problem

The three heuristics output the details of unspent transactions and this can be considered as the one of the main tools where bitcoin users are given the facility to obtain unspent transactions when the tainted hashes are given as the input. One of the main drawback in the current implementation is that it consumes high time and space in order to produce results. There is a norm in Bitcoin that 7-30 transactions happen per second. This means that every second matters in Bitcoin.

Few transactions were taken as tainted transactions which ranges from 2018 to 2009 with numerous number of unspent transactions. Time consumption for all the unspent transactions in Poison and Haircut methods are represented in below graphs.

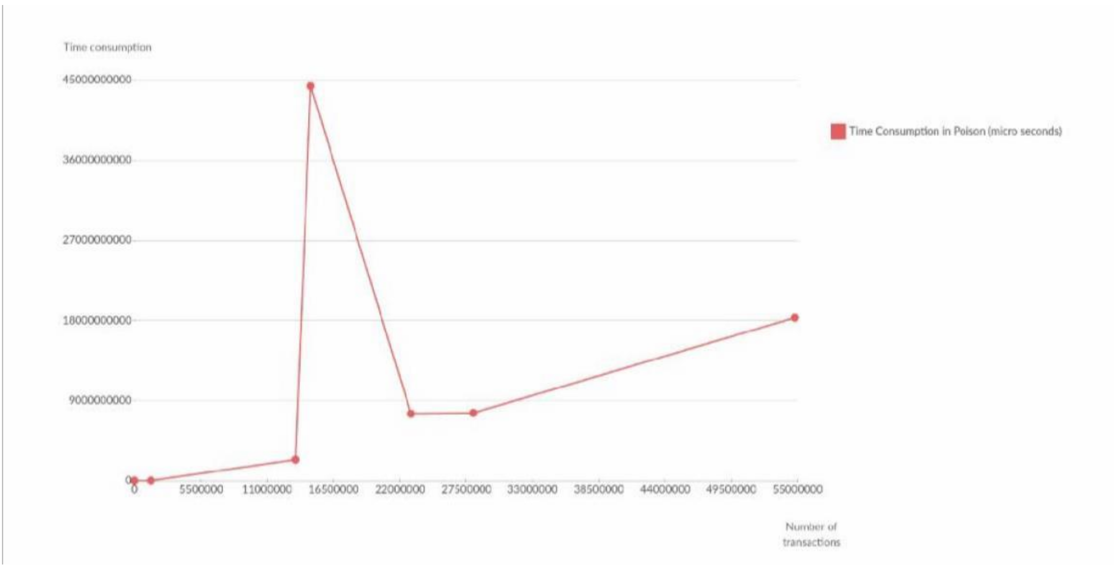


Figure 1.1: Time Consumption to obtain unspent transactions in Poison method

As it is depicted in the graph above, with the number of unspent transactions time consumption has been increased and after some point time consumption started to reduce. This is because when same unspent transactions are queried it consumes a lesser time as it already in the cache.

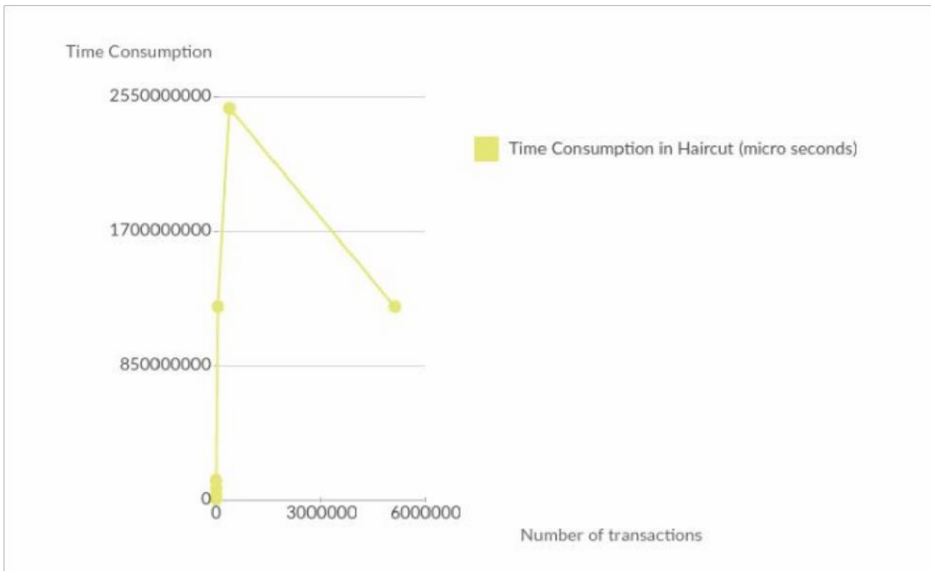


Figure 1.2: Time consumption to obtain unspent transactions in Haircut method

The above represents time consumption in the Haircut method. Same tainted transactions used for the Poison method has been used. According to above diagrams, some tainted transactions in Poison method has been taken almost 5 hours to produce results. Whenever a user needs to check the authenticity of a transaction, he or she has to use these heuristic methods which consumes high time, in order to produce results.

The memory consumption is captured for one of the tainted transactions taken for time consumption graphs, which had the maximum number of unspent transactions. Poison method is used when retrieving the unspent transactions.

Table 1.2: Space consumption to obtain unspent transactions in Poison method.

Virtual Memory	Resident Memory	Shared Memory
89.8Gb	14.2Gb	12.7Gb

According to memory consumption, total virtual memory requested by this tainted function is 89.8GB. Even though this is requested the total amount is not consumed during the running time of the function. The shared memory and physical memory consumed by each process is stated under SHR and RES.

Memory consumption is captured for the same tainted transaction using Haircut method.

Table 1.3: Space consumption to obtain unspent transactions in Haircut method.

Virtual Memory	Resident Memory	Shared Memory
88.5Gb	14.9Gb	14.3Gb

In Haircut method the total virtual memory requested by the tool is 88.5GB. This amount is slightly lower than the Poison method. This may be because Haircut method is less complex than Poison method.

Both methods consume high virtual memory consumption which is a problem for many users to run these tainted functionalities again and again when they need to check the authenticity of a given transaction.

Even if the results obtained with high time and memory consumption, it is a difficult task for users to check whether a particular transaction is tainted or not because through the current implementation it outputs millions of unspent transactions. There is no proper easy mechanism for a bitcoin user to check whether a particular transaction is tainted or not. This can be considered as a major shortcoming in tainted bitcoin identification.

1.2.2 Research Questions

Solutions to the above identified problems can be taken in several aspects. Restructuring the data representation and data accessing method is one of the algorithmic efforts that can be taken to make the tainted bitcoin identification process efficient. Probabilistic data structures will be utilized as the data structure and through this research their suitability to tainted bitcoin identification, how these data structures increase its performance by retaining the accepted time and space consumption also will be analyzed. The analysis will be conducted through below research questions.

1.2.2.1 Question 01

Is it possible to utilize a probabilistic/synopsis data structure which can eventually make the tainted bitcoin identification process efficient by reducing the time consumption?

1.2.2.2 Question 02

Is it possible to utilize a probabilistic/synopsis data structure which can eventually make the tainted bitcoin identification process efficient by reducing the space consumption?

1.3 Research Aim and Objectives

1.3.1 Research Aim

The main aim of this research was to identify the possibility of using probabilistic data structures for an efficient tainted bitcoin identification.

1.3.2 Research Objectives

The objectives of the research are as follows:

- Identify a suitable probabilistic data structure
- Identify the necessary parameters when initializing the data structure
- Implement tainted bitcoin identification tool
- Evaluate the performance of the data structure and compare results

1.4 Justification of the Research

Nowadays, bitcoin enthusiasts have a great desire to check the authenticity of bitcoins to know whether they are an outcome of a blacklisting practice or not. So, the current tainted bitcoin identification implementations need to be optimized in an efficient manner. This is the intended scientific contribution through this research.

Probabilistic data structures have a higher performance, mostly when it comes to membership queries. Utilizing them for blockchain analysis can be considered as a significant computational contribution. Identifying possible probabilistic data structures, their suitability to tainted bitcoin identification process will be analyzed which is the computational contribution of this research.

1.5 Methodology

Constructive research methodology will be followed throughout the research. As the initial step, research problems were identified, as high space/time consumption incurred for the tainted bitcoin identification process and having no proper mechanism to find the authenticity of a given transaction. So, the attempt through this research is to find the suitability of probabilistic data structures for tainted bitcoin identification.

As the second step, detailed literature review is conducted regarding three heuristics and its operation. At the same time possibility of using different probabilistic data structures is assessed, their current applications, benefits they have, applicability to the blockchain is also assessed by considering the unique features of blockchain.

Research design and the evaluation will be descriptively explained in the coming sections. Final step would be to present the research findings in a form of a research paper and the thesis.

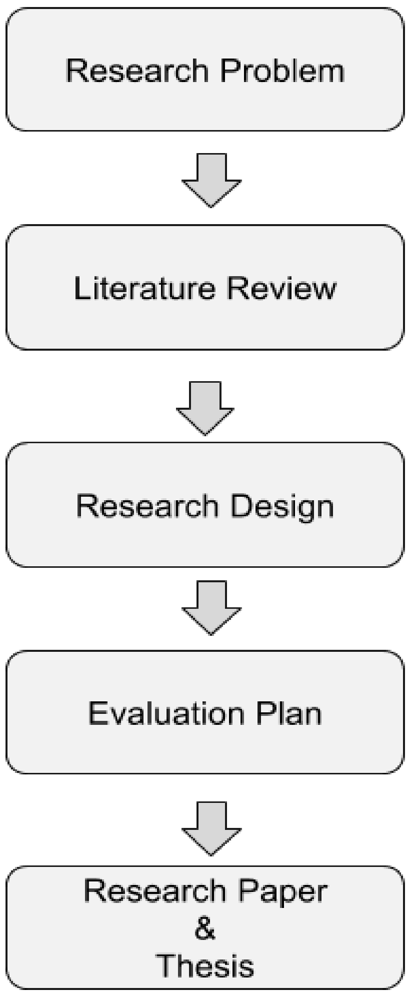


Figure 1.3: Research Methodology

1.6 Outline of the Dissertation

The first chapter of thesis includes the introduction of this research. Introduction includes research problem, research methodology, scope and delimitations, key assumptions and other introduction details for the research. The next chapter of the thesis will be literature review. This chapter includes the prior researches done towards tainted bitcoin identification, different probabilistic data structures, how they produce results, comparison of different probabilistic data structures etc. The third chapter includes the design of the research solution and fourth chapter includes the research implementation with the necessary tools utilized when providing the solution. The next chapter includes the evaluation of the research where the accuracy, time and space consumption of the new solution will be analyzed. Final chapter is the conclusion where overall conclusion is made based on the results of the research solution.

1.7 Scope and Delimitations

1.7.1 Scope

The research tries to restructure the data representation with the aim of making the tainted bitcoin identification efficient. The suitability of various probabilistic data structures will be analyzed and finally the accuracy of the results, the space and time consumption will be analyzed.

1.7.2 Delimitations

The research does not try to make any changes to current heuristics. The new probabilistic data structure caters only to make the tainted bitcoin identification process efficient and it might not applicable for other analysis purposes.

1.8 Summary

This chapter laid the foundations for the research. The problems caused for this research were the high time and space consumption for tainted bitcoin identification and no proper way to check the authenticity of a given transaction. Thus, the aim of this research is to make the tainted bitcoin identification process efficient in a probabilistic manner. All the unspent transactions are queried through a heuristic method and no change will be done to those heuristics and it is out of the scope of this research. On these foundations, the dissertation would proceed with a detailed description of the research.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, a review on related work regarding the approaches taken for tainted bitcoin identification with three heuristics are included in the section 2.2. Section 2.3 includes details about different probabilistic data structures and their unique features. Section 2.4 represents other related work which were not discussed in the previous sections.

2.2 Tainted Bitcoin Identification

Bitcoin has got a vast amount of popularity because of its unique features when compared to traditional currency. Absence of a central authority, pseudo-anonymity are some of the prominent features of Bitcoin[1]. Some of its features like pseudo-anonymity [7] badly influenced some bitcoin users for which they use it for nefarious purposes like contraband deals, money laundering etc [2], [3]. The tainted bitcoins used for these kind of transactions are blacklisted so that these bitcoins are less valuable, less liquid and hard to spend. Hence bitcoin users has a great tendency in finding a mechanism to identify blacklisted bitcoins before dealing with transactions. With that intention three heuristics have been defined namely Poison, Haircut and FIFO by M. Möser, R. Böhme, and D. Breuker [3] and Anderson et al[2].

Out of the three heuristics Poison is the drastic policy in which transactions with at least a single tainted bitcoin is enough for the whole transaction considered to be invalidated. When compared to Poison, Haircut has a less drastic policy where the devaluation is done proportionally according to the amount of bitcoins associated with a given address[3].

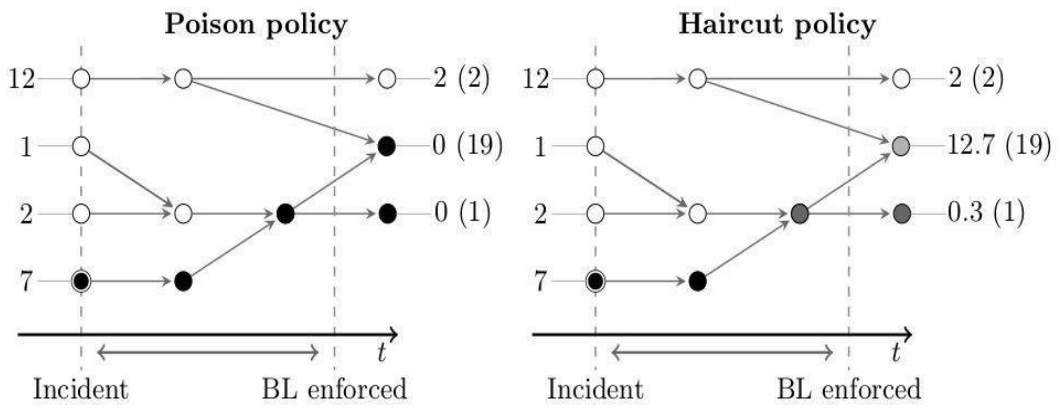


Figure 2.1: Poison tainted calculation, Haircut tainted calculation [3]

A clear characteristic of a blacklisted bitcoin is that it tempts to transfer through the network rapidly compared to other bitcoins. Hence order of transactions is a good feature to identify blacklisted bitcoins which has become the ground principle of FIFO method. FIFO devalues blacklisted bitcoins by considering the order of inputs[2], [3].

Harry, et al [5] proposed the tool BlockSci which is an open source software platform used for blockchain analysis. According to Anderson, Ross, et al [4] BlockSci has the tainted bitcoin identification implementation [6]. In BlockSci implementation when a hash value of a tainted address given as a parameter it outputs the details of a list of unspent transactions. This mechanism is failed to provide a solution as to check the authenticity of a given transaction. Even though BlockSci has done the implementation on these heuristics, getting hash values of blacklisted transactions is much of a difficult task. Currently, tainted addresses are maintained by the users of the "Bitcoin Forum"[8]. As there is a delay already in getting blacklisted transactions, the tainted bitcoin identification process becomes useless if it is not efficient. Hence the tainted bitcoin identification process must be efficient as much as possible despite the delay of getting tainted transactions.

2.3 Probabilistic Data Structures

Large data analysis has become one of the prominent research avenue. Number of unique items, most frequent item, existence of an item in the data set are some of the queries that needs to be answered when analyzing data. The common approach is to use a deterministic data structure like a hashSet or a hashmap and obtain results with error free accuracy. But when the data set becomes very large in quantity wise, it is difficult to deal with the mentioned data structures as they are not fit into memory. At the same time, those data structures are not fit enough for streaming application which typically require data to be processed in one pass and perform incremental updates.

Probabilistic data structures can be considered as an alternative to the above mentioned deterministic data structures which are extremely useful for big data and streaming applications. These data structures use hash functions to include data in the data structure and they are included in a compact form. Collisions are ignored in these data structures but errors can be well-controlled under certain thresholds.

There are different probabilistic data structures such as Bloom filters, Cuckoo filters, Count Min sketches which are descriptively mentioned in the below sub sections.

2.3.1 Bloom Filters

Bloom Filters are one of the most popular probabilistic data structures which can be used for high-speed set membership queries. Bloom filters use a hashing mechanism to store details in its data structure. As a result this performs a constant searching time in Big-O notation when answering the membership queries. The below diagram shows how the querying process has become efficient due to hashing technique instead of a typical array like data structure.

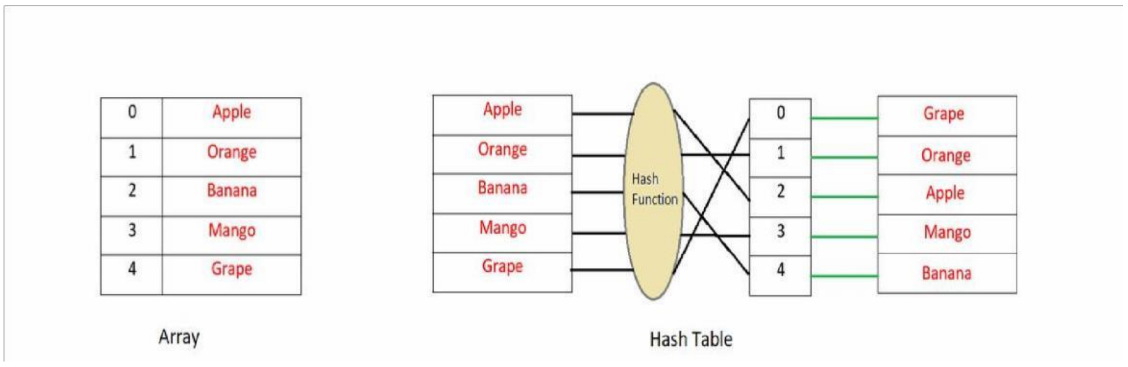


Figure 2.2: Comparison between arrays and hash tables [9]

Bloom filters give binary answers to membership's queries which means it replies yes or no for questions such as "Is this included in the dataset?" Before answering it should go through a hash function and fill the filter with data. These hash functions depend on the number of items expected to have in the Bloom filter and the acceptable false positive rate[10]. Bloom filters provide two answers. They are yes: which the particular item is included in the given set and no: which the item is not in the set. The special feature in this data structure is when the answer is 'No' then the item is definitely not in the dataset while answer "Yes" having two cases. When it is "Yes", case1 is having the item in the set while the case2 is item not actually in the set but is given as "yes" because of another item which also put onto the same location [10].

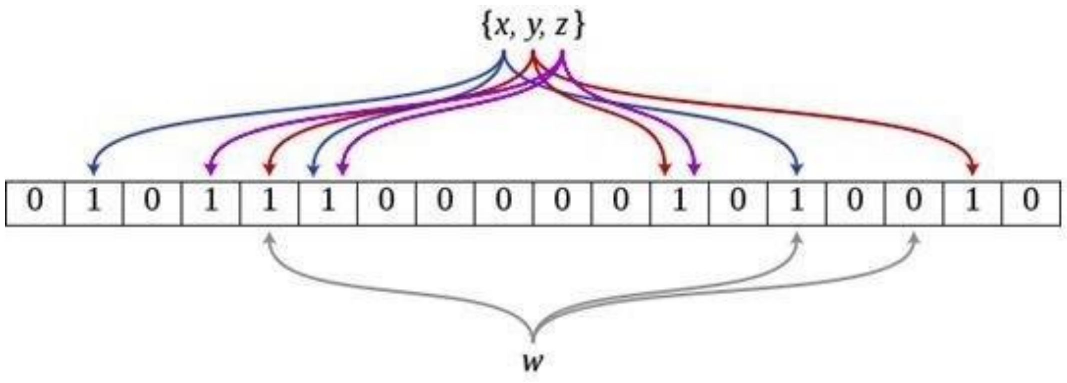


Figure 2.3: Bloom filter representation [10]

Even though Bloom filters help to reduce the overhead of evaluating membership queries, it has its own drawbacks. One of the main drawbacks is that it does not support item deletion where in a case of deletion it requires entire filter to rebuild again[10], [11].

According to Harry et al[5], BlockSci uses a LRU cache and a Bloom filter to passe data from the blockchain for analysis purposes. Cache contains 6.8% amount of addresses out of total addresses and it is updated using Least Recently Used policy. Bloom filter contains list of seen addresses. When an address is not in the cache, bloom filter is queried before the database. Bloom filters are the one of the most popular probabilistic data structures where they present results probabilistically. Hence the overhead of querying data from the disk is reduced due to the usage of a Bloom filter. This is one of the main bitcoin related use case where probabilistic data structures have been utilized for blockchain analysis purposes.

There are different kinds of Bloom filters implemented by embedding different unique features. According to Fan, Bin, et al[11],some of them can be listed as Standard bloom filters, Counting bloom filters, Blocked bloom filters, d-left counting bloom filters, Quotient filters etc[12].

2.3.1.1 Counting Bloom Filters

Unlike standard bloom filters, counting bloom filters support deletion operation. Counting bloom filters use an array of counters instead of an array of bits[13]. In a case of an insertion it increments the value of k counters, in lieu of setting k bits. When in a lookup operation, it check if each of the required counters is non-zero. The delete operation decrements the value of each respective counters. Counting bloom filters consumes four times more space than a standard bloom filter.

Even though counting bloom filters performs the delete operation, it performs at the cost of additional space[12].

2.3.1.2 Blocked bloom filters

Blocked bloom filters do not support deletion, but provide better spatial locality on lookups[12]. Blocked bloom filters consist of an array of small bloom filters, each fitting in one CPU cache line. Each item is stored in one of these small filters and the hash partitioning technique is used as the mechanism. Due to the imbalanced load across the array of small bloom filters, the false positive rate becomes much higher than the other bloom filters.

2.3.1.3 d-left Counting bloom filters

These bloom filters use d-left hashing and use fingerprints when keeping items. These filters remove items by removing their fingerprint. These bloom filters have reduced the space cost by 50% compared to counting bloom filters[12].

2.3.1.4 Quotient filters

These bloom filters also support deletion operation though the use of fingerprints. These filters use a technique similar to linear probing to locate a fingerprint. But Quotient filters require additional meta-data to encode each entry, which requires 10-25% more space than a standard bloom filter. The performance of this data structure deteriorates when the occupancy of the hash table exceeds 75%.

2.3.2 Cuckoo filters

Cuckoo filter is also a probabilistic data structure and it is one of best alternative to Bloom filters. Cuckoo filters consist of Cuckoo hash tables which has array of buckets where each item has two buckets. Cuckoo filters consists of two hash functions corresponding to the mentioned two buckets. The lookup operations checks in both buckets where a particular item is included in a set or not[12], [14].

According to Fan, Bin, et al [12] Cuckoo filters use lots of alternative solutions to make it look up operations efficient and also to achieve high-space efficiency. Using partial cuckoo hashing instead of traditional cuckoo hashing is one of such alternative solutions. The main target of using partial cuckoo hashing technique is to achieve space efficiency[15]. Partial cuckoo hashing deals only with fingerprints which reasons for its better performance compared to Bloom filters. When fingerprints are stored there is no other way to restore or rehash original keys to find their alternate solutions. Hence partial cuckoo hashing technique uses fingerprint to calculate the alternate location[12], [16]. Before the insertion of a single item hashing scheme calculates indexes of the two candidate buckets.

$$
\begin{aligned}
h_1(x) &= \text{hash}(x), \\
h_2(x) &= h_1(x) \oplus \text{hash}(x\text{'s fingerprint}).
\end{aligned}
$$

This simply means when it is needed to identify the alternate location of an item, get the index of the current location, apply the hash function to the fingerprint value of that location and then apply 'xor' operation to both values to get the alternate location. Hence, this makes the need to retrieve the original value of a fingerprint useless because the alternate location can be simply obtained from the information in the bucket.

When it comes to lookup performance Cuckoo filters return false if the item is definitely not in any of the two buckets, but if it's true there are two possible cases as in Bloom filters. But the false positive rate in Cuckoo filters are really low compared to Bloom filters. In Cuckoo filters there do not exist false negatives as long as bucket overflow never occurs[12].

One of the main advantage of Cuckoo filters is that they support delete operations without harming the filter performance and its accuracy. It checks both candidate buckets for a given item and if any fingerprint matches in any bucket, one copy of that matched fingerprint is eliminated from the bucket.

Another Cuckoo filter implementation is adding a stash for each sub-table of the filter with the aim of improving its success probability which is proposed by Eppstein, David[16]. A stash is a collection of key-value pairs where each stash stores a collection of <location, fingerprint> pairs, for fingerprints that were not be able to store within its sub-table. To perform a query, first checks in both locations and if it was not included, the stash is also searched. This causes an additional sequence of memory access but not harm its performance by slowing down the searches nor degrades its reliability.

According to Eppstein, David [16] Blocked Cuckoo hashing is also another implementation of Cuckoo filters where each cell of the hash table stores a block of different key-value pairs. When a key is inserted and one of its two cells in not full it may be placed directly. However when both cells are full one of the keys already placed should be kicked out, and moved to another location. Replacing keys are formulated in a sequence by which a chain of dislocations are created. These sequence of moves can be viewed as an augmenting path in a graph where vertices can be defined as table cells and edges are pairs of cells that each key maps to. It performs constant expected time in breadth-first algorithm for finding these augmenting paths.

2.3.3 Count min sketch

According to Cormode, Graham[17], [18], Count min sketch is a probabilistic sub-linear space data structure which is capable of representing a high-dimensional vector and answering queries on this vector, mostly point queries and product queries with strong accuracy guarantees. To a certain extent count min sketch is similar to bloom filters but compared to bloom filters count min sketch represents a multi-set while bloom filters represent a bitmap[10].

The count min sketch data structure consists of two dimensional dxw array of counters with d number of independent pairwise hash functions h1…….hd of range w. Here w can be defined as $w = [e/\varepsilon]$, and d as $[ln1/\delta]$. ε is the required accuracy level and δ is the certainty to reach the accuracy. To increment the accounts in the two dimensional array

, hash positions should be calculated with d hash functions and update the counts at respective positions.

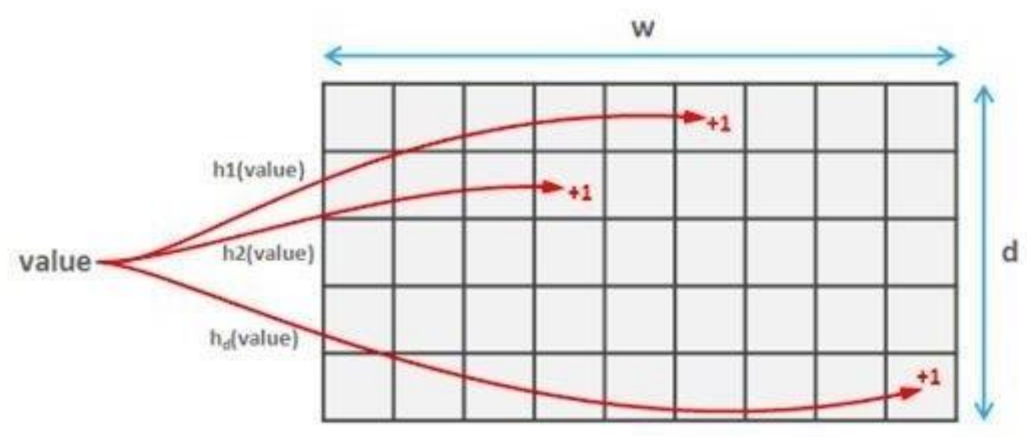


Figure 2.4: Count min sketch representation [17]

The count estimate for an item is the minimum value of the counts at the array positions determined by the d hash functions. Error can be minimized and the accuracy can be increased by choosing appropriate values for d and w. This performs better accuracy for high frequent items.

2.4 Other Related Work

The following papers include findings on practical applications of probabilistic data structures, different probabilistic/synopsis data structures which have not been mentioned in the prior sections.

Analysis of Bitcoin Network Dataset for fraud, Zambre, Deepak, and Ajey Shah 2013 [19]

The aim of this paper is to identify peculiar properties of bitcoin users carrying out heists or robbery. Three types of robberies are discussed in this paper namely Stone Man Loss (SML), All In Vain (AIV) and Mass Mybitcoin Theft (MMT). A classification technique is used to accomplish the aim of this research. K-means algorithm is used for the model with six different features. When the features are fed into the model it classifies users into two clusters which separate good users from rogue users. One of the shortcomings in this model is that it cannot detect any robberies that happen over an extended period of time in small increments. For synthetically generated rogue users, this model performs 76.5% accuracy, but there exists an ambiguity whether this model performs the same but there exists an ambiguity whether this model performs the same accuracy in a real situation.

Synopsis data structures for massive data sets, Gibbons, Phillip B., and Yossi Matias, 1999 [20]

This paper mainly focused on the uses of synopsis data structures. Synopsis data structures use very little and provide fast approximate answers to queries. A number of concrete examples of synopsis data structures and the way of keeping them up-to-date in the presence of online updates to the data sets are also analyzed.

The two main challenges when defining a synopsis data structure

- What synopsis of the full data set to keep in the limited space in order to maximize the accuracy and confidence of its approximate responses
- How to efficiently compute the synopsis and maintain it in the presence of updates to the data set

Synopsis data structures are used in a heuristic way, which do not have proven any formal properties regarding performance or accuracy under the presence of updates in the dataset.

Network Applications of Bloom Filters: A survey, Broder, Andrei, and Michael Mitzenmacher, 2002 [21]

The aim of this paper is to survey the ways in which Bloom Filters have been used and modified in a variety of network problems. Bloom filters can be used to summarize data in peer to peer networks. Some applications have utilized bloom filters such as approximate set reconciliation for content delivery, set intersection for keyword searches etc.

A drawback of bloom filter is that it allows false positives, but it is acceptable to a certain extent for large data analysis.

Synopses for Massive Data: Samples, Histograms, Wavelets, Sketches, Cormode, Graham, et al, 2011 [22]

This paper explicates on Basic principles and recent developments related to Approximate Query Processing (AQP). It is mainly focused on key synopses: random samples, histograms, wavelets and sketches and issues such as accuracy, space, time efficiency, optimality, practicality, error bounds on query answers and incremental maintenance.

Sampling well suited for broad patterns and has a poor performance on applications such as fraud or anomaly detection. Sampling gives lower accuracy when the data is highly variable. Histograms can be used for estimation purposes but do not suitable for higher dimensional data. They also do not perform well when the data is dynamic and shifting. Wavelets are well suitable for capturing nonlocal structures and under dynamic situations but perform less when extending to higher dimensions. Sketches are very efficient to process updates to data. Sketching algorithms rely on hash functions.

2.4 Summary

This chapter mainly focused on providing an extensive review regarding tainted bitcoin identification and probabilistic data structures. Initially, it discussed the need and steps taken for tainted bitcoin identification, main heuristics defined to obtain unspent transactions related to a tainted transaction. Next a detailed review was done regarding different probabilistic data structures, their basic structure along with their unique features and drawbacks.

Chapter 3

Design

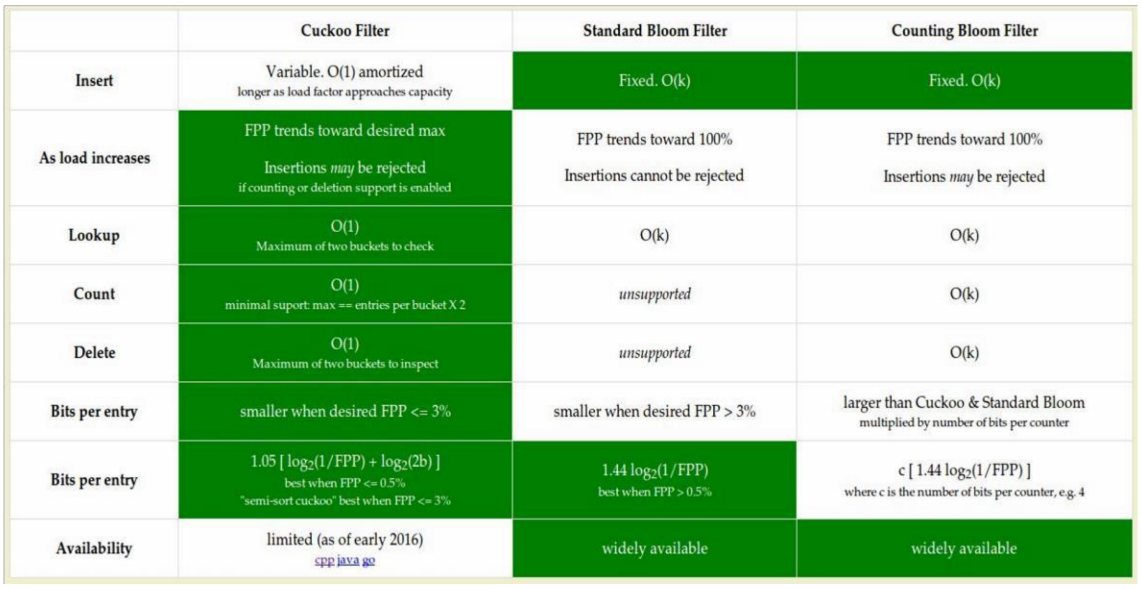
3.1 Introduction

This chapter explicates the proposed solutions to the research problem. Section 3.2 includes overall research design and Cuckoo filter design is included in the section 3.3.

3.2 Research Design

The proposed solution is designed in such way where users can easily identify the authenticity of a transaction. This design is provided based on a probabilistic approach which means that the tainted bitcoin identification tool is designed by utilizing a probabilistic data structure which is a Cuckoo filter.

As mentioned in the literature review, there exists some popular probabilistic data structures such as Bloom filter, Count min sketch, Cuckoo filter. Out of all the probabilistic data structures Cuckoo filters have chosen for the proposed solution as they have a better performance compared to other probabilistic data structures. A simple comparison is depicted in the following table which includes a comparison among the some probabilistic data structures.



	Cuckoo Filter	Standard Bloom Filter	Counting Bloom Filter
Insert	Variable. O(1) amortized longer as load factor approaches capacity	Fixed. O(k)	Fixed. O(k)
As load increases	FPP trends toward desired max Insertions *may* be rejected if counting or deletion support is enabled	FPP trends toward 100% Insertions cannot be rejected	FPP trends toward 100% Insertions *may* be rejected
Lookup	O(1) Maximum of two buckets to check	O(k)	O(k)
Count	O(1) minimal suport: max == entries per bucket X 2	*unsupported*	O(k)
Delete	O(1) Maximum of two buckets to inspect	*unsupported*	O(k)
Bits per entry	smaller when desired FPP <= 3%	smaller when desired FPP > 3%	larger than Cuckoo & Standard Bloom multiplied by number of bits per counter
Bits per entry	1.05 [$\log_2(1/FPP) + \log_2(2b)$] best when FPP <= 0.5% "semi-sort cuckoo" best when FPP <= 3%	1.44 $\log_2(1/FPP)$ best when FPP > 0.5%	c [1.44 $\log_2(1/FPP)$] where c is the number of bits per counter, e.g. 4
Availability	limited (as of early 2016) cpp java go	widely available	widely available

Figure 3.1: Comparison of Cuckoo filters and bloom filters [23]

As per the above table, it is apparent that Cuckoo filters have a better lookup performance when it comes to membership queries. Its running time has a constant time complexity which makes it totally qualified for the proposed solution. Additionally, Cuckoo filter have another unique feature where it supports for item deletion which the other data structures do not have.

The below graphs further strengthen the fact that Cuckoo filters have a better performance compared to Bloom filters. They were drawn not considering the bitcoin context, but considering a general instance.

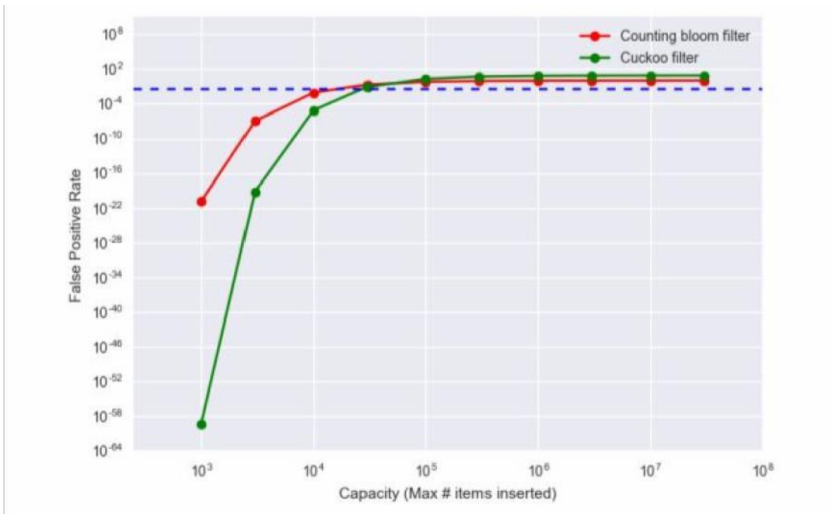


Figure 3.2: Cuckoo filter and counting bloom filter capacity comparison [24]

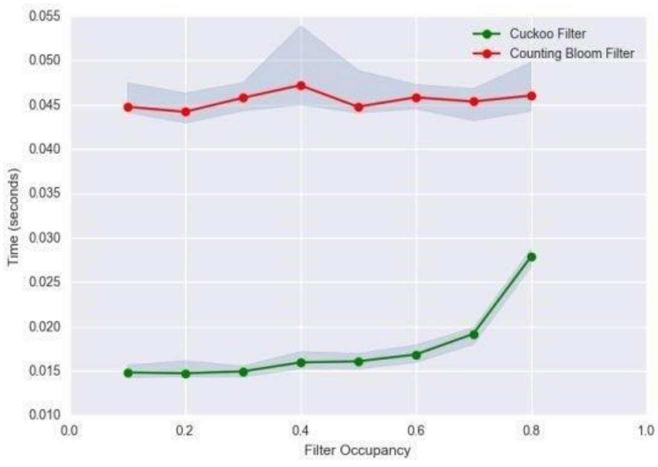


Figure 3.3: Cuckoo filter and counting bloom filter occupancy comparison [24]

The more capacity or filter occupancy means it has the capability of keeping items as many as possible in the data structure. That means the accuracy rate of the data structure automatically increases as it has many items stored in the data structure.

In accordance with the above features, Cuckoo filter has been chosen as the probabilistic data structure to keep all the unspent transactions for a given tainted transaction.

The tainted bitcoin identification application has the following sub components.

- Backend application
    - Tainted tool implemented with BlockSci
- Frontend application
    - User Interface

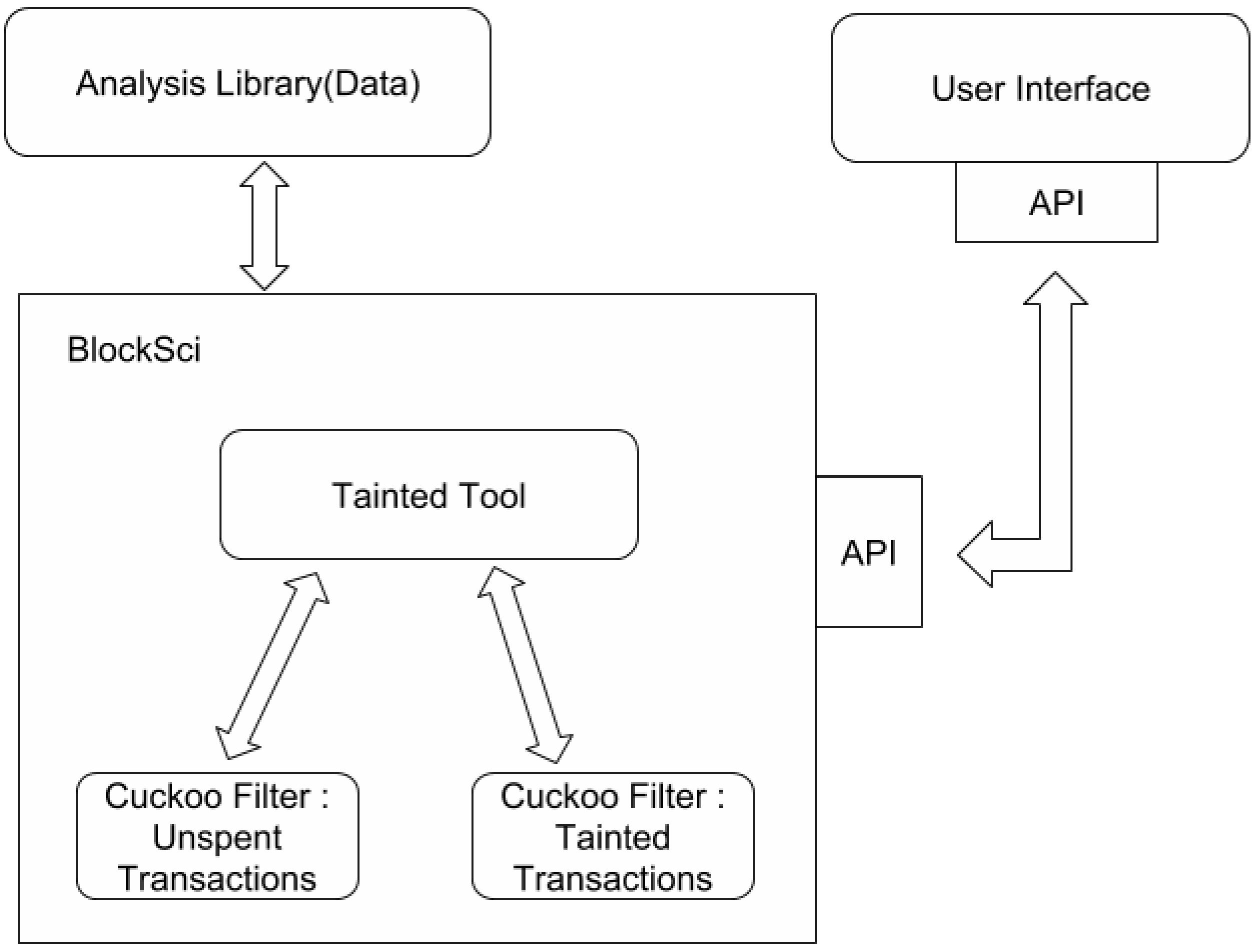

Analysis Library(Data)
User Interface
API
BlockSci
Tainted Tool
API
Cuckoo Filter : Unspent Transactions
Cuckoo Filter : Tainted Transactions

Figure 3.4: Architecture of tainted bitcoin identification application

Analysis library is the parsed data set which is taken from pure Blockchain after parsing through the BlockSci parser.

3.2.1 Backend Application

Backend application is designed to develop with the BlockSci implementation. Two Cuckoo filters are used for the backend application for the following purposes.

- To maintain the tainted transaction details
- To maintain the details of unspent transactions which are connected with tainted transactions

The two Cuckoo filters are used in the following manner.

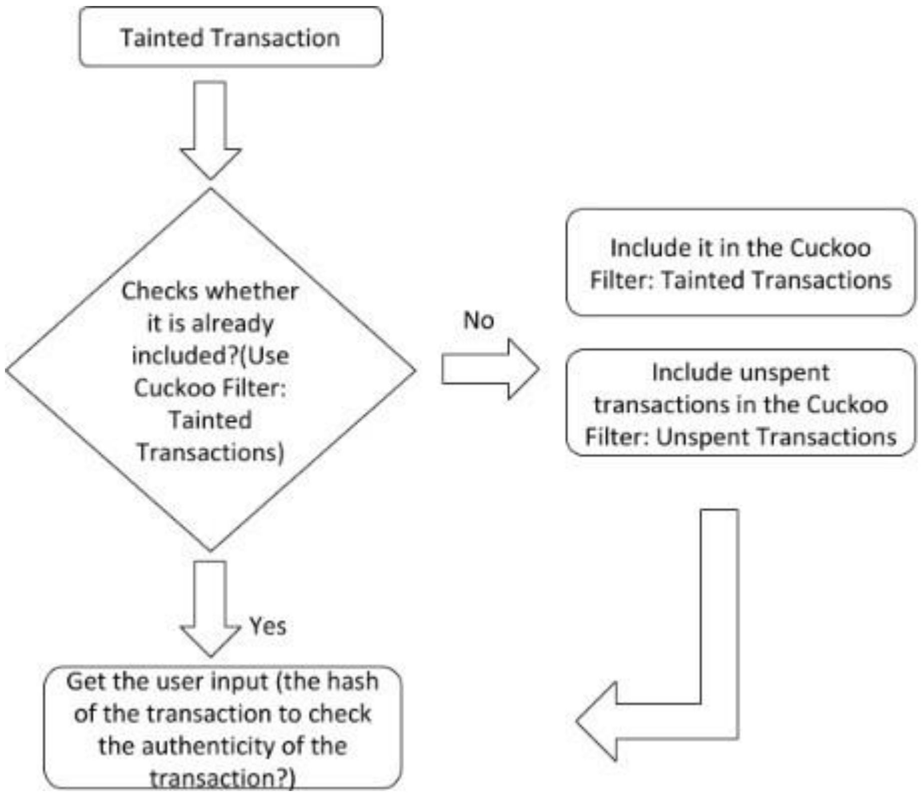

Tainted Transaction
Checks whether it is already included?(Use Cuckoo Filter: Tainted Transactions)
No
Include it in the Cuckoo Filter: Tainted Transactions
Include unspent transactions in the Cuckoo Filter: Unspent Transactions
Yes
Get the user input (the hash of the transaction to check the authenticity of the transaction?)

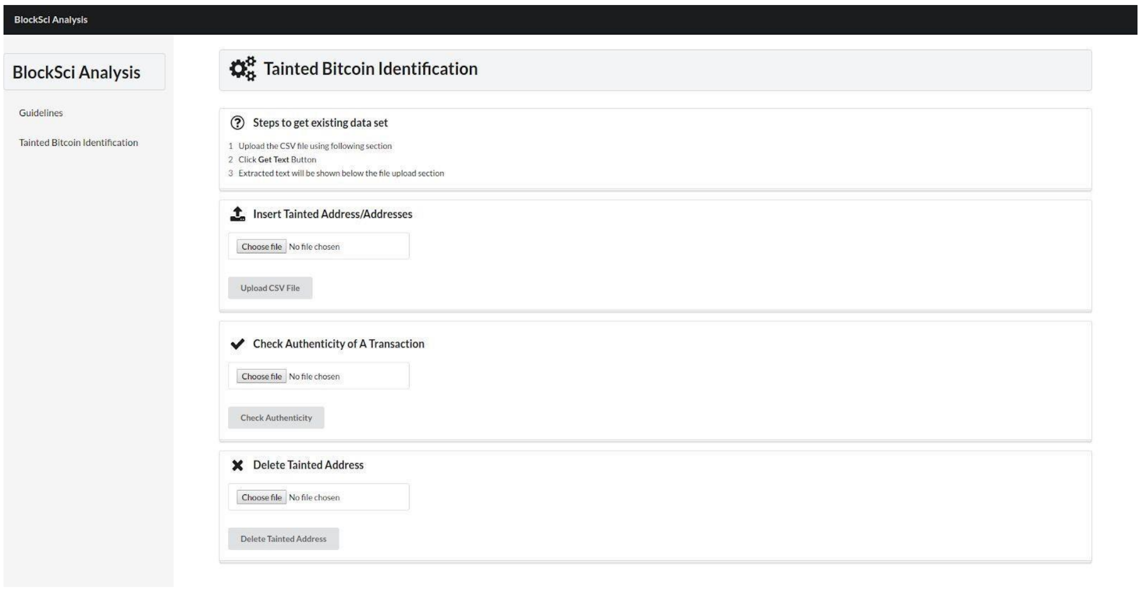
Figure 3.5: Two cuckoo filter usage for tainted bitcoin identification

When a tainted address is given to the backend application, first it checks if the given tainted transaction is already included in the data structure. For that it uses a separate Cuckoo filter. If its included then user does not need to include all the unspent transactions related to the tainted address as they have already been added to the data structure. Hence, user only needs to give the transaction to check its authenticity. If the tainted address does not contain in the data structure, then it needs to be added to the Cuckoo filter which contains tainted transactions, and then include all the unspent transactions to the other Cuckoo filter which is then can be used to check the authenticity of a given transaction. The second Cuckoo filter keeps the details of unspent transactions in a probabilistic manner.

3.2.2 Frontend Application

Users need to give the address of the tainted transaction. It can be given through a CSV file. If the user's tainted address is already in the Cuckoo filter then user can directly give the second input, which is the transaction that needs to check the authenticity. If it is a new tainted address then the backend application will take some time to prepare the data structure with all the unspent transactions related to the new tainted address.

The frontend application also has the option to delete a tainted address and the related unspent transactions from the data structure.


BlockSci Analysis

BlockSci Analysis

Guidelines

Tainted Bitcoin Identification

Tainted Bitcoin Identification

Steps to get existing data set

1 Upload the CSV file using following section
2 Click **Get Text** Button
3 Extracted text will be shown below the file upload section

Insert Tainted Address/Addresses

Choose file No file chosen

Upload CSV File

Check Authenticity of A Transaction

Choose file No file chosen

Check Authenticity

Delete Tainted Address

Choose file No file chosen

Delete Tainted Address

Figure 3.6: Front end view of the tainted bitcoin identification.

3.2 Cuckoo Filter Design

Cuckoo filters can be designed by considering various parameters.

- Number of bits per item
- Number of nests per bucket

These two factors directly affects load factor and the false positive rate of the Cuckoo filter. The 2-4 Cuckoo filter is used for the tainted bitcoin identification tool. It has two hash tables with four nests per bucket and sixteen bits have been used to represent a single item.

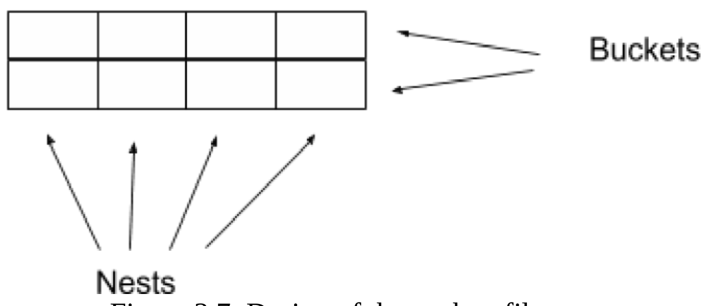


Figure 3.7: Design of the cuckoo filter

The Cuckoo is designed with the following operations.

- Add
  - This is used to include all the unspent transactions related to a tainted address in the data structure.

- Lookup
  - This is used to check a given transaction is already included in the data structure. (To check the authenticity of a transaction)

- Delete
  - This is used to remove transactions from the data structure which were already included in the data set.

3.3 Summary

This chapter provided a detailed description regarding tainted bitcoin identification tool and its components. The main focus is put on the Cuckoo filter design where the number of bits per item and number of nests per bucket is decided. Two Cuckoo filters will be utilized for the tainted bitcoin identification tool.

Chapter 4

Implementation

4.1 Introduction

This chapter elaborates the implementation details of the proposed solutions. Section 4.1 describes the software tools and techniques utilized for the implementation, section 4.2 represents the implementation assumptions, section 4.3 represents the implementation details of the Cuckoo filter, section 4.4 represents the implementation details of the tainted bitcoin identification tool.

4.2 Software Tools

There are many other tools and techniques used for this research for BlockSci setup and taint analysis.

4.2.1 Docker

Setting up BlockSci in the local machine is much of a complex task as it has many dependencies. Hence, BlockSci is setup using a Docker file as all the dependencies such as cmake are written in it.

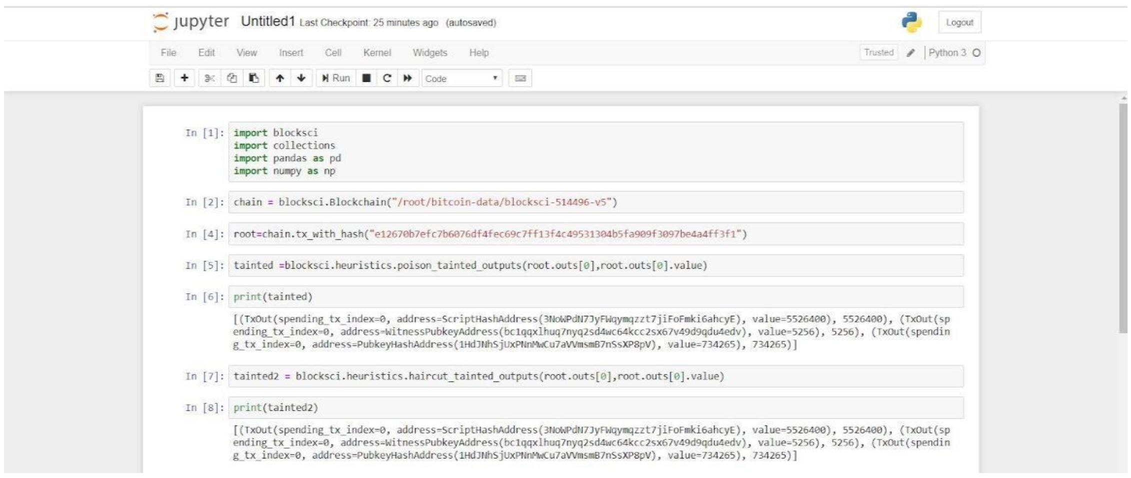
```
RUN  cd /root && \

wget https://cmake.org/files/v3.10/cmake-3.10.0.tar.gz && \    tar xzf
cmake-3.10.0.tar.gz && \    cd cmake-3.10.0/ && \    cmake . && \    make
&& \    make install
```

Docker volume mapping is used to map the local files and data folder into the docker machine and the implementation is done in the host machine. This is done when running the docker file.

```
sudo docker run -it --privileged -d --name $NAME  -v

/home/uthpala/Uthpala/Uthpala/DockerVolume/blocksci-
data:/root/bitcoin-dat a -v

/home/uthpala/Uthpala/Uthpala/DockerVolume/BlockSci/tools:/root/BlockS
ci/tools  -v

/home/uthpala/Uthpala/Uthpala/DockerVolume/BlockSci/src:/root/BlockSci
/src -p

$PORT:8888 $IMAGE_NAME
```

4.2.2 Jupyter

The Jupyter Notebook App is a server-client application that allows editing and running notebook documents via a web browser. BlockSci can be accessed from Jupyter and it is used to get the outputs of the unspent transactions. The same result can be accessed through a terminal, but the view given by Jupyter is much user friendly than the terminal window.



```
In [1]: import blocksci
        import collections
        import pandas as pd
        import numpy as np

In [2]: chain = blocksci.Blockchain("/root/bitcoin-data/blocksci-514496-v5")

In [4]: root=chain.tx_with_hash("e12670b7efc7b6076df4fec69c7ff13f4c49531304b5fa909f3097be4a4ff3f1")

In [5]: tainted =blocksci.heuristics.poison_tainted_outputs(root.outs[0],root.outs[0].value)

In [6]: print(tainted)

        [(TxOut(spending_tx_index=0, address=ScriptHashAddress(3NoWPdN7JyFWqymqzzt7jiFoFmki6ahcyE), value=5526400), 5526400), (TxOut(sp
        ending_tx_index=0, address=WitnessPubkeyAddress(bc1qqxlhuq7nyq2sd4wc64kcc2sx67v49d9qdu4edv), value=5256), 5256), (TxOut(spendin
        g_tx_index=0, address=PubkeyHashAddress(1HdJNhSjUxPNnMwCu7aVVmsmB7nSsXP8pV), value=734265), 734265)]

In [7]: tainted2 = blocksci.heuristics.haircut_tainted_outputs(root.outs[0],root.outs[0].value)

In [8]: print(tainted2)

        [(TxOut(spending_tx_index=0, address=ScriptHashAddress(3NoWPdN7JyFWqymqzzt7jiFoFmki6ahcyE), value=5526400), 5526400), (TxOut(sp
        ending_tx_index=0, address=WitnessPubkeyAddress(bc1qqxlhuq7nyq2sd4wc64kcc2sx67v49d9qdu4edv), value=5256), 5256), (TxOut(spendin
        g_tx_index=0, address=PubkeyHashAddress(1HdJNhSjUxPNnMwCu7aVVmsmB7nSsXP8pV), value=734265), 734265)]
```

Figure 4.1: Jupyter notebook

4.2.3 BlockSci Codebase

The tainted bitcoin identification is implemented on top on the BlockSci codebase. BlockSci has some tools implemented for different purposes such as for clustering, data parsing. Tainted bitcoin identification tool is also implemented under BlockSci tools. Heuristic methods such as Poison, Haircut are also implemented in BlockSci and these implementations are used in the tainted bitcoin identification tool.

4.2.4 Shell Scripting

After the implementation or even after any modification the code segment needs to be build and compiled in the Docker machine. Logging to Docker and do the compilation takes some effort and time. So the all the compilation steps in the Docker machine is written in a shell script. So that it is easy to do any changes or add new implementation without taking much effort.

```
#!/bin/bash

docker exec fypBlockScitemp sh -c  "cd /root/BlockSci/release  &&
CC=gcc-7  CXX=g++-7 cmake -DCMAKE_BUILD_TYPE=Release .. && make &&
make install"
```

Due to the above shell scripting compilation has become a one step process.

Additionally, shell scripting used for the running process of the taint tool. It includes several steps to follow inside the Docker machine. Due to the below shell script it could have been done as a single step process inside the host machine.

```
#!/bin/bash

docker exec fypBlockScitemp sh -c  "cd
/root/BlockSci/release/tools/taint  && ./blocksci_taint"
```

4.2 Implementation Assumptions

BlockSci is one of the main tool that currently exists for BlockSci analysis purposes. Even though it is comparatively good compared to other tools it has its own failures in the version used (v 0.5) for this research.

One of the main failure is that some hashes cannot be accurately parsed through the blockchain by the BlockSci parser. For an instance transactions which have script hashes cannot be parsed properly and public key hashes are parsed accurately. Hence, only the public key hashes are used for tainted bitcoin identification with the assumption that it will work accurately for other hashes as well.

Another failure is in the FIFO heuristic implementation. An infinite loop runs when tainted bitcoin identification tool tries to get all the unspent transactions using FIFO method. Hence, only Poison and Haircut heuristic methods are used to obtain the unspent transactions for a given tainted transaction.

4.3 Implementation Details: Cuckoo Filter

Cuckoo filter is based on a standard Cuckoo filter which is also called as 2-4 Cuckoo filter. In this Cuckoo filter there are four nests defined per bucket.

```
#define  CUCKOO NESTS PER BUCKET       4
```

Here 16 bits are used to represent a single item in the Cuckoo filter.

```
typedef  struct  {

    uint16_t   fingerprint;

 }_attribute_((packed)) cuckoo_nest_t ;
```

Here the size of the uint16_t is 16 bits.

The Cuckoo filter can be initialized as follows.

```
cuckoo_filter_new(&filter, size , max_items, (uint32_t) (time(NULL) &
0xffffffff));
```

The parameters are the filter, size of the filter, maximum number of items intended to include, and a special variable called seed which is used when calculating hash values.

4.4 Implementation Details: Tainted Bitcoin Identification Tool

This tool gets the outputs of a given tainted transaction and queries all the unspent transactions of a given address using a heuristic method and include them in the data structure which can be later used to identify the authenticity of a transaction.

The tainted transactions are taken from a file upload.

```
getline(myFile,input,'\n');

     if(myFile.good()!=0){

            -------------

     }
```

The tainted hash value is taken as the input and the transaction details related to the transaction is retrieved and assigned to the variable, root.

```
auto root = blocksci::Transaction(input,chain.getAccess());
```

Details of a particular transaction assigned to the variable root are listed below.

```
Tx( len(txins)=0, len(txouts)=1, size_bytes=135, block_height=2000, tx_index=2030)
```

According to the above details, transactions with outputs and transactions which do not have outputs can be clearly identified.

The next step of tainted bitcoin identification is to check whether a particular tainted transaction includes any outputs. If it does not have any outputs then no need of using the tainted function because there is no unspent transactions exist. Another important fact is only the public key hashes are used as discussed in the implementation assumption.

```
if (root.outputCount() > 0) {
    auto count = root.outputCount();
        for (auto i = 0; i < count; i++) {
  if ((root.outputs()[i].getType() ==
blocksci::AddressType::PUBKEYHASH) ){
          ---------
          ---------
       }
   }
}
```

After that a heuristic method is used to obtain all the unspent transactions. Poison heuristic method is used as follows in the implementation. Two parameters are the hash of the output and the value of the output.

```
auto tainted =
blocksci::heuristics::getPoisonTainted(root.outputs()[i],
root.outputs()[i].getValue());
```

While taint function gives the outputs of unspent transactions, the details of the outputs are stored in the Cuckoo filter. The unique hash value of each transaction is chosen to keep inside the Cuckoo filter. There are three parameters to be given.

- The filter (Place where data is stored)
- Data item
- Size of the data item

```
cuckoo_filter_add(filter,const_cast<char*>(pubkeyHash.c_str()),34);
```

After building the data structure, authenticity of a given transaction is checked against the transactions in the Cuckoo filter. Three parameters should be given.

- The filter
- Data item which is used to check the authenticity
- Size of the data item

If it’s included output is given as tainted.

```
cuckoo_filter_contains
(filter,const_cast<char*>("1LRkrnymve4Gm7eLeK3dMBzrwcDwJ9sY9L"), 34);
```

In an instance where a particular item needs to be deleted, it can be done as follows. For the deletion the following details should be given.

- The filter
- Transaction to be deleted
- Size of the transaction

```
 cuckoo_filter_remove(filter,const_cast<char*>(pubkeyHash.c_str()),
34);
```

4.4 Summary

In this chapter, the software tools utilized to implement the proposed solution was elaborated followed by the important functionalities of the proposed solution. The main implementation details of the Cuckoo filter such as filter implementation with necessary parameters are given and the most important functionalities of the Cuckoo filter are also stated. Finally the implementations of the tainted bitcoin identification are also stated how it is combined with the heuristic implementation of BlockSci. The implementation details of the evaluation model will be described in Chapter 5.

Chapter 5

Results and Evaluation

5.1 Introduction

This chapter elaborates how results are evaluated and the success level of the proposed solution. In Section 5.1 evaluation model is stated with respect to evaluation parameters used for tainted bitcoin identification. The next chapter includes tools and techniques utilized for the evaluation. The final section includes a detailed description of results and an explanation concerning the evaluation metrics.

5.2 Evaluation Model

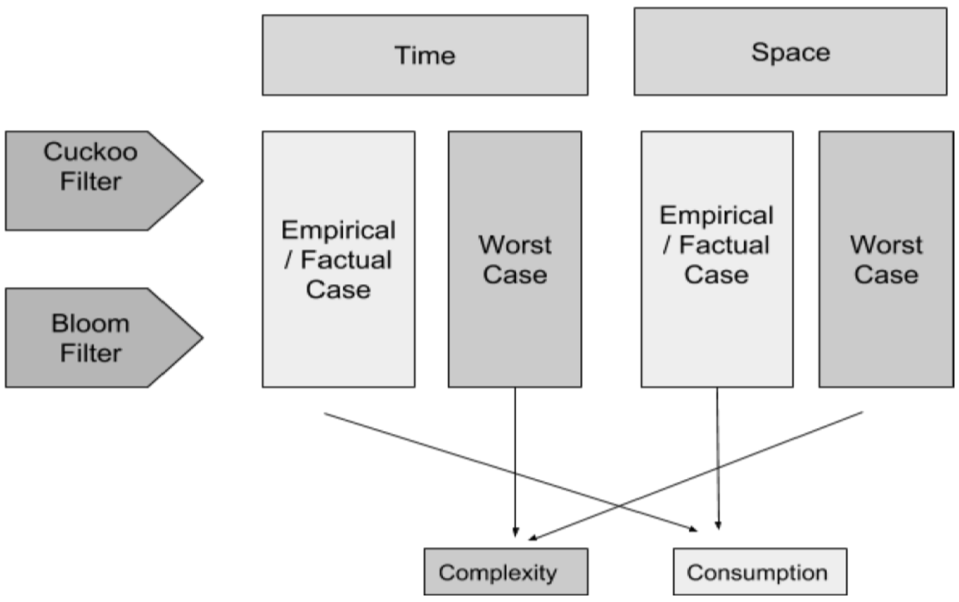


Figure 5.1: Evaluation model

The evaluation of the proposed solution is conducted with respect to time and space metrics. Worst case is theoretically evaluated by considering time and space complexities while factual case is practically evaluated by considering time and space consumptions. For the evaluation Bloom filter is also considered other than Cuckoo filter for results comparison.

The performance of the Cuckoo filter is also evaluated as follows.

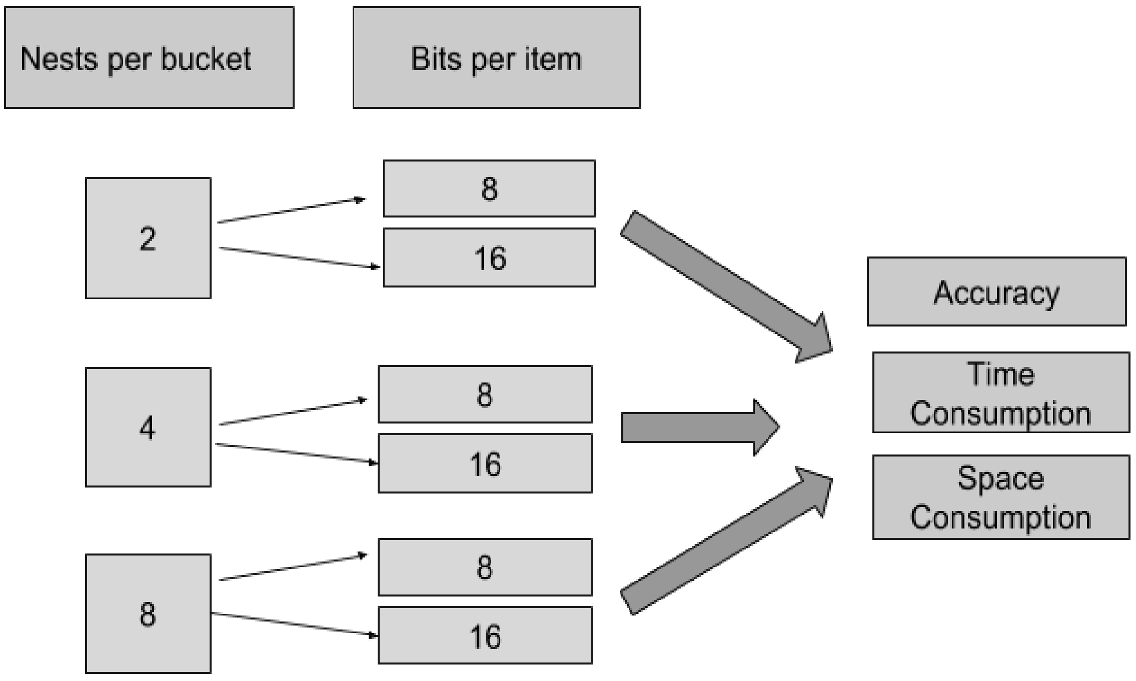


Figure 5.2: Evaluation model of the cuckoo filter

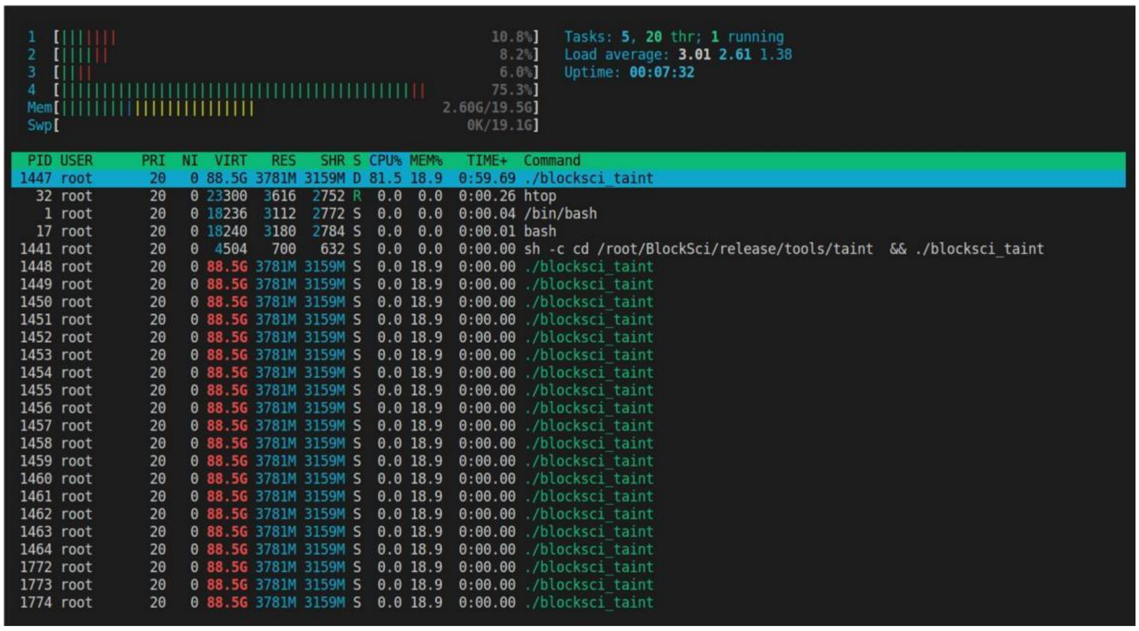
The accuracy of the Cuckoo filter depends on nests per bucket and bits per item. The evaluation will be conducted with respect to time and space consumption in terms of the mentioned six cases.

5.3 Evaluation Tools and Techniques

The below code segment is used when capturing time consumption for the various operations in tainted bitcoin identification.

```
auto start = high_resolution_clock::now();

auto stop = high_resolution_clock::now();

auto duration = duration_cast<microseconds>(stop - start);
```

For space consumption, Htop tool is used where it represents the total memory requested by each process through the VIRT (Virtual Memory) tab.



```
  1  [|||||||                                            10.8%]   Tasks: 5, 20 thr; 1 running
  2  [||||||                                              8.2%]   Load average: 3.01 2.61 1.38
  3  [||||                                                6.0%]   Uptime: 00:07:32
  4  [||||||||||||||||||||||||||||||||||||||||||||||     75.3%]
  Mem[||||||||||||||||||||||||                     2.60G/19.5G]
  Swp[                                                0K/19.1G]

  PID USER      PRI  NI  VIRT   RES   SHR S CPU% MEM%   TIME+  Command
 1447 root       20   0 88.5G 3781M 3159M D 81.5 18.9  0:59.69 ./blocksci_taint
   32 root       20   0 23300  3616  2752 R  0.0  0.0  0:00.26 htop
    1 root       20   0 18236  3112  2772 S  0.0  0.0  0:00.04 /bin/bash
   17 root       20   0 18240  3180  2784 S  0.0  0.0  0:00.01 bash
 1441 root       20   0  4504   700   632 S  0.0  0.0  0:00.00 sh -c cd /root/BlockSci/release/tools/taint  && ./blocksci_taint
 1448 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1449 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1450 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1451 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1452 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1453 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1454 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1455 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1456 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1457 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1458 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1459 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1460 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1461 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1462 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1463 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1464 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1772 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1773 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
 1774 root       20   0 88.5G 3781M 3159M S  0.0 18.9  0:00.00 ./blocksci_taint
```

Figure 5.3: Output format of the Htop tool

5.4 Results

Initially time complexity of the solution is evaluated which captures the worst case instances with respect to time. The next section includes time consumption of the tainted bitcoin identification tool. Subsequently space metric is considered with respect to space complexity and space consumption. Final evaluation is on the performance of the Cuckoo filter with reference to its implementation parameters.

When obtaining unspent transactions in both data insertion and deletion operations, Poison method is used as the heuristic method. Poison is the most complex method whereas Haircut and FIFO are less complex and definitely consume lesser time and space compared to Poison method. Hence the suitability of the solution is evaluated against the most complex method whereby the other methods will definitely fit for the solution if Poison method does.

5.4.1 Time Complexity of the Solution

The tainted bitcoin identification tool is the main tool used to identify the authenticity of a given transaction. The time complexity of the solution is analyzed with respect to data addition into the data structure, deletion and the availability check of a particular transaction (membership queries) by utilizing both Cuckoo filter and Bloom filter data structures.

5.4.1.1 Time Complexity of Data Addition

Time complexity is obtained by subdividing the implementation of the taint tool.

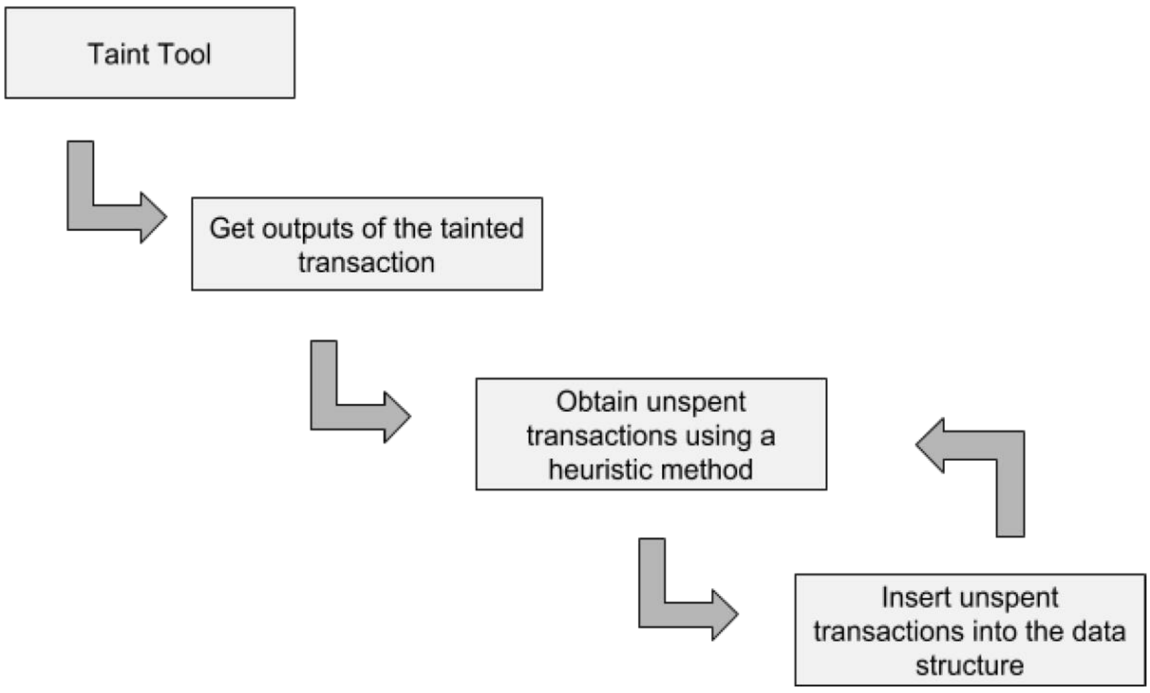

Taint Tool
Get outputs of the tainted transaction
Obtain unspent transactions using a heuristic method
Insert unspent transactions into the data structure

Figure 5.4: Data insertion process of tainted bitcoin identification

The pseudo code of the above diagram is as follows.

```
Tainted Bitcoin Identification(tainted transaction){
get outputs of the tainted transaction
    for i=0 to number of outputs{
         if i is unspent -> add to the data structure

        while output count not equals to zero{

            apply a  heuristic method to obtain unspent transactions
(hash value of i , value of i){

                 if transaction is unspent -> add to the data
structure }

        }

     }

  }
```

Get Outputs of the Tainted Transaction

The number of output transactions for a particular tainted transaction is obtained. The unspent tainted transactions are queried by going through the obtained outputs.

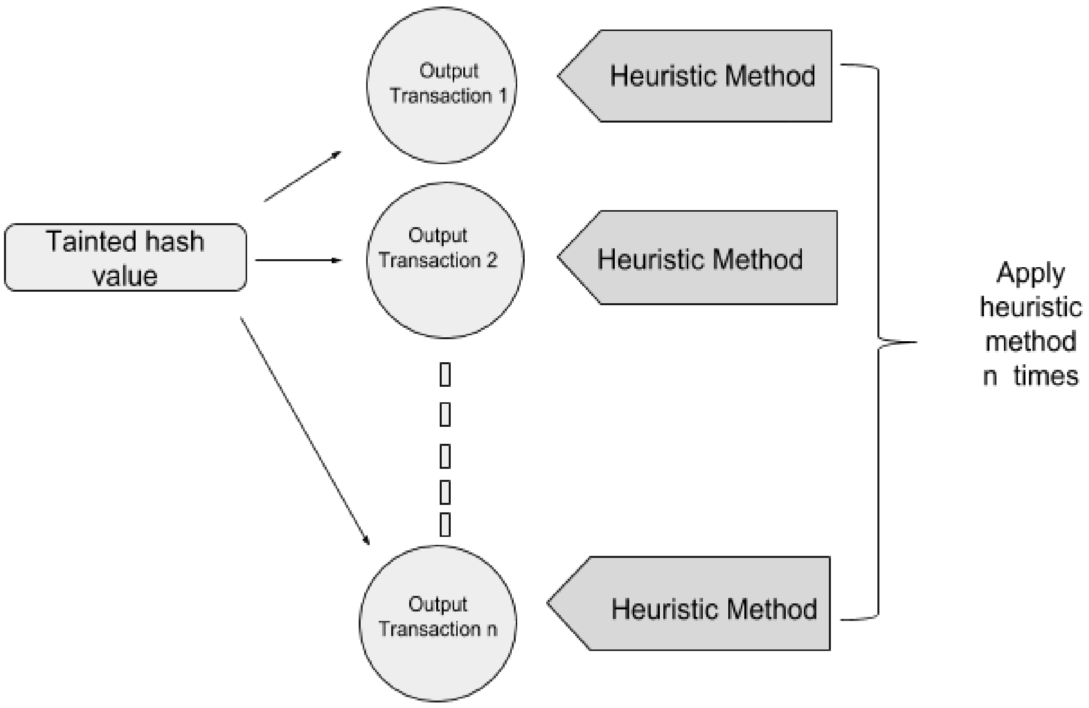


Figure 5.5: Applying heuristic method for the outputs of a tainted transaction

The time complexity of the above implementation depends on the number of output transactions. The time complexity of first sub part is O (n) when the number of outputs is equal to n. Hence this function consumes a linear time.

Obtain unspent transactions using a heuristic method

The tainted bitcoin identification function runs through all the outputs given and query again through all these outputs if those outputs are unspent. This querying process will cease until a particular transaction does not contain any outputs.

The pseudo code of the above recursive function is stated below:

```
getOutputs (param1, param2){
while(output count NOT equal to zero){
      --------------------------------------
      for 1 to output count{
            getOutputs (param1, param2)
         }
      }
}
```

Here param1 and param2 are hash of the output and the value of the output.

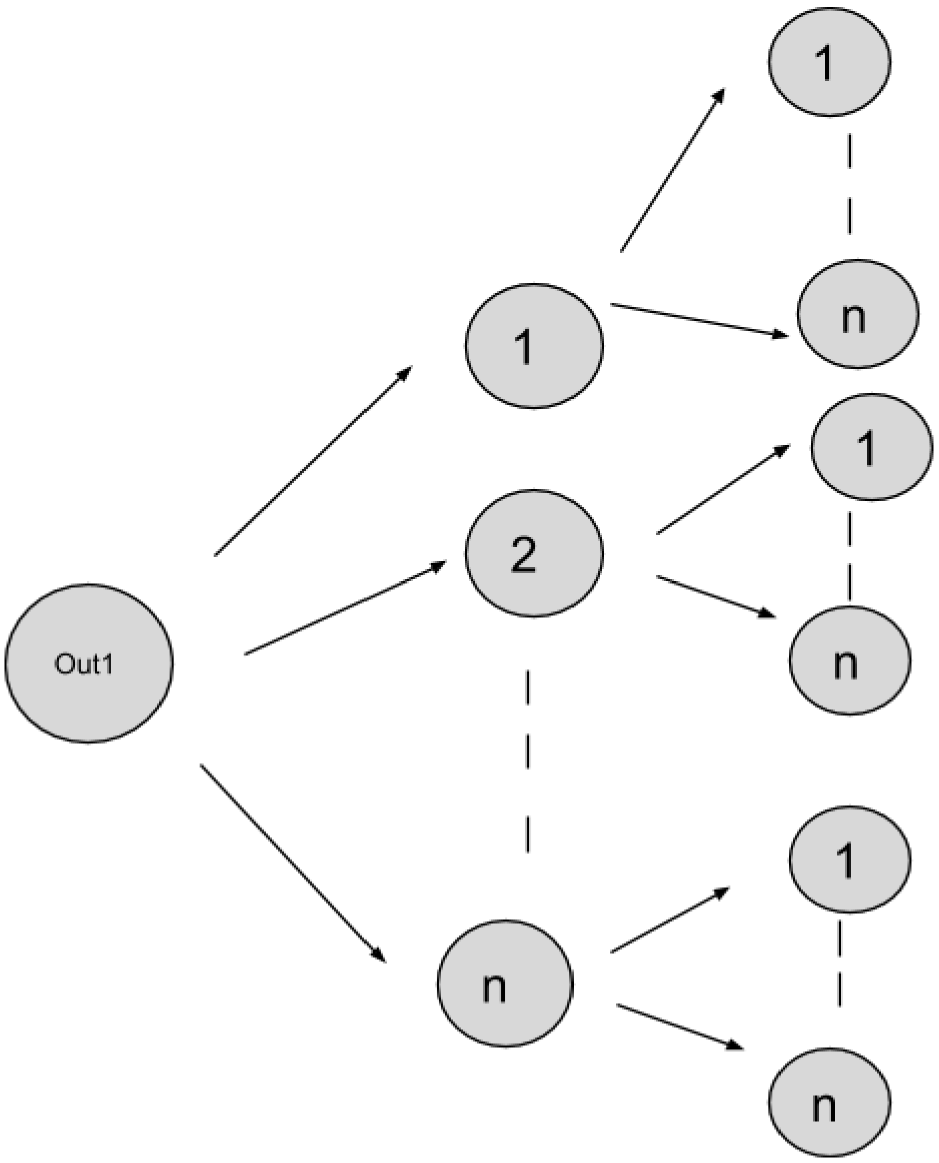


Figure 5.6: Querying unspent transactions with the assumption that each transaction has n number of outputs (worst case)

For an instance one of the output out of 9 outputs of a tainted transaction is taken as per in the above diagram. It is assumed that the branching factor of each transaction is n which the worst case scenario is. Thus, output 1 has n number of outputs and each of these outputs have n number of outputs. This querying process ends up when a particular transaction does not contain any outputs.

Hence, the time complexity can be calculated as $O\ (\underbrace{n\ {*}n\ {*}n{*}...........{*}n}_{\text{n times}})$

Therefore, for the heuristic method, it consumes O (n^n), an exponential time complexity.

Time Complexity of Data Insertion: Bloom Filter

The insertion time depends on the number of hash functions defined for a single input as the number of insertions equal to the number of hash functions. So the time complexity of this insertion part is O (k) where k is the number of hash functions defined.

Time Complexity of Data Insertion: Cuckoo Filter

Insertion of an item into Cuckoo Filter is a bit complex compared to Bloom Filter. Here there are two main functionalities related to item addition. Before adding any item into the data structure, first it checks whether this item already contains in the respective locations. In order to find out the location two hash functions are used which it gives bucket locations of the data structure. Then by looping through the number of nests per bucket it checks whether the particular item is already included or not.

If the item is not already included in both locations (two possible locations from two hash functions) then location of the bucket with respect to the item is gained using hash functions. The item is added to one of the vacant nests in the bucket where the bucket location is given by the hash functions. The addition also done to the correct nest by looping through the number of nests per bucket.

Hence the complexity is equal to number of nests per bucket O (4), which is a constant time complexity.

The overall complexity of item insertion in Bloom Filter and Cuckoo Filter are as follows.

Bloom Filter: $O(n^* n^n * k)$

$O(n^n)$

Cuckoo Filter: $O(n * n^n * 4)$

$O(n^n)$

As per the above calculation, it is apparent that Cuckoo filters have a lesser time complexity compared to Bloom filters with respect to item addition.

5.4.1.2 Time Complexity for Membership Queries

The most favorable feature in probabilistic data structures is that they performs a less time complexity when it is comes to membership queries. In Bloom filters the queried item is checked by going through the defined number of hash functions. So its running time depends on the number of hash functions.

In Cuckoo filters the bucket location is given by two hash functions and it checks for a particular data item in the defined bucket by looping through the number of nests defined per bucket. So it has a constant time complexity.

Bloom filter: O (n)   Cuckoo filter: O (4)

According to above information it is clear that Cuckoo filters have a better performance compared to Bloom filters. This special characteristic of Cuckoo filter is the main reason for its choice for tainted bitcoin identification.

5.4.1.3 Time Complexity of Data Deletion

Additionally, Cuckoo filters have another unique feature where it stand out from Bloom filter data structure. Cuckoo filters support for data deletion without causing any performance or accuracy issues. But Bloom filters have serious issues when it comes to deletions so it is a popular myth that Bloom filters do not support for data deletion.

Assume a use case where a particular transaction which was previously considered as tainted was later identified as a pure/legal transaction, then all the unspent transactions associated with the tainted transaction which were previously added to the Cuckoo filter need to be removed. So this can be easily done without compromising its accuracy nor its performance with Cuckoo filters.

Data deletion has a similar process as data insertion.

- Get outputs of the transaction which was previously considered as tainted
- Apply a heuristic method to obtain unspent transactions
- Delete unspent transactions from the data structure

Time complexity for the first two processes are similar as data insertion whereby it can depicted as follows.

$$O\ (n * n^n) \sim O\ (n^n)$$

When deleting an item from the Cuckoo filter, after getting the bucket location from two hash functions it check for the correct nest by looping through the number of nests per bucket. Hence the complexity is O (4).

The overall time complexity of data deletion in Cuckoo filter can be represented as follows.

Cuckoo Filter: $O\ (n * n^n * 4)$

$$O\ (n^n)$$

With the above mentioned facts, it is apparent that Cuckoo filters have a better performance compared to Bloom filters in tainted bitcoin identification.

5.4.2 Time Consumption of the Solution

5.4.2.1 Time Consumption for Data Addition

The time consumption (in microseconds) is captured with respect to data addition in both data structures. Here various number of unspent transactions have been used and for each tainted transaction and a new data structure is created for analysis purposes.

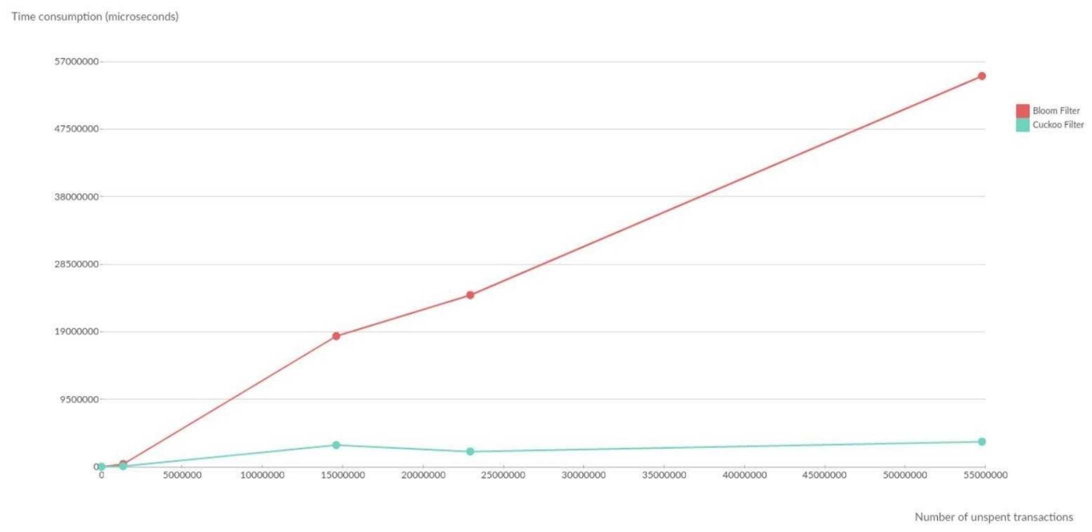


Figure 5.7: Time consumption for data addition in bloom filters and cuckoo filters

Based on the above diagram, it is clear that Cuckoo filters have a drastic time reduction compared to Bloom filters.

Even though theoretically time complexity of data insertion is exponential, practically the number of unspent transactions are limited due to the following factors

- All transactions do not have output transactions
- If outputs exist, some have already been spent

The below diagram represents a tainted transaction from year 2018 and how its tree chart visualization.

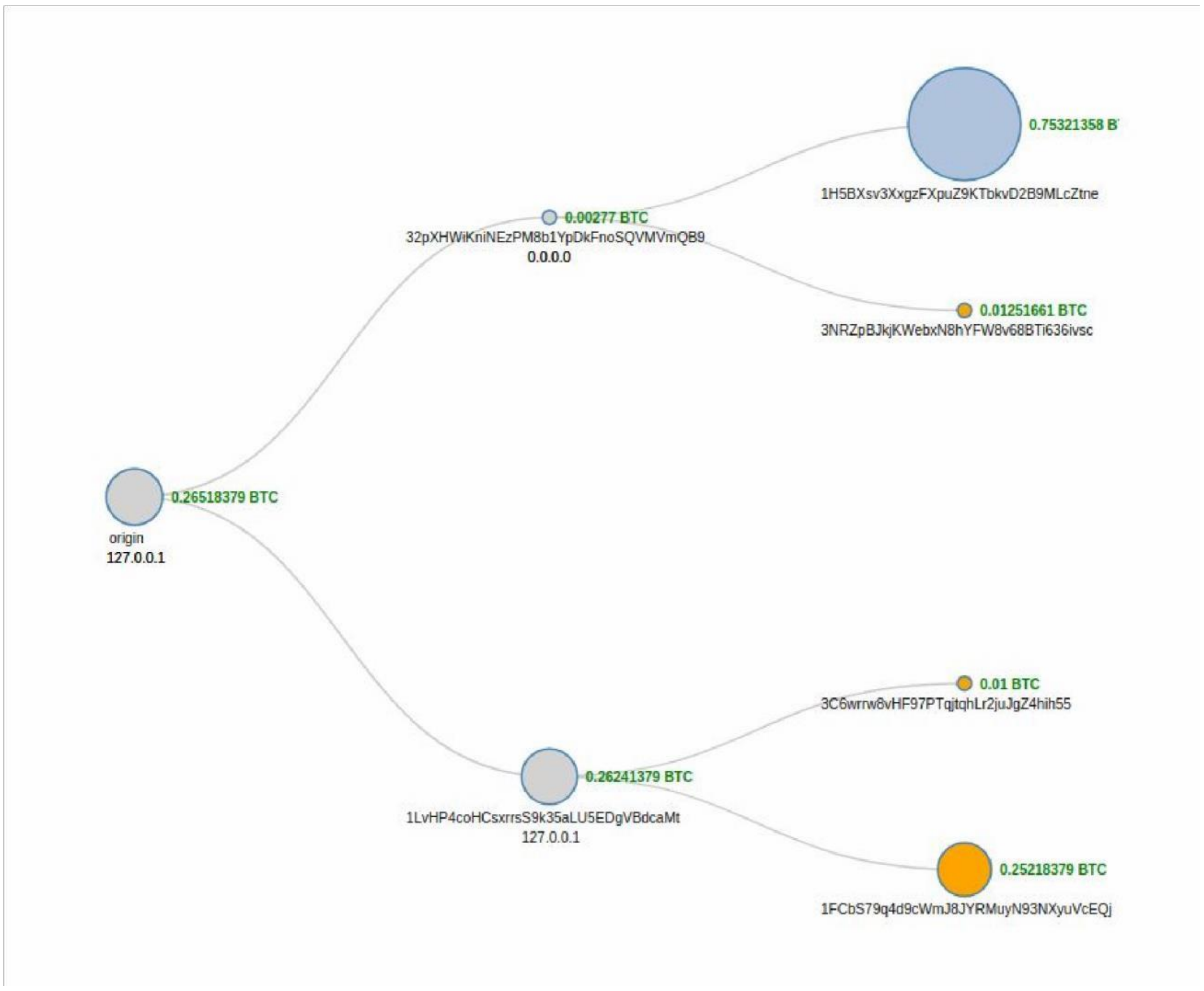

0.75321358 B
1H5BXsv3XxgzFXpuZ9KTbkvD2B9MLcZtne
0.00277 BTC
32pXHWiKniNEzPM8b1YpDkFnoSQVMVmQB9
0.0.0.0
0.01251661 BTC
3NRZpBJkjKWebxN8hYFW8v68BTi636ivsc
0.26518379 BTC
origin
127.0.0.1
0.01 BTC
3C6wrrw8vHF97PTqjtqhLr2juJgZ4hih55
0.26241379 BTC
1LvHP4coHCsxrrsS9k35aLU5EDgVBdcaMt
127.0.0.1
0.25218379 BTC
1FCbS79q4d9cWmJ8JYRMuyN93NXyuVcEQj

Figure 5.8: Tree chart visualization

In the above diagram, yellow circle represents spent transactions while blue represents unspent transactions. So only the transactions represent in blue color circles are queried and added to the data structure. Hence, this limits the number of transactions queried and added to the data structure.

5.4.2.2 Time Consumption for Membership Queries

Time consumption is captured using few transactions which have been included into the data structure prior to the existence check. Outputs obtained are represented in the below table.

Table 5.1: Time consumption for membership queries in bloom filters and cuckoo filters

	Time Consumption (microseconds)				
Cuckoo filter	0	0	1	0	0
Bloom filter	1	1	14	1	2

As per the above results, Cuckoo filters consume less time for membership queries compared to Bloom filters.

5.4.2.3 Time Consumption for Data Deletion

Since only, Cuckoo filter supports for data deletion time consumption is captured with the number of unspent transactions against its time consumption in microseconds.

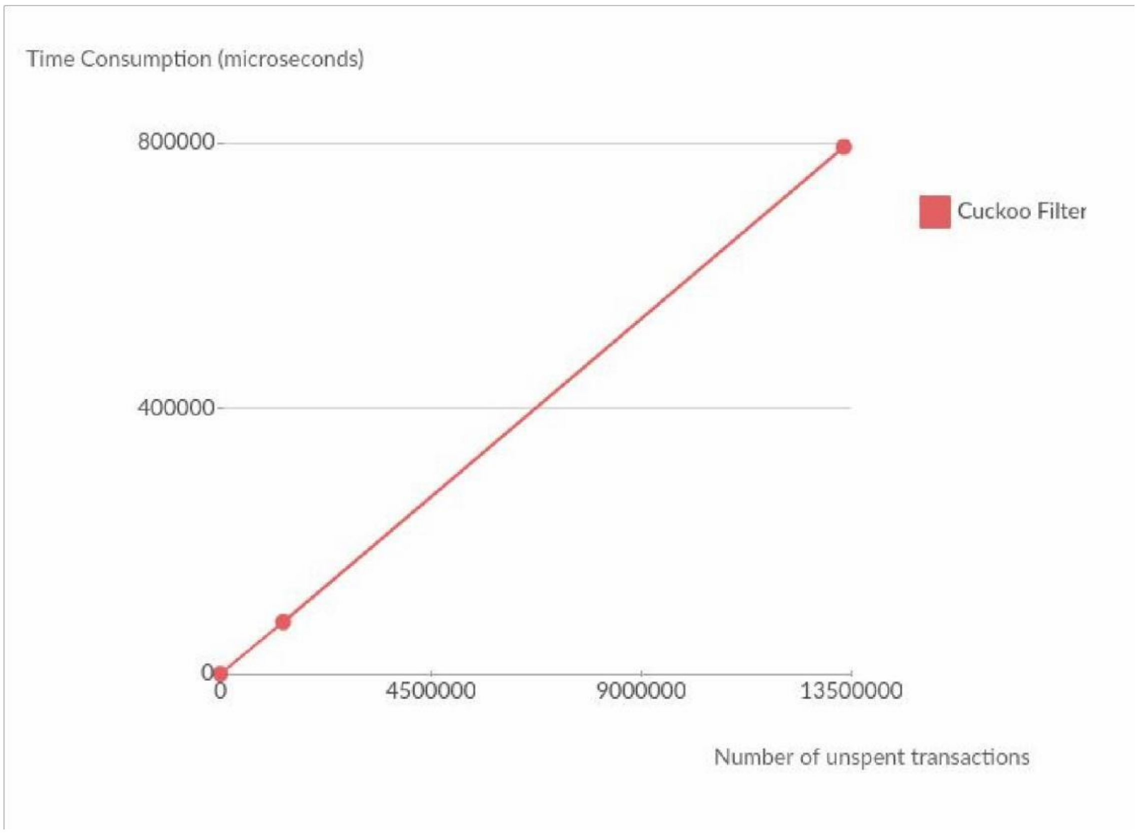


Figure 5.9: Time consumption for data deletion in cuckoo filters

5.4.3 Space complexity of the Solution

Space complexity is the amount of memory used by the tainted bitcoin identification function during its execution.

5.4.3.1 Space Complexity of Data Insertion

The space complexity of data insertion will be captured as the initial step of space complexity analysis. For that it will be analyzed under following parameters.

- ❖ Space complexity of the data structure
  - ➢ Bloom Filter
  - ➢ Cuckoo Filter
- ❖ Space complexity of obtaining unspent transactions using heuristic methods
- ❖ Space complexity for data addition

Space complexity of the data structure: Cuckoo Filter

The space complexity of the Cuckoo filter depends on bucket count and cuckoo nests per bucket.

Filter size = bucket count x nests per bucket

In the standard implementation cuckoo nests per bucket is defined as 4 and bucket count is calculated by using values of maximum key count and cuckoo nests per bucket. Here maximum key count means the maximum number of items that is supposed to add into the Cuckoo filter.

Bucket count = f (max key count (m) / cuckoo nests per bucket (4))

The bucket count is calculated in a separate function which is depicted here as f. It is clear that the bucket count is solely depend on maximum key count which is defined here as m. So the time complexity of the Cuckoo filter is O (m).

Space Complexity of the Data Structure: Bloom Filter

In Bloom filter space complexity directly depends on the size of the filter which is given at the time of the filter initialization. So it is clear that the space complexity depends on the size defined (m) at the time of filter initialization. Hence space complexity can be defined as O (m).

Space Complexity of Obtaining Unspent Transactions Using Heuristic Methods

When obtaining unspent transactions, the heuristic implementation behaves as a recursive function as follows. Here the worst case scenario is considered where each transaction has n number of output transactions. (Branching factor is n)

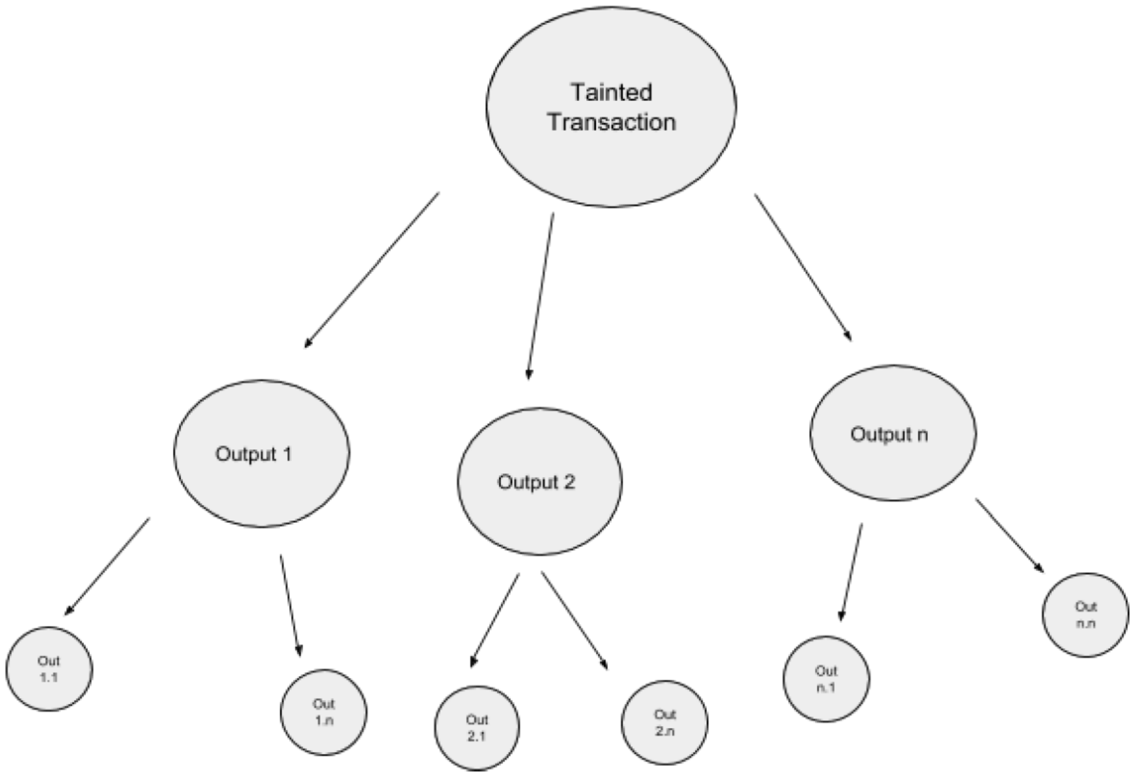


Figure 5.10: Representation of a tainted transaction having n number of output transaction (worst case)

The querying process will cease if a particular transaction does not contain any outputs. For n number of transactions, there exist $n^n + n$ number of recursions. So the space complexity of this algorithm for n number of transactions can be defined as follows.

$$O\ (n^n + n)$$
$$O\ (n^n)$$

Space Complexity of data addition: Cuckoo Filter

When adding items into the Cuckoo filter three extra variables are needed other than the input parameters. These parameters are needed for fingerprint, value of the hash 1, value of the hash 2. So the space complexity is O (3).

Space Complexity of data addition: Bloom Filter

Bloom filter requires an extra space up to the number of hash functions. So the space complexity of data insertion can be defined as O (k) where k is the number of hash functions.

The overall space complexity can be counted by summing up the complexities of the above parameters.

Cuckoo Filter: $O\ (\ (n^n\ *\ 3) + m) \sim O\ (n^n\ )$ Bloom Filter: $O((n^n * k\ ) + m\ ) \sim O\ (n^n\ )$

According to the above calculation, it is apparent that Cuckoo filter is much efficient in terms of space complexity. It is assumed that the number of hash functions should be greater than k in order to obtain reasonable efficiency.

5.4.3.2 Space Complexity of Membership Queries

When evaluating the space complexity, the following parameters need to be concerned.

- ❖ Space complexity of the data structure
    - ➢ Bloom Filter
    - ➢ Cuckoo Filter
- ❖ Space complexity for member query

Cuckoo filter consumes three extra variables namely fingerprint, value for hash 1, value for hash2. Thus, the space complexity of the Cuckoo filter can be defined as O (3) while Bloom filters' space complexity is O (k) where k is the number of hash functions.

The overall complexity for membership queries can be depicted as follows.

Cuckoo Filter: O (m +3), m is the maximum items in the data structure

Bloom Filter : O (m + k), m is the size of the Bloom filter

As per the above evaluation, it is clear that Cuckoo filters have a lesser space complexity for membership queries.

5.4.3.3 Space Complexity of Data Deletion

Since Cuckoo filters only supports for data deletion, complexity analysis is done only for Cuckoo filters. In order to evaluate the space complexity of delete operation complexity of the following parameters need to be evaluated.

- ❖ Space complexity of the data structure
    - ➢ Bloom Filter
    - ➢ Cuckoo Filter
- ❖ Space complexity of obtaining unspent transactions using heuristic methods
- ❖ Space complexity for data deletion

As per the above operations, data deletion also needs three extra variables other than the input values. Hence the complexity can be defined as O (3).

Overall space complexity of delete operation can be depicted as follows.

$$\text{Cuckoo Filter: } O((n^n * 3) + m)) \sim O(n^n)$$

5.4.4 Space Consumption of the Solution

Space consumption is captured by considering only the VIRT memory which captures the total amount of memory requested by processes. This amount of memory may not consume all the time during its execution but it represents the maximum memory required.

5.4.4.1 Space Consumption for Data Addition

The space consumption is captured using both data structures for a tainted transaction which has one of the most number of output transactions using the memory capturing tool, Htop. The space consumption is captured for five tainted transactions.

Table 5.2: Memory consumption for data addition in bloom filter and cuckoo filter

	Total Memory Consumption (VIRT Memory)				
Bloom Filter	88.7Gb	89.5Gb	91.1Gb	89.8Gb	88.8Gb
Cuckoo Filter	88.3Gb	89.4Gb	90.1Gb	89.4Gb	88.1Gb

5.4.4.2 Space Consumption for Membership Queries

The existence check is done for five unspent transactions and the results obtained are represented below.

Table 5.3: Memory consumption for membership queries in bloom filter and cuckoo filter

	Total Memory Consumption (VIRT Memory)				
Bloom Filter	88.7Gb	89.4Gb	89.2Gb	90.1Gb	90.2Gb
Cuckoo Filter	88.3Gb	89.4Gb	88.4Gb	89.8Gb	90.1Gb

5.4.4.3 Space Consumption for Data Deletion

Table 5.4: Memory consumption for data deletion in cuckoo filter

	Total Memory Consumption (VIRT Memory)				
Cuckoo Filter	88.3Gb	88.4Gb	89.3Gb	89.4Gb	90.1Gb

According to the space consumption results, it is apparent that there does not exist any clear space efficiency in Cuckoo filters compared to Bloom filters.

5.4.5 Cuckoo Filter Evaluation

The prior observations were based on a 2-4 Cuckoo filter which means it has two hash tables with four nests per bucket and 16 bits were used to represent a single item. The false positive rate depends on load factor and number of bits used for an item. Load factor varies as per the following table.

Table 5.5: Load factor which respect to bucket size in cuckoo filter

Bucket Size (Nests per bucket)	Load Factor
1	50%
2	84%
4	95%
8	98%

When considering the standard Cuckoo filter (2-4 Cuckoo filter) with a 95% load factor,

$$\text{Bits per item} = ((\log_2 (1/FPR) + 2) / \text{Load Factor})$$

According to the above equation,

False positive rate is equal to 0.0001

False positive rate is equal to 0.0001.

As the next step of the evaluation, performance of the Cuckoo filter is captured under following conditions.

Table 5.6: Five implementation of cuckoo filter (5 cases)

Cases	Bits per Item	Load Factor
1	16	98%
2	8	98%
3	8	95%
4	16	84%
5	8	84%

Time consumption and space consumption are captured with all the above cases and with the 2-4 standard Cuckoo filter (A single item is represented by 16 bits).

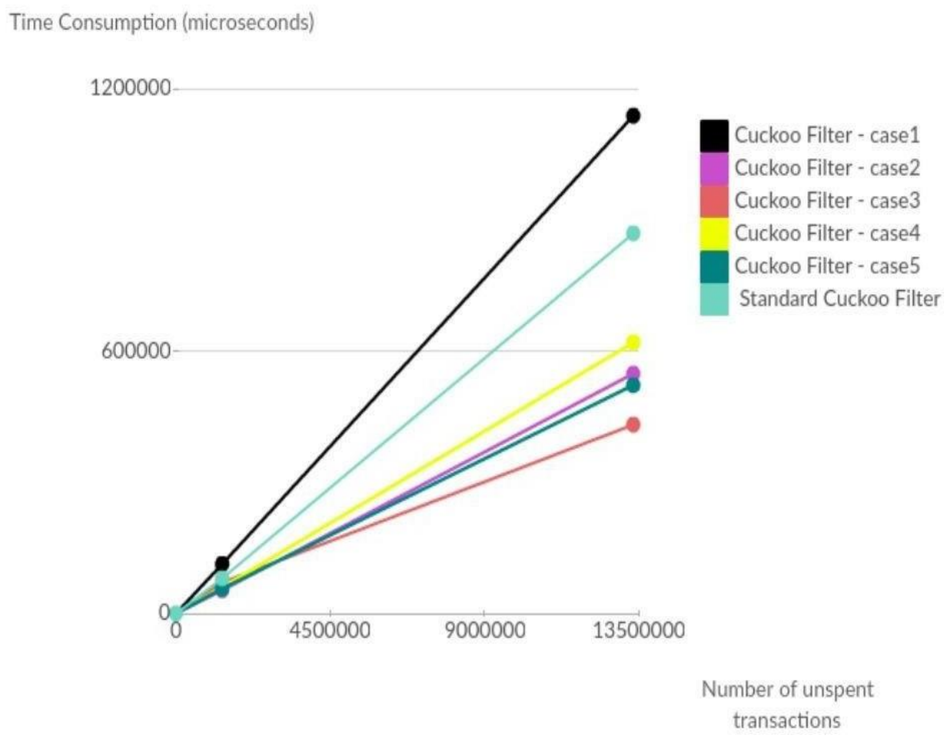


Figure 5.11: Time consumption of five cases and standard cuckoo filter

As per the above graph, it is clear that Cuckoo filters which represent an item with 8 bits have a lesser time consumption compared to other Cuckoo filters. The Cuckoo filter with 95% load factor which means 4 nests per bucket has the least time consumption.

Space consumption is also captured using the tool Htop for all the above cases. A transaction with the largest number of unspent transactions was taken as the tainted address.

Table 5.7: Total memory consumption of five cases and standard cuckoo filter

Cases	Total Memory Consumption
1	89.4Gb
2	88.9Gb
3	88.7Gb
4	89.4Gb

5	88.9Gb
Standard Cuckoo Filter	89.2Gb

In accordance with the above results, case 2, case 3 and case 5 Cuckoo filters have a lesser space consumption compared to other Cuckoo filters.

The false positive rate is calculated for the above cases.

Table 5.8: False positive rate of five cases and standard cuckoo filter

Cases	Bits Per Item	Load Factor	False Positive Rate
1	16	98% (8 nests per bucket / 2-8 Cuckoo Filter)	0.000076
2	8	98% (8 nests per bucket / 2-8 Cuckoo Filter)	0.017
3	8	95% (4 nests per bucket / 2-4 Cuckoo Filter)	0.021
4	16	84% (2 nests per bucket / 2-2 Cuckoo Filter)	0.00036
5	8	84% (2 nests per bucket / 2-2 Cuckoo Filter)	0.038
Standard Cuckoo Filter	16	95% (4 nests per bucket / 2-4 Cuckoo Filter)	0.00011

Cuckoo filters with lowest false positive rate has the highest time consumption when adding unspent transactions into it. (Case 1) According to the above results, it is clear that time /space consumption and false positive rate has an inverse relationship. It is possible to define a Cuckoo filter with the highest accuracy but compromising its performance. So the 2-4 Cuckoo filter is designed with a considerable accuracy rate and performance.

5.5 Summary

This chapter elaborated a detailed evaluation mainly considering time and space metrics with respect to its performance. For the evaluation Bloom filter is also is used other than Cuckoo filter to compare the results. Time and space complexity is analyzed theoretically as the worst case scenario in tainted bitcoin identification while factual case results are obtained through time and space consumption. Towards the end of the evaluation, performance of the Cuckoo filter is analyzed and how it varies with respect to false positive rate by changing the number of items representing a single item and load factor. However, it is observed that there is an inverse relationship between false positive rate and the performance of the Cuckoo filter.

Chapter 6

Conclusion

6.1 Introduction

This chapter includes a review of the research aims and objectives, research problem, limitations of the current work and implications for further research.

6.2 Conclusions about Research Questions (Aims/ Objectives)

The main aim of this research was to identify the possibility of using probabilistic data structures for an efficient tainted bitcoin identification. With that aim the main research objectives were,

- Identify a suitable probabilistic data structure
- Identify the necessary parameters when initializing the data structure
- Implement a tainted bitcoin identification tool
- Evaluate the performance of the data structure and compare the results

According to the literature review, Cuckoo filters have a better performance compared to other probabilistic data structures. Hence, they are utilized to make the tainted bitcoin identification efficient. However, Bloom filter is also considered for the evaluation purposes to compare results against with the Cuckoo filter.

According to the evaluation it is apparent that probabilistic data structures can be effectively used for tainted bitcoin identification. Among other probabilistic data structures, Cuckoo filters performed better mostly when to comes to time consumption. When considering space consumption, the results were almost like same for both Cuckoo filters and Bloom filters.

In theoretical analysis, Cuckoo filters performed better with regards to time and space complexities. The performance of the Cuckoo filter varied based on load factor and number of bits used to represent a single item. According to results obtained, it is apparent that 2-4 Cuckoo filter has a better accuracy rate(lesser false positive rate) across its time and space consumption.

It can be concluded that probabilistic data structures can be utilized for an efficient tainted bitcoin identification specifically Cuckoo filters.

6.3 Conclusions about Research Problem

The research problem stated was the high space and time consumption in tainted bitcoin identification and the absence of a mechanism to identify the authenticity of a given transaction. Thus, the usage of a probabilistic data structure was considered for an efficient tainted bitcoin identification.

Due to the usage of the probabilistic data structure, an additional time is consumed when adding unspent transactions into the data structure other than obtaining unspent transactions from a heuristic method. Hence, additional time and space are consumed when building the data structure. But after that, this data structure can be used as a tool to provide the authenticity of a given transaction within a few microseconds as it has a constant time complexity theoretically.

Even though, there is a huge time consumption reduction in the solution, performance of the solution does not efficient via space consumption as expected. Space consumption after usage of the data structure is almost same when obtaining results using heuristic methods.

It can be concluded that the proposed solution was able to address the problem statement with a huge time consumption reduction while maintaining the space consumption.

6.4 Limitations

The proposed solution was based on BlockSci implementations such as BlockSci parser and BlockSci heuristic methods. Due to some problems in BlockSci parser and one of the heuristic method, the proposed solution was bounded by some limitations.

The proposed solution was implemented for transactions with Public key hashes as it does not work well for transactions with Script hashes. Additionally, unspent transactions were obtained only using Poison and Haircut heuristic methods as FIFO implementation was a bit problematic.

6.5 Implications for Further Research

The tainted bitcoin identification is done with limitations due to problematic factors in BlockSci. Thus, the tool can re-evaluated with transactions having different types of transactions. The performance of the current tainted bitcoin identification was bounded by the performance of the heuristic methods. Current heuristic methods are not efficient enough for analysis purposes. Thus, an efficient heuristic method can be explored.

References

[1] S. Nakamoto, “Bitcoin: A peer-to-peer electronic cash system,” 2008.

[2] R. Anderson, I. Shumailov, and M. Ahmed, “Making Bitcoin Legal,” in *Security Protocols Workshop*, 2018.

[3] M. M{\"o}ser, R. B{\"o}hme, and D. Breuker, “Towards risk scoring of Bitcoin transactions,” in *International Conference on Financial Cryptography and Data Security*, Springer, 2014, pp. 16--32.

[4] R. Anderson, I. Shumailov, M. Ahmed, and A. Rietmann, “Bitcoin redux,” 2018.

[5] H. Kalodner, S. Goldfeder, A. Chator, M. M{\"o}ser, and A. Narayanan, “BlockSci: Design and applications of a blockchain analysis platform,” *ArXiv Prepr. ArXiv170902489*, 2017.

[6] “BlockSci.” citp/BlockSci,Github,2018.

[7] F. Reid and M. Harrigan, “An analysis of anonymity in the bitcoin system,” in *Security and privacy in social networks*, Springer, 2013, pp. 197--223.

[8] “Tainter checker list.” Bitcointalk.org.

[9] “Probabilistic Data structures: Bloom filter.” Hacker Noon.

[10] “Introduction to Probabilistic Data Structures.” DZone Big Data, DZone.com.

[11] B. Fan, D. G. Andersen, and M. Kaminsky, “Cuckoo filter: Better than bloom,”*USENIX Login*, vol. 38, no. 4, pp. 36--40}, 2013.

[12] B. Fan, D. G. Andersen, M. Kaminsky, and M. D. Mitzenmacher, “Cuckoo filter: Practically better than bloom,” in *Proceedings of the 10th ACM International on Conference on emerging Networking Experiments and Technologies*, ACM, 2014, pp. 75--88.

[13] F. Bonomi, M. Mitzenmacher, R. Panigrahy, S. Singh, and G. Varghese, “An improved construction for counting bloom filters,” in *European Symposium on Algorithms*, Springer, 2006, pp. 684--695.

[14] V. Gupta and F. Breitinger, “How cuckoo filter can improve existing approximate matching techniques,” in *International Conference on Digital Forensics and Cyber Crime*, Springer, 2015, pp. 39--52.

[15] M. Mitzenmacher, S. Pontarelli, and P. Reviriego, “Adaptive Cuckoo Filters,” in *2018 Proceedings of the Twentieth Workshop on Algorithm Engineering and Experiments (ALENEX)*, SIAM, 2018, pp. 36--47.

[16] D. Eppstein, “Cuckoo filter: Simplification and analysis,” *ArXiv Prepr. ArXiv160406067*, 2016.

[17] G. Cormode, “Count-min sketch},” in *Encyclopedia of Database System*, Springer, 2009, pp. 511--516.

[18] G. Cormode and S. Muthukrishnan, “An improved data stream summary: the count-min sketch and its applications,” *J. Algorithms*, vol. 55, no. 1, pp. 58--75, 2015.

[19] D. Zambre and A. Shah, “Analysis of bitcoin network dataset for fraud,” 2013.

[20] P. B. Gibbons and Y. Matias, “Synopsis data structures for massive data sets,” *Extern. Mem. Algorithms*, vol. 50, pp. 39--70, 1999.

[21] A. Broder and M. Mitzenmacher, “Network applications of bloom filters: A survey,” *Internet Math.*, vol. 1, no. 4, pp. 485--509, 2004.

[22] G. Cormode, M. Garofalakis, P. J. Haas, and C. Jermaine, “Synopses for massive data: Samples, histograms, wavelets, sketches,” *Found. Trendstextregistered Databases*, vol. 4, no. 1--3, pp. 1--294, 2011.

[23] “Probabilistic Filters by Example: Cuckoo Filter and Bloom Filters.” Bdupras.github.io.

[24] “Probabilistic Data structure Showdown: Cuckoo Filters vs Bloom Filters.” Blog.forwardlabs.com.