MoneroSci: Linkability and Traceability Analysis of Monero Blockchain

S.G.H. Liyanage

MoneroSci: Linkability and Traceability Analysis of Monero Blockchain

S.G.H. Liyanage

Index No: 14000784

Supervisor : Dr. T.N.K. De Zoysa

December 2018

Submitted in partial fulfillment of the requirements of the
B.Sc. (Hons) in Computer Science Final Year Project (SCS 4124)

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 organizations.

Candidate Name: Mr. S.G.H. Liyanage

....

Signature of Candidate   Date:

This is to certify that this dissertation is based on the work of

Mr. S.G.H. Liyanage

under my supervision. The thesis has been prepared according to the format stipulated and is of acceptable standard.

Principle Supervisor's Name: Dr. T.N.K. De Zoysa

....

Signature of Supervisor   Date:

Abstract

Cryptocurrencies are one of the most popular and valuable digital currency using as a medium for the financial transaction. There are several cryptocurrencies such as Bitcoin, Ethereum, Ripple, Litecoin, etc. But the main issue of those is the anonymity within a transaction. Considering Bitcoin, every transaction on the blockchain is broadcast publicly and visible for all entities in the network, but the owner of each wallet is unknown. Every transaction is bind with the owner's public address. If the owner's public address reveals the transaction's anonymity also reveals. This is the term called pseudo-anonymity.Monero is a one of a cryptocurrency found for the solution of the problem in anonymity. Monero has a set of inbuilt privacy features than Bitcoin. Monero developers have ensured this privacy feature by addressing the main two properties. Unlinkability and untraceability. But some researchers have found some issues of Monero unlinkability and untraceability guarantees by making some attacks for the Monero. But anyone hasn't verified or evaluate those attacks because for the Monero it has no optimized environment to analyze, explore or query the blockchain.

In this dissertation, has proposed an optimized environment to explore and analyze the Monero blockchain called MoneroSci which perform queries faster than existing tools and has verified the existing traceability attacks conducted by the past researchers. And has extended the attack to analyze the linkability guarantee in Monero. During the study, it has evaluated the performance of MoneroSci Parser and Analysis Library. MoneroSci loads the whole blockchain data in 15 seconds for the analysis. And it quantified the 87% of transactions are traceable based on the attacks.

Keywords - *Cryptocurrency, Monero, MoneroSci, Traceability, Linkability, Blockchain, Analysis-platform*

Preface

An optimized environment to explore, analyze and query for the Monero blockchain called MoneroSci has introduced in this dissertation. MoneroSci is special because this is the only tool that can be used as an analysis tool or forensic tool to the law enforcement duties for the Monero blockchain. On top of the MoneroSci, the dissertation has evaluated the existing traceability attacks conducted by some researchers. And also includes key findings of Monero linkability analysis using the existing research concepts, heuristics and own opinions of myself. This linkability analysis is special because no one has addressed this scope in Monero yet. MoneroSci includes three main models. MoneroSci parser, Core Blockchain Data and MoneroSci Analysis Library. The development of MoneroSci parser was solely my own work and has not been proposed in any other study related to blockchain analysis platforms. It developed as a multithreaded application which is operated function parallelly to gain better performance. Core Blockchain Data and MoneroSci Analysis Library are based on the concept of BlockSci which is existing blockchain analysis platform. To implement the traceability attacks in the existing research I have referred the algorithms researchers mentioned and used to build those attacks.

Acknowledgements

I would like to express my sincere gratitude to my research supervisor, Dr T.N.K. De Zoysa, senior lecturer of University of Colombo School of Computing and my research co-supervisor, Mr. T.G.A.S.M. De Silva, assistant lecturer of University of Colombo School of Computing for providing me continuous guidance and supervision throughout the research.

I would also like to extend my sincere gratitude to Dr D. N. Ranasinghe, Dr H.A. Caldera and Mr. Charitha Elvitigala for providing feedback on my research proposal and interim evaluation to improve my study. 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 beloved mother and father for always being my strength. And also dedicated to all my family members, to primary and secondary school teachers, to lecturers of University of Colombo School of Computing who has been an immense support to me throughout this journey of life. It is a great pleasure for me to acknowledge all the people who helped me tremendously to successfully complete my research.

Table of Contents

Declaration **i**

Abstract **ii**

Preface **iii**

Acknowledgements **iv**

Table of contents **vii**

List of figures **ix**

List of tables **x**

List of acronyms **xi**

1 Introduction **1**

1.1 Background to the Research . . . 2

1.2 Research Problem and Research Questions . . . 4

1.3 Research Aim and Objectives . . . 5

1.4 Justification for the Research . . . 5

1.5 Methodology . . . 7

1.6 Outline of the Dissertation . . . 9

1.7 Delimitations of Scope . . . 9

1.7.1 In-Scope . . . 9

1.7.2 Out-Scope . . . 10

1.8 Summary . . . 10

2 Literature Review **11**

2.1 Introduction . . . 11

2.2 CryptoNote 2.0 Protocol . . . 11

2.3 Related Works . . . 13

2.3.1 Monero untraceability guarantee 13
2.3.2 Blockchain Analysis Platforms 17
2.4 Summary 18

3 Design **20**
3.1 Introduction 20
3.2 Design of MoneroSci Analysis Tool 20
3.2.1 Importing Data 21
3.2.2 Parser 21
3.2.3 Core Blockchain Data 24
3.2.4 Analysis Library 24
3.3 Designs of Unlinkability and Untraceability
Attacks 26
3.3.1 Attack I: Leveraging Zero Mix-ins 26
3.3.2 Attack II: Leveraging Output Merging 27
3.3.3 Extend Attack II Design for Unlinkability 28
3.4 Summary 30

4 Implementation **31**
4.1 Introduction 31
4.2 Software Tools 31
4.3 Implementation Details - MoneroSci Tool 32
4.3.1 Parser 32
4.3.2 Core Blockchain Data 35
4.3.3 Analysis Library 39
4.4 Implementation Details - Traceability Attacks 43
4.4.1 Attack I: Leveraging Zero Mix-ins 43
4.4.2 Attack II: Leveraging Output Merging 45
4.4.3 Extend Attack II for Unlinkability 47
4.5 Summary 49

5 Results and Evaluation **50**
5.1 Introduction 50
5.2 Evaluation Model 50
5.2.1 Performance of MoneroSci Parser 50
5.2.2 Performance of MoneroSci Analysis Library 51
5.2.3 Monero Traceability and Linkability Guarantee 51
5.3 Results 52

5.3.1 Performance of MoneroSci Parser 52
5.3.2 Performance of MoneroSci Analysis Library 56
5.3.3 Monero Traceability with Mix-ins 59
5.3.4 Traceability Attack I: Leveraging Zero Mix-ins 60
5.3.5 Traceability Attack II: Leveraging Output Merging 62
5.3.6 Extend Attack II for Unlinkability 64
5.4 Summary 67

6 Conclusions 68
6.1 Introduction 68
6.2 Conclusions about research questions
(aims/objectives) 68
6.3 Conclusions about research problem 69
6.4 Limitations 70
6.5 Implications for further research 70

References 71

A MoneroSci Documentation 74

List of Figures

1.1 Proposed research methodology 7

2.1 Standard transaction structure [19] 12
2.2 Incoming transaction check [19] 13
2.3 Attack I: Cascade effect due to zero mix-ins [17]) 14
2.4 Attack II: Leveraging Output Merging [17] 15
2.5 Overview of BlockSci's architecture [13] 17

3.1 MoneroSci high-level architecture diagram 21
3.2 Overview of MoneroSci parser in multithreaded environment 22
3.3 Execution flow of MoneroSci parser 23
3.4 Overview of MoneroSci Analysis Library 25
3.5 Structure of the linked transaction graph 30

4.1 Structure of Core Blockchain Data 36

5.1 Output of the MoneroSci parser 52
5.2 Memory usage of MoneroSci Parser 54
5.3 CPU Utilization of MoneroSci Parser 55
5.4 Block parsing time of MoneroSci Parser 55
5.5 MoneroSci vs Onion Monero Blockchain Explorer block ranges access time 56
5.6 MoneroSci vs Onion Monero Blockchain Explorer cumulative block access time 56
5.7 Retrieving all block data by MoneroSci 57
5.8 Monero mix-ins with frequencies 59
5.9 Results of finding zero mixin percentage 61
5.10 Result of employing Attack II on non-RingCTs 63
5.11 Attack II: Overall observed percentage of TP, FP and UP 63
5.12 Degree distribution of non-RingCT linked graph 64
5.13 Degree distribution of RingCT linked graph 65

5.14 Number of nodes and edges in RingCT linked graph over the time. . 66
5.15 Degree distribution in RingCT linked graph over the time. 67

A.1 Homepage of the MoneroSci documentation 74
A.2 Configuration details of MoneroSci parser in the documentation . . 75
A.3 Configuration details of MoneroSci Analysis Library in the documentation 75
A.4 Blockchain class of MoneroSci Analysis Library in the documentation 76
A.5 Block class of MoneroSci Analysis Library in the documentation . . 76
A.6 Tx class of MoneroSci Analysis Library in the documentation . . . 77
A.7 Input class of MoneroSci Analysis Library in the documentation . . 77
A.8 Output class of MoneroSci Analysis Library in the documentation . 78
A.9 Ring class of MoneroSci Analysis Library in the documentation . . 78

List of Tables

2.1 Related works on traceability analysis of Monero blockchain 19

5.1 Parsed Blockchain Data Counts 53
5.2 Performance evaluation of MoneroSci Parser 53
5.3 Monero mix-ins with cumulative frequencies 60
5.4 Attack I zero mixin cascade effect in percentage 61

List of Acronyms

BTC Bitcoin.

CBD Core Blockchain Data.

FP False Positive.

GB Gega Byte.

I2P Invisible Internet Project.

JSON JavaScript Object Notation.

MAL MoneroSci Analysis Library.

RingCT Ring Confidential Transaction.

TP True Positive.

UP Unknown Positive.

USD United State Dollars.

vCPU virtual central processing unit.

VM virtual machine.

XMR Monero.

Chapter 1

Introduction

Nowadays cryptocurrency is one of the most popular digital asset designed to play the role of exchanges. It uses strong cryptographical techniques such as encryption and decryption to secure the financial transaction. The major benefit of using cryptocurrency is that there is no trusted third-party involvement to control you within the transaction. When considering the cryptocurrencies Bitcoin, Ethereum, Ripple, Litecoin, and Monero are the most popular and trending cryptocurrencies in the digital market.

Monero is a leading privacy cryptocurrency using the digital market as well as dark web markets due to its privacy features than the other coins. Due to that reason, there were some cyber-attacks based on Monero payments. Here privacy is an most important topic in digital cryptocurrencies. It is not a secret that, everyone needs privacy because of neither organisations or individuals want to broadcast their private data and identities onto a public blockchain which can easily access by anyone without getting permission from them or without any restrictions. To address this issue Monero developers have added inbuilt privacy features for the Monero to support the private transactions. To ensure the inbuilt privacy features developers have concerned about two main properties. Unlinkability and Untraceability.

While Monero does the explaining and providing such privacy features in theoretically some researchers have focused on Monero privacy features in practically. Based on their observations there are some issues in unlinkability and untraceability properties in Monero. But the main problem of analyzing these unlinkability and untraceability properties is there is no optimized environment to query the Monero blockchain. Analyzing blockchain is not an easy task because blockchain includes a set of blocks, transactions, inputs, and outputs which are more than 60GB size storage. Therefore, for the Monero, it should need to have an optimized tool for the researchers or the interested parties to analyze the Monero blockchain to quantify these privacy properties and reduce the issues in Monero. As well as it can be used as a tool for law enforcement duties to track the attackers who doing malicious works with the Monero.

1.1 Background to the Research

Cryptocurrency was born out of the need for secure communication between payer and payee. There is no any third-party involvement for the transactions and no need of trust-based model for each transaction. Cryptocurrencies allow fast, inexpensive payments and from anyone to anywhere in the world. Nowadays there are more than 1500 different types of cryptocurrencies in the digital market and most of the coins are mainly used by people due to their features. Cryptocurrency term was got popular after Satoshi Nakamoto's Bitcoin white paper [1]. Because it has been a successful implementation of the concept of peer to peer electronic cash system and it has proposed a better solution to the double-spending problem in the digital currency. Thereafter, bitcoin caught the market capital with more than USD 300B. Now Bitcoin is the most popular decentralized cryptocurrency in the world. But the most critical flaw in bitcoin is its lack of privacy as evidence of several analysis in the past [2, 3, 4, 5].

Privacy and anonymity are very important facts in the uses of cryptocurrencies. When considering the bitcoin, every transaction on the blockchain is broadcast publicly and visible for all entities in the network, but the owner of each wallet is unknown. Every transaction's output is bind with the user's public electronic address. Tying these public addresses to real-world identities is now relatively easy, because bitcoin user has to cash out somewhere, and that usually involves linking bitcoin addresses to bank accounts. Because of that in bitcoin user's identity depends on their bitcoin transaction address. This is the reason of Bitcoin provides pseudo anonymity to their users [6, 7].

Privacy in a transaction is more important to make sure that cryptocurrency is fungible. Which means is all the coins are the same. With privacy, it's hiding the history of the coin. That is important for the scalability of the blockchain and every person doesn't want to use a currency that shows the history of all the transactions ever done.

In light of the privacy issues in Bitcoin, a new cryptocurrency called Monero (XMR) was launched on April 18th 2014. Monero is a privacy-centric cryptocurrency which allows users to make transactions without exposing the financial and private details to the public. Nowadays Monero is a popular, trending and leading privacy coin in the cryptocurrency market. Due to that, it has a market capitalization of USD 2.2B [8]. Monero Blockchain and transactions are working similarly as Bitcoin's peer to peer electronic cash network. But according to the Monero research lab papers, there are more additional features in Monero which guarantee the privacy in Monero more than Bitcoin.

In Monero, the main privacy ensuring methods is called "mix-ins". Monero has inbuilt mix-in mechanism. And also, Monero uses complex and new cryptographic methods like Ring Signatures, Stealth Addresses, Ring Confidential Transactions (RingCT) and KOVRI I2P protocol to ensure the privacy of their end users.

Ring signature is the method to make sure a transaction can't be tied back to a specific individual. First presented by D. Chaum and E. van Heyst [9]. This method uses other transaction's outputs as mix-ins to hide the actual input of the transaction. This is the method that Monero uses for untraceability feature [10, 11]. Stealth address is the method using for transactions unlinkability. In stealth addresses, transactions are not directly coming to the user's public address. It generates a one-time public address for each transaction. Ring confidential transaction is the way of hiding the transaction's amount. This feature was not obligatory at the beginning of introducing in January 2017, but after September 2017 it is obligatory for every Monero transaction. Therefore, Monero blockchain transactions can seperate into two main transactions called, RingCT-transactions and non-RingCT transactions. KOVRI I2P protocol provides the additional security layer in the network layer. It hides internet traffic such as passive network monitoring by encrypting and route through the I2P nodes. These are very strong privacy features in theoretically but practically Monero developer team have not proven these concepts are robust to any attack develop by malicious attackers or other data miners.

For other cryptocurrencies such as Bitcoin, Litecoin, Namecoin etc. there are tools to analyze the whole blockchain in quick. But Monero it doesn't. More details about the analysis tools are discussed in chapter two. Analysis of blockchain data is very important and useful for both scientific and commercial purposes. Blockchains are very large and growing quickly. Therefore, it must need to have an optimized environment which can be used to analyze the blockchain data to find hidden data patterns, find security and privacy threats, etc.

1.2 Research Problem and Research Questions

According to the Monero [12], they have provided strong privacy features to their end users to protect the user's privacy within the transactions. But, while Monero does the explaining and providing strong privacy features in theoretically, some researchers have focused on Monero untraceability and unlinkability guarantee in practically. From their research, they have observed some issues in unlinkability and untraceability guarantees of Monero. Therefore, the main problem of the Monero is that, does Monero really ensure unlinkability and untraceability privacy features for their end users?

Another problem for Monero is, there is no optimized data analysis platform yet to analyze the whole blockchain. When considering Bitcoin and some trending cryptocurrencies like Litecoin, Namecoin, and Zcash there is a tool for expressive analysis and explore the blockchain data called BlockSci [13]. In Monero, there are some tools to retrieve Monero data from the blockchain via JSON readers, but those tools do not perform well for analysis scenarios. The main limitation of the BlockSci is, it is not supported for Monero due to the different structure of the blockchain compared with the Bitcoin and others. Mainly, Monero doesn't follow the "one-input, one-output" paradigm. In other words, the transaction graph contains an additional type of node, called the mix-ins. Analysis of blockchain data is very important and useful for scientific researches and commercial applications to find the gaps and improve the performance further comparing to the exists.

Considering this research problem, the generated research questions are as follows:

- Can verify proposed traceability attacks conducted by Amrit Kumar et al. and Malte Möser et al. for Monero blockchain?
- Is there any optimized way to analyze the whole Monero blockchain?
- Develop a tool for exploring and analyzing linkability and traceability of Monero blockchain and transactions

1.3 Research Aim and Objectives

The main aim of the research is to create an optimized environment to explore and analyze the Monero blockchain called MoneroSci and quantify the unlinkability and untraceability guarantee in Monero on top of it.

The objectives of the research are as follows:

- Find an optimized way to explore and analyze the Monero blockchain.
- Deploy MoneroSci tool as a forensic tool for law enforcement duties.
- Propose new techniques to quantify the efficacy of untraceability and unlinkability in Monero transactions.
- Create a tool called MoneroSci for Monero blockchain to explore, analyze and trace the transactions' traceability and linkability based on currently existing attacks for Monero.
- Share research findings with the Monero development team and the general community.
- Publishing a research paper based on "MoneroSci: Linkability and Traceability Analysis of Monero Blockchain".

1.4 Justification for the Research

Monero is a leading privacy coin in the cryptocurrency market with growing market capitalization of USD 2.2 billion. Nowadays users are dropping Bitcoin due to their lack of privacy features and moving to new privacy coins like Monero due to their inbuilt privacy features.

But there are several attacks and criminal uses in Monero, because of its popularity and privacy. Some of Monero attacks are "Monero mining malware targeting Android users in the name of fake Google Play update", "Cryptocurrency mining botnet malware called Smominru", "Monero mining malware on Jenkins servers". To date, these attacks have reportedly managed to mine more than 10,000 Monero, or about USD 3 million. This is a considerable security threat and no one has addressed yet to trace these attacks in Monero transactions.

In 2017 there had been three widely public scenarios of criminal activity involving

Monero transactions. First, the popular darknet market called AlphaBay began accepting Monero deposits in July 2016. In July 2017, US law enforcement raided an AlphaBay server and seized 12,000 Monero (worth around $500,000) [14].

Second, the Shadow Brokers a notorious hacking group, from June 2017 onwards offered to accept Monero payments for subscription access to zero-day vulnerabilities and exploit tools. They have advised their hopeful subscribers to publish their email addresses in the Monero blockchain, this leading to these transactions being identified [15].

Third, WannaCry ransomware operators received Bitcoin ransomware payments, to a common address. Then they have exchanged Bitcoin for Monero using the Swiss exchange service ShapeShift due to the Bitcoin's privacy issues. The Swiss exchange subsequently announced their cooperation with US law enforcement and began blacklisting Bitcoin ransoms. However, $36,922 have already been exchanged for Monero [16].

If there is a way to link and trace the Monero transactions using an optimized environment, it is possible to reduce the motivation of doing attacks like these. So far, there is no any tool available to explore and analyze such crimes for Monero which can use for law enforcement duties and studies about Monero blockchain for researchers.

1.5 Methodology

The main goal and significance of this research is to create an optimized environment to explore and analyze the Monero blockchain called MoneroSci and use this tool as a forensic tool for law enforcement duties. Currently, there is no any tool for Monero to explore and analyze the whole blockchain data at the same time of increasing various types of cryptocurrency crimes. Figure 1.1 represents a high-level diagram of the proposed research methodology.

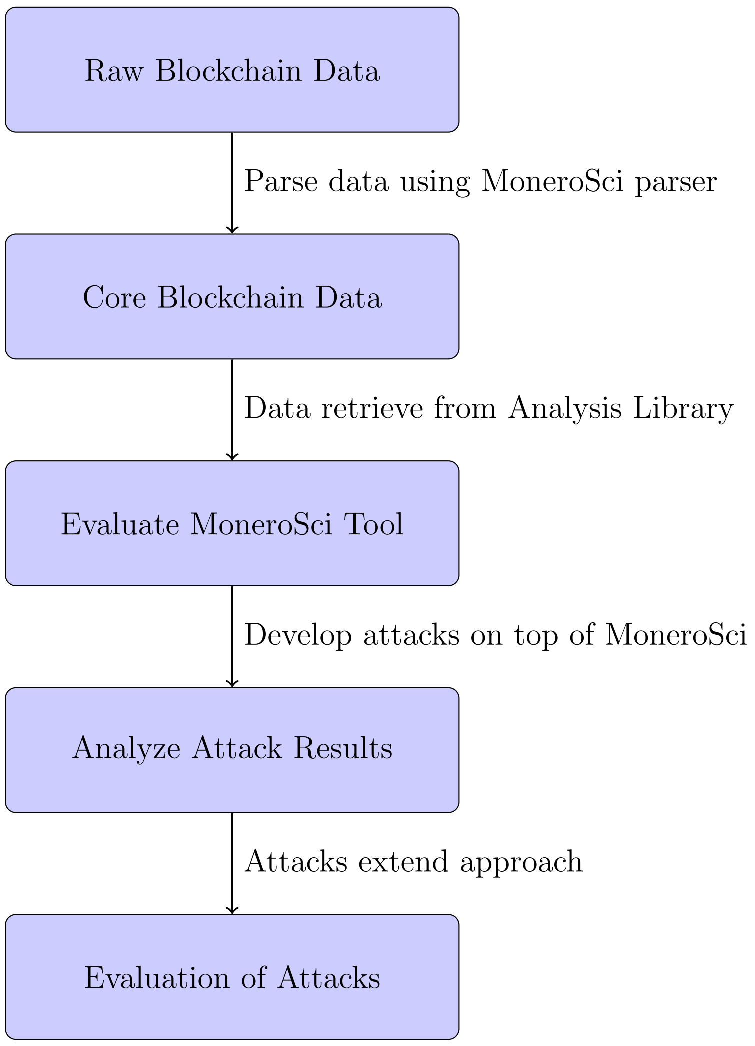


Figure 1.1: Proposed research methodology

During the research, public Monero blockchain will be used as the data set of the research. But it cannot directly use as the data set because of the on-disk format of blockchains is highly inefficient. As the purpose of this research to create an optimized environment, there should be a way to explore and analyze the on-disk formatted blockchain data in an efficient way.

To achieve this goal, the number of techniques will be used to optimize the speed of access to Monero blockchain data. First, replace hash pointers with IDs to shrink the data structure and optimize linkage. Reason for this is accessing these IDs are never performance critical in scientific analysis instead of using hashes. Second, deduplicate public address/hashes data. This will be achieved using a data structure called Bloom Filter. Bloom filter is a data structure designed to find, whether an element is present in a memory or not in memory-efficiently. Therefore, bloom filter can be used to optimize the searching mechanism. Third, link stealth addresses (outputs) to the ring members of input (mix-ins) that spend them in order.

In MoneroSci parsing is sequential. The blockchain must be processed sequentially because it maintains an index for every element in the blockchain. Such as for blocks, transactions, stealth-addresses, ring members and key images. The parser is required to transform the raw blockchain into the MoneroSci analysis data format (Core blockchain data). The parser will be developed as multi-threading application which similar to the producer-consumer scenario. Parsing raw blockchain data into MoneroSci analysis format is required only one time.

To analyze the transformed raw blockchain data (MoneroSci core blockchain data), MoneroSci exposes a python wrapper interface (MoneroSci Analysis Library). Which include several classes and methods for blocks, transactions, ring members and stealth addresses to retrieve data from blockchain in an object-oriented way. The analysis of Monero blockchain will be done in a statistical way to find how the Monero guarantees their unlinkability and untraceability features to their end users.

The evaluation of this methodology will be a quantitative approach as it focuses on to verify the existing attacks on top of MoneroSci which used to quantify the efficacy of Monero's untraceability guarantee developed by Amrit Kumar, Shruti Tople, Clément Fischer, Prateek Saxena from National University of Singapore [17] and Malte Möser*, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin [18] by using MoneroSci core blockchain data. It needs to implement a relevant algorithm for each existing attack on the MoneroSci exploring and analyzing tool.

MoneroSci analysis tool performance evaluation is focused on two ways. Performance of the MoneroSci Parser and performance of MoneroSci Analysis Library. Based on the results of untraceability attacks, new techniques for quantifying the efficacy of unlinkability in Monero transactions has considered. This part is experimental research of this research.

1.6 Outline of the Dissertation

The dissertation is structured as follows. Chapter two describes the theory based on the CryptoNote protocol and the related works which are observed the vulnerabilities in Monero blockchain. Chapter three elaborates the proposed research design and methodology. The potential way of addressing the research problems of this research is discussed in this chapter. Chapter four discusses the implementation details of the proposed research design and the methodology. Chapter five presents the results and evaluation model of the proposed approaches. The last chapter, chapter six discusses the conclusion of the thesis and outlines the future work which can use for any researcher who interests and willing to extend this research.

1.7 Delimitations of Scope

1.7.1 In-Scope

The following will fall within the scope of the project in its current study:

- MoneroSci (exploring and analyzing tool) is developed for all types of coins in CryptoNote cryptocurrency family.
- Up to April 6th 2018 (block height 1546000) public Monero blockchain data chosen as the data set.
- Only Monero blockchain analysis will be considered in the current study.
- MoneroSci parser will be proposed for Monero current block-height.
- Existing traceability attacks conducted by Amrit Kumar et al. and Malte Möser et al. will be considered for Monero blockchain.

1.7.2 Out-Scope

The following will not be covered under this project in its current study:

- Monero mining pool analysis will not be covered in the current phase.
- MoneroSci will not be tested for all types of coins in CryptoNote cryptocurrency family.

1.8 Summary

Monero is one of a popular cryptocurrency uses for private transactions. People need privacy because of neither organizations or individuals want to broadcast their private financial transactions to the outside world. Nowadays Monero is the most popular cryptocurrency for the malicious attackers due to its inbuilt privacy features than comparing other cryptocurrencies such as Bitcoin and Ethereum. Therefore, dark web markets like AlphaBay also now allowing the payments based on Monero. Still, there is no optimized environment to analyze such thing and explore the Monero blockchain efficient way. Considering the Monero inbuilt privacy features practically, some researchers have observed issues in unlinkability and untraceability guarantees. The research objectives mainly focus to develop optimized analyzing environment called MoneroSci and quantify the unlinkability and untraceability guarantee in Monero blockchain on top of it.

Chapter 2

Literature Review

2.1 Introduction

In this chapter, fundamental concepts of Monero transaction, related works on Monero untraceability features and related works on blockchain analysis platforms are provided. Section 2.2 focuses on the CryptoNote protocol which is Monero using as its protocol, Section 2.3.1 focuses on the related works on Monero untraceability features and Section 2.3.2 focuses on the related works on blockchain analysis platforms. In section 2.4 focuses on the overall summary of the literature review chapter.

2.2 CryptoNote 2.0 Protocol

Monero is a fork of the Bytecoin. Bytecoin is the first real-life implementation of CryptoNote. Currently, Monero is running on CryptoNote 2.0 protocol. According to the CryptoNote white paper [19], Monero is a solution to the main deficiencies of Bitcoin. In Bitcoin, the most critical flaw is its lack of privacy. When considering the blockchain technology, blockchain is a distributed database with the immutable dataset. That means is data in the blockchain cannot be updated or deleted and it only allows the data read and write. Due to that reason, the privacy issue is occurred from data reading from the blockchain. In whitepaper of CryptoNote [19], they have stated that how the Monero aims to address this privacy issue by considering two properties. Firstly, Untraceability: for each incoming transaction, all possible senders are equiprobable. Secondly, Unlinkability: for any two outgoing transactions, it is impossible to prove they were sent to the same person. An introduction to the Literature Review, this is the standard transaction sequence of CryptoNote [19] transactions.

1. Payer wants to send payment for the payee (know his Monero address), the first payer unpacks payee address and gets payee public key (A, B).

2. Payer generate random r value and compute a one-time public key P=Hs(rA)G + B.

3. Payer use P as a destination key for output and packs value R=rG somewhere into the transaction.

4. Payer sends the transaction.

5. Payee checks every passing transaction with his private key (a,b) and computes P' = Hs(aR)G + B. If payee verifies that transaction belongs to him if and only if P' = P.

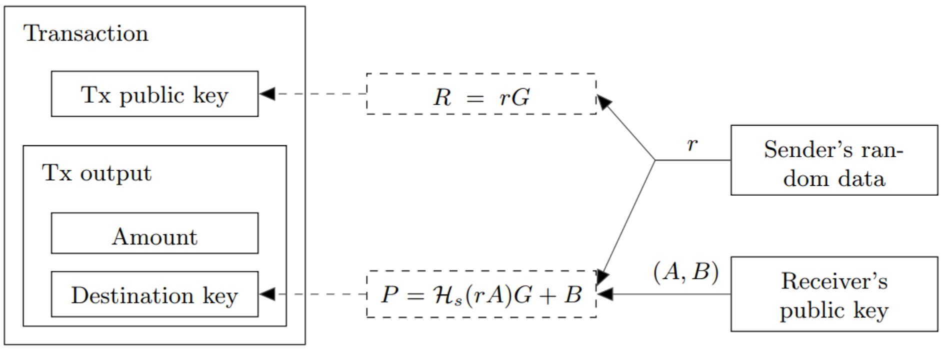


Figure 2.1: Standard transaction structure [19]

6. The payee can recover the corresponding one-time private key x=Hs(aR)G + b, so P=xG. Here (a,b) are the private key pair of Payee. He can send this output at any time by signing the transaction with x.

Hs: cryptographic hash function	**(A,B):** payee's public keys	**r:** random value
P/P': one time public key	**(a,b):** payee's private keys	**G:** a base point

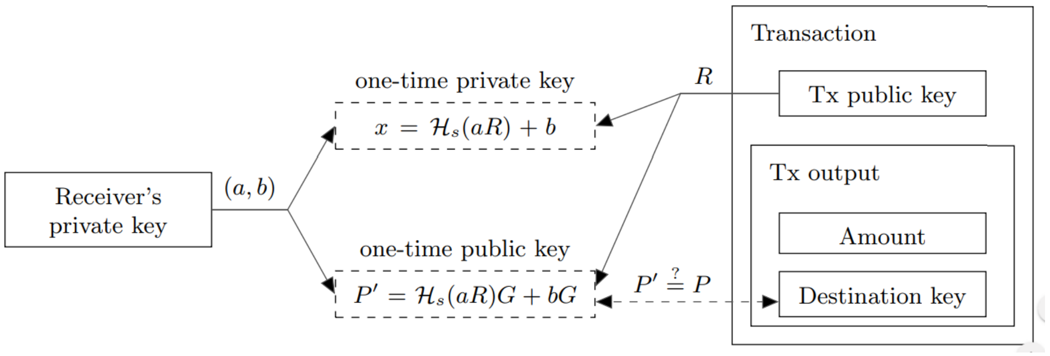


Figure 2.2: Incoming transaction check [19]

2.3 Related Works

This section discusses the related works on Monero untraceability features and blockchain analysis platforms.

2.3.1 Monero untraceability guarantee

Most of the researchers motivated by two prior works on privacy analysis of Monero: MRL-001 [20] and MRL-004 [21] research papers authored by Surae Noether*, Sarang Noether and Adam Mackenzie and Monero Core Team [22]. This prior works reported on the theoretical possibility of mounting the attacks for Monero. But the research gap of this is the impact of the attacks in real-world scenarios. Reason for this is, they have an only concern about these attacks in theoretically. In [20], the researchers describe a plausible attack on a ring-signature based anonymity system. But they have only considered an active attacker that must own coins used in previous transactions. In [21], researchers have discussed a passive attack scenario and provided a simulation analysis predicting that the mandatory 2-mixin minimum (implemented in version 0.9) would "allow the system to recover from a passive attack quite quickly". Here also have the same research gap of they have not measured the impact of the attacks in real-world scenarios.

Due to providing these research gaps from Monero research team and core developers, some researches have tried to fill this gap by quantifying the existing and past threat on the blockchain data. They provide some attacks to show the risk of using no mix-ins in practice, how often the risk may arise, how the impact evolved the overtime and how far cascade effect can propagate.

A Traceability Analysis of Monero's Blockchain [17] research paper has evaluated the deducibility attack in Monero to fill the above research gap. Authors of this paper are Amrit Kumar, Shruti Tople, Clément Fischer, Prateek Saxena from the National University of Singapore. In this research paper, they have quantified the efficacy of three attacks on Monero's untraceability guarantee, which promises to make it hard to trace the origin of a received fund, by analyzing Monero blockchain data. They have used publicly available Monero blockchain data from the first transaction on April 18th 2014 up to the transaction on February 6th 2017 as the sample data set. In [17], researchers have developed three attacks routines which gives statistically how much transactions are traceable. Their observations are based on these three attacks.

Attack I: Leveraging Zero Mix-ins-

Attack I presence of inputs spent using zero mix-ins. According to their observations over 65% of data set inputs used zero mix-ins (anonymity set size of one) and are trivially traceable. Overview of attack I routine is as follows,

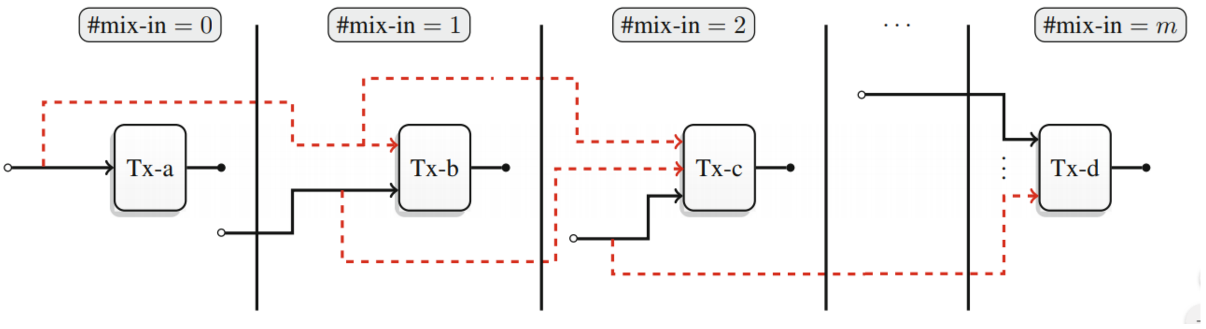


Figure 2.3: Attack I: Cascade effect due to zero mix-ins [17]

Each transaction has only one input (left of the transaction) and one output (on the right). The number of mix-ins used increases from left to right. Dashed lines represent the input keys identified as a mix-in. Lines in bold are the real input keys being spent. According to figure 2.3, from left to right mix-ins are cascade for right side transactions. This cascade gives traceability of another 22% of the inputs, leading to a total of 87.9% of traceable inputs.

Attack II: Leveraging Output Merging-

Attack II functions under the assumption that while creating a transaction, it is unlikely to choose several mix-ins that are outputs of a single previous transaction. Based on this assumption researchers have stated that Attack II has an overall true positive rate of 87% to identify the real key being spent in the transaction due to merging output concept. Overview of attack II routine is as follows,

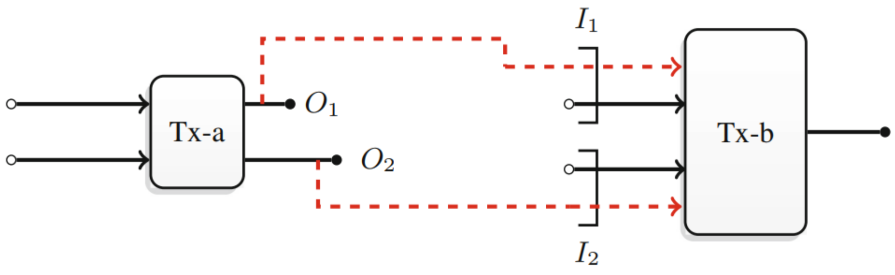


Figure 2.4: Attack II: Leveraging Output Merging [17]

Tx-a is a transaction with one input that uses one mix-in. It has two outputs O1 and O2. Tx-b is another transaction that has two inputs I1 and I2. Each input again has one mix-in. Both I1 and I2 include outputs of Tx-a. According to Attack II, the input keys O1 and O2 represented using the dashed line are the real keys being spent in Tx-b.

Attack III: Temporal Analysis-

Attack III leverages the fact that an output does not remain unspent for an infinite time. This is very accurate and very often the most recent output is the real one being spent. And also, they have shown user spending patterns do not follow the expected (variant of) triangular distribution by this attack.

According to the [17], researchers have done works to evaluate the Monero's untraceability guarantee. But the research gap of here is that they have not provided any method or way to verify these statistical results obtained by three attacks for any other researcher. There is no any tool or method to verify these numbers. And also, there are some limitations in [17]. Some attacks they have not tested in the second generation of Monero called Monero RingCT transactions [23]. And also they have stated that Attack II can be used to break the unlinkability guarantee in Monero. But they have not focused on this because of the entire paper have focused only on traceability analysis of Monero blockchain.

An Empirical Analysis of Traceability in the Monero Blockchain [18] research paper empirically evaluates weaknesses in Monero's mixin sampling strategy to fill the research gap of evaluating theoretical attacks in real-world scenarios. Authors of this paper are Malte Möser*, Kyle Soska, Ethan Heilman, Kevin Lee, Henry Heffan, Shashvat Srivastava, Kyle Hogan, Jason Hennessey, Andrew Miller, Arvind Narayanan, and Nicolas Christin. They have used Monero blockchain data from the first transaction on April 18th 2014 up to the transaction on February 6th 2017 as the sample data set of the research.

In [18], researchers have stated that they have evaluated the two weaknesses in Monero mixin sampling strategy. First, about 62% of transaction inputs with one or more mixins are vulnerable to chain-reaction analysis. That means is the real input being spent can be deduced by the mixin elimination. Second, Monero mixins are sampled in such a way that they can be easily distinguished from the real coin by their age distribution. That means the real input being spent is usually the newest input. For evaluation of the weaknesses [18], researchers have implemented a Neo4j graph database including data up to block 1288774. Based on that they have built iterative algorithms to deduce the actual input being spent. In the latter part of the research paper [18] they have addressed on after removing mining pool activity, there remains a large amount of potentially privacy-sensitive transactions that are affected for weaknesses. But the current phase of this research mining pool activities are not considered. Research gap of this paper [18] if that, there is no any method or way to verify the results they have obtained during the research. The main limitation of this paper [18] is they considered only about Monero's mix-in sampling strategy.

2.3.2 Blockchain Analysis Platforms

When considering a tools that can be used for analyze and explore the blockchain technology, BlockSci is the currently best performing analyzing and exploring tool support for the different blockchains. Such as Bitcoin, Litecoin, Namecoin, and Zcash. BlockSci has developed at Princeton University and they have published a paper including its processes and performance of the BlockSci.

BlockSci: Design and applications of a blockchain analysis platform [13] paper describe the functionality of BlockSci. In [13] the main limitation of this is BlockSci is unsupported for some blockchains. Such as Monero and Ethereum. In [13] authors have stated the reason for it is Monero and Ethereum does not follow "one-input one-output" paradigm. Monero uses mix-ins for its transaction inputs. Therefore, Monero internal structure is different with comparing to the other blockchains' internal structures.

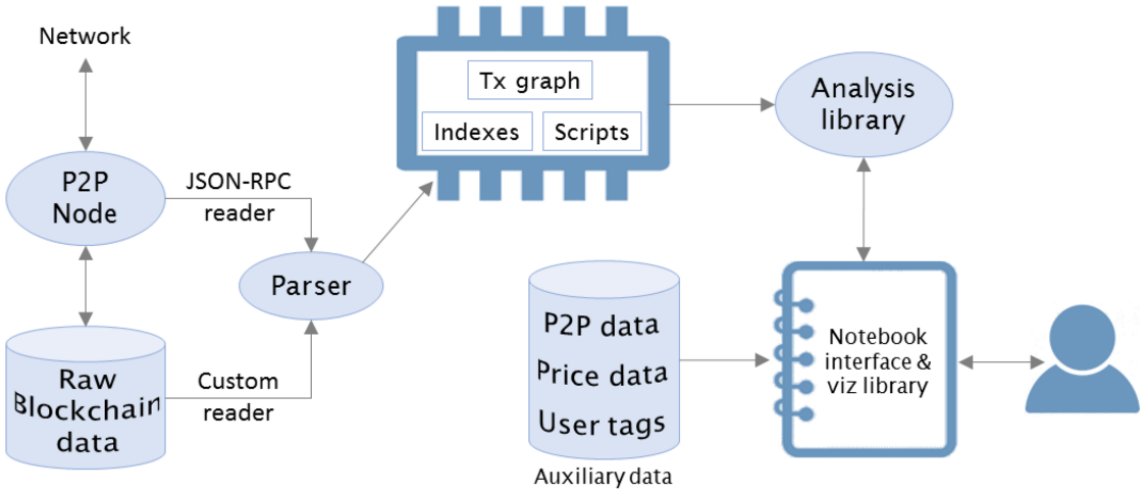


Figure 2.5: Overview of BlockSci's architecture [13]

According to the paper [13], their architecture has four main sections. First, Recording and importing data. BlockSci uses the publicly available blockchain data and it importing data by connecting to the blockchain network. Second, Parser. The parser converts all the raw blockchain data into well-structured way (core blockchain data) which can use for analyze the blockchain. Third, Core blockchain data, these are the output of the parser which is the primary dataset for analysis. Fourth, BlockSci Analysis Library. BlockSci Analysis Library loads this data as an in-memory database, which the user can either query directly or through a Jupyter notebook interface.

Considering other blockchain analysis tools, some are special purpose blockchain analysis tools and some are databases that have used for blockchain analysis. Rubin presents BTCSpark [24] which is a distributed blockchain analysis platform based on Apache Spark, Bartoletti et al. present a Scala-based blockchain analysis library [25] are some blockchain analysis tool and most of these tools are developed for analysing the Bitcoin blockchain. In [18], Malte Möser et al. have used Neo4j graph database which includes 1288774 blocks (11.5GB of data in total) to extract the relevant information from Monero blockchain. But they have not mentioned the accuracy, performance details of the method they have used during the research. Onion Monero Blockchain Explorer [26] is a tool, most of the users use to explore the Monero blockchain. But it is a JSON API based data exploring tool and it is not supported for querying, filtering and analyzing the Monero blockchain data.

2.4 Summary

This chapter mainly focused on fundamental concepts of Monero transactions based on CryptoNote 2.0 protocol, related works of Monero untraceability guarantee and related works of blockchain analysis platforms. Following table describes the summary of related works including the main limitations and key findings of each research.

Table 2.1: Related works on traceability analysis of Monero blockchain

Title	Main focus	Limitations	Key Findings
Traceability Analysis of Monero's Blockchain [17]	Traceability Analysis	Not consider about unlinkability in Monero. Attacks are not verified. No any implementation details about the attacks.	Attack I: Leveraging Zero Mix-ins finds that over 65% of inputs have zero mix-in. Attack I total traceable transaction percentage as 89%. Attack II: Leveraging Output Merging finds 87% transactions are traceable. Attack III: Temporal Analysis finds the most recent output is the real one being spent.
An Empirical Analysis of Traceability in the Monero Blockchain [18]	Traceability Analysis	Not consider about unlinkability in Monero. Attacks are not verified. No any implementation details about the attacks.	62% of transaction inputs with one or more mix-ins are vulnerable to "chain reaction" analysis Monero mix-ins are sampled in such a way that they can be easily distinguished from the real coins by their age distribution.

Chapter 3

Design

3.1 Introduction

This chapter provides the design methodology of the research project. It contains the main two designs. The first design is the MoneroSci tool which is an optimized environment for exploring and analyzing the Monero blockchain. The second design is the attacks for quantifying unlinkability and untraceability guarantee of Monero blockchain.

3.2 Design of MoneroSci Analysis Tool

Analysis of blockchain data is a very important and useful thing for both scientific and industrial applications. MoneroSci is the optimized analysis platform which going to develop within this study, can be used to analyze the CryptoNote 2.0 protocol base cryptocurrencies such a Monero, Bytecoin. As discussed in the methodology in chapter one, MoneroSci tool consists of three main components. Parser, Core Blockchain data and Analysis Library. MoneroSci uses two routes to import data from the raw blockchain and it uses the parser to convert the raw blockchain data into well-structured data called as Core Blockchain Data. Then Core Blockchain Data is exposed to the users by using a data interface called MoneroSci Analysis Library. Figure 3.1 shows the designed high-level architecture diagram of the MoneroSci tool within this study.

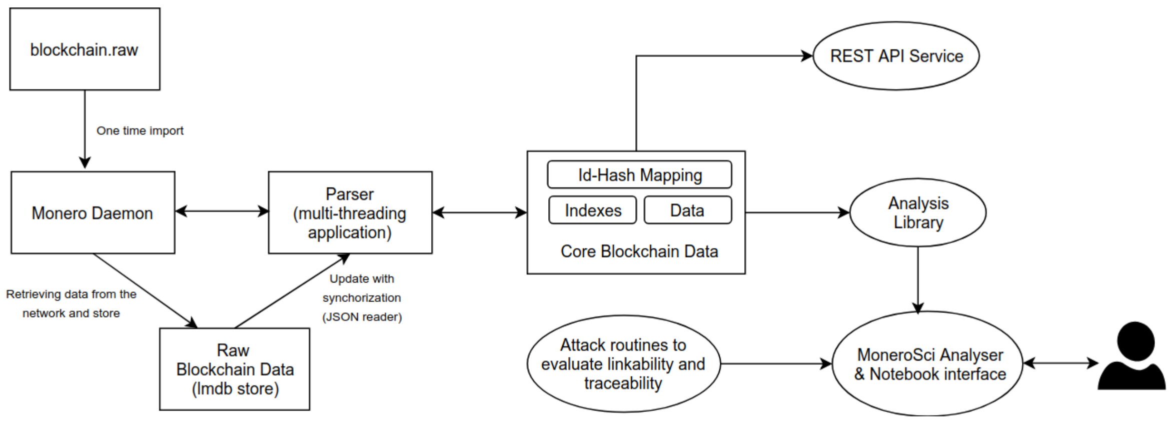

blockchain.raw
One time import
Monero Daemon
Parser (multi-threading application)
Id-Hash Mapping
Indexes
Data
Core Blockchain Data
REST API Service
Analysis Library
Retrieving data from the network and store
Update with synchorization (JSON reader)
Raw Blockchain Data (lmdb store)
Attack routines to evaluate linkability and traceability
MoneroSci Analyser & Notebook interface

Figure 3.1: MoneroSci high-level architecture diagram

3.2.1 Importing Data

There are two main routes for importing raw blockchain data into the MoneroSci. MoneroSci is directly connected with the Monero Daemon which is the software that ships with the Monero tree developed by Monero developers. Monero daemon handles the Monero blockchain and gets new block data from the Monero peer to peer network if new blocks exist. And also, Daemon will be working as a synchronizer between MoneroSci data and Monero network data. The duty of the Monero daemon is getting block data from the network and storing them in the lmdb data storage in the secondary storage. Therefore, MoneroSci has the ability to get Monero blockchain data in both ways. Directly from the Monero daemon or load data from previously stored lmdb files.

3.2.2 Parser

Reading blockchain data of on-disk format is highly inefficient. Therefore, it needs a single representation of the raw blockchain data can easily fit into the memory. The parser does is convert raw blockchain data into the well-structured dataset called Core Blockchain Data. Considering the structure of MoneroSci parser figure 3.2 shows an overview of it.

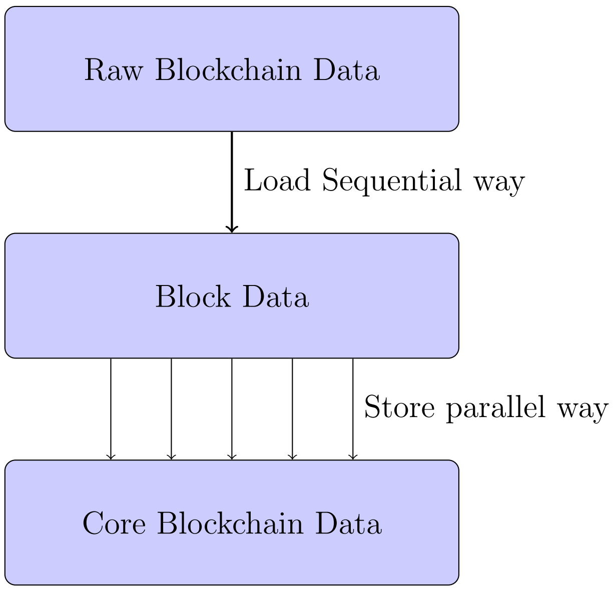


Figure 3.2: Overview of MoneroSci parser in multithreaded environment

Parsing the raw blockchain data is sequential. The blockchain must be processed sequentially because parser maintains the indexes and blockchain IDs such as block id, transaction id, input id, key image id, ring member id, and output id. Reason for the maintain the indexes and ids is indexes and ids are the main key point of performance. Currently, Monero blockchain has more than 1.6 million blocks, 4.1 million transactions, 20 million inputs and outputs. Therefore, It is impossible to do the process of the whole blockchain in a sequential way. Because of that MoneroSci parser has developed as a multithreaded application with the bunch of optimization techniques.

- Replace hash pointers with IDs to shrink the data structure and optimize linkage.
- De-duplicate public keys and hash data.
- Apply bloom filter to optimize the searching elements in the blockchain.
- Create indexes for every element in the blockchain.

Figure 3.3 shows the execution flow of the MoneroSci parser. One thread loads the blockchain elements on by one into the main memory and other threads store the blockchain elements such as blocks, transactions, inputs, outputs, mix-ins in the database when the conditions are satisfied.
The multithreaded parser works as a producer-consumer. One thread act as the producer. It loads the raw blockchain data block by block from the daemon or lmdb

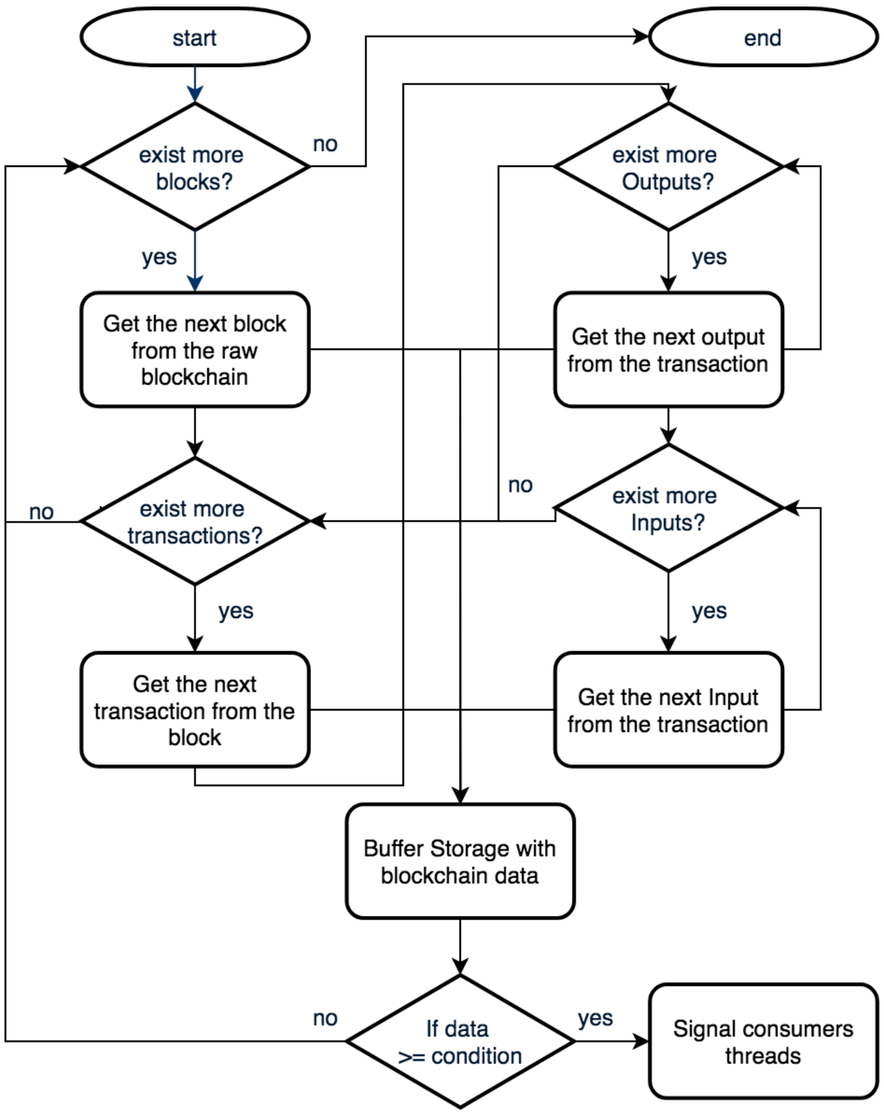

start
end
exist more blocks?
no
yes
Get the next block from the raw blockchain
exist more transactions?
no
yes
Get the next transaction from the block
exist more Outputs?
yes
Get the next output from the transaction
exist more Inputs?
no
yes
Get the next Input from the transaction
Buffer Storage with blockchain data
If data >= condition
no
yes
Signal consumers threads

Figure 3.3: Execution flow of MoneroSci parser

data file into the main memory. It checks the all transactions of the block, all the inputs and outputs of each transaction in that block and ring members(mix-ins) of each inputs for each transactions in that block. Raw blockchain data means hard disk data. If that thread loads the sufficient dataset with the indexes and ids into the memory it signals the other consumer threads in the server. Consumer threads store the loaded blockchain data in relevant data stores inside the Core Blockchain Data.

3.2.3 Core Blockchain Data

The output of the MoneroSci parser is the Core Blockchain Data, which is the dataset for the analysis of Monero blockchain. It contains three main data sections.

- **ID-Hash mapping -** stores all the public keys and hashes of blockchain elements with separate unique ID. As an example, Hash of 110th block save as key value pair. Key is the Hash. Value is the id generated by the parser.

  ```
  { "Element_HASH" : "Generated ID"}
  ```

- **Monero-Indexes -** stores all the blockchain elements' indexes. It includes all the generated elements ids in the relational database. It helps to query the blockchain element data in faster.
- **Monero-Data -** stores all the details of blockchain elements in the relational database.

3.2.4 Analysis Library

The way of access Core Blockchain Data is using MoneroSci Analysis Library. It contains a set of classes and methods for every element of the Monero blockchain which can use for analyze and explore the Monero blockchain easily. It act as a data interface for the out side. MoneroSci Analysis Library includes the following main classes,

- Blockchain Class - contains blockchain analysis methods
- Block Class - contains block analysis methods
- Tx Class - contains transaction analysis methods
- Input Class - contains input analysis methods
- Output Class - contains output analysis methods
- Ring Class - contains rings(mix-ins) analysis methods

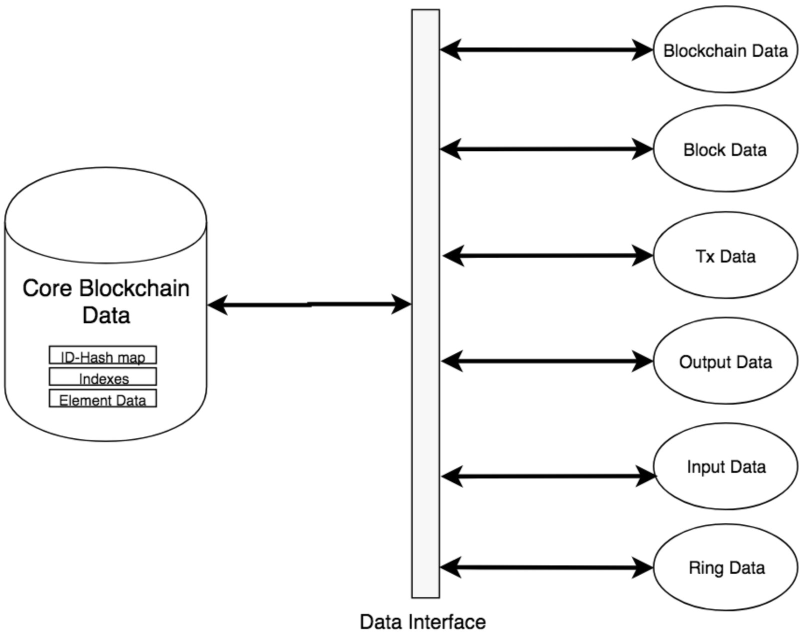


Figure 3.4: Overview of MoneroSci Analysis Library

Blockchain class includes a set of methods and attributes for query the data contains blockchain. It mainly includes general methods which applied for the whole Monero blockchain. Block class contains a set of methods and attributes to query the data of blocks in the blockchain. Tx class represents a transaction inside the block. Input and Output classes represent inputs and outputs contain in the transaction and it also contains a set of methods and attributes to query the data of it. Ring class is a special class which represents the mix-ins of the inputs. Analysis Library contains some sort of methods which are made easier in analysis kinds of stuff. For example, if someone wants to get transactions which are used more than X value as transaction fee it can be done in one query. Therefore Analysis Library exposes data retrieving and analyzing interface for the users.

3.3 Designs of Unlinkability and Untraceability Attacks

For quantifying the untraceability guarantee in Monero there are two main attacks proposed by Amrit Kumar et al. Attack I: Leveraging Zero Mix-ins and Attack II: Leveraging Output Merging [17].

3.3.1 Attack I: Leveraging Zero Mix-ins

This attack mainly looking for the traceability guarantee of non-RingCT transactions in Monero blockchain. If a transaction uses zero mixin strategy that's mean in the transaction input it only includes actual inputs being spent, that transaction is trivially traceable. And also, if that identified actual public key used for one mixin strategy, that's mean in the transaction input include two keys one as actual key and one as a decoy, here actual key also traceable due to the cascade effect of zero mixin to the one mixin strategy. Then one mixin identified keys are effect to find actual keys in two mix-ins, two mix-ins identified keys are effect to find actual keys in three mix-ins and so on. Finally, it finds set of traceable actual spent keys due to this zero mixin strategy in non-RingCT transactions. Following algorithm illustrate the design of attack I proposed in [17].

Algorithm: Heuristic I

```
n : the number of iterations
T : the maximum block height.
spentKeys = {}
# Each entry of keysToAnalyze is a list of keys.
keysToAnalyze = []

for height <= T do
    # Retrieve block with given height
    block = getBlock(height)
    # Retrieve non-coinbase transactions
    transactions = getTransactions(block)
    for tx in transactions do
    # Retrieve inputs in the transaction
        inputs = getInputs(tx)
        for input in inputs do
            # Retrieve input keys in this input
            inKeys = getInputKeys(input)
            # Add all keys as a list
            keysToAnalyze.add(inKeys)
```

```
for n times or until spentKeys reaches stable point
   for inKeys in keysToAnalyze do
      # Store keys that are note spent.
      untracedKeys = {}
      for inKey in inKeys do
         if inKey < spentKeys then
            untracedKeys = untracedKeys + inKey

      if |untracedKeys| = 1 then
         spentKeys = spentKeys + untracedKeys
         # Remove newly identified spent key
         # from each entry of keysToAnalyze
         keysToAnalyze.removeAll(untracedKeys)

# A set spentKeys of spent output keys.
return spentKeys
```

3.3.2 Attack II: Leveraging Output Merging

Attack II runs under the heuristic of creating a new transaction it is unlikely to choose more than one mix-ins (ring member) from a single previous transaction which are outputs of it. In order to simplify the design, it defined two terms. One is source transaction and second is destination transaction. Here destination transaction uses more than one output of source transaction as its input. Therefore to find the source-destination transactions first, need to find all the outputs of the Monero transactions on top of the MoneroSci. Then need to find what are the inputs that these outputs act as ring members. In Monero output public keys and input ring member, public keys are same. From that, it can find a set of transactions these outputs use to spent or mix-in. Then can find what are the transactions used more than two outputs of a single transaction outputs for its input ring. Following algorithm illustrate the design of attack II proposed in [17].

Algorithm: Heuristic II

```
Data: T : the maximum block height to analyze.

candidateSets = {}
for height <= T do
   block = getBlock(height)
   # Retrieve non-coinbase transactions
   transactions = getTransactions(block)
```

```
    for tx in transactions do
        # Retrieve outputs in the transaction
        outputKeys = getOutputKeys(tx)
        # Invoke analyzeOutput( , ) defined below
        candidateSets = candidateSets +
        analyzeOutput(height, outputKeys)

#A list candidateSets of linked inputs/outputs.
return candidateSets

def analyzeOutput(height1, outputKeys) as
    candidateSets = {}
    for height2 in [height1 + 1,T ] do
        block = getBlock(height2)
        # Retrieve non-coinbase transactions
        transactions = getTransactions(block)
            for tx in transactions do
                nonEmptyIntersections = 0
                candidateKeys = {}
                for input of tx do
                    # Retrieve input keys in this input
                    inKeys = getInputKeys(input)
                    if inKeys and outputKeys , {} then
                        #add new candidate keys
                        candidateKeys =
                        candidateKeys + (inKeys and outputKeys)

            if 2 <= nonEmptyIntersections and
                2 <= |candidateKeys| then
                candidateSets.add(candidateKeys)

return candidateSets
```

3.3.3 Extend Attack II Design for Unlinkability

Based on the heuristic of Attack II, Attack II can extend for the quantify the unlinkability guarantee of Monero blockchain. If one transaction uses more than two outputs of a single previous transaction for its input ring members and observed that those keys are the real ones being spent during the transaction, then those outputs and inputs link the two transactions and they must belong to the same Monero user. This attack generate two main link graphs for non-RingCT and RingCT transactions. There after based on the graphs can do empirical analysis about the unlinkability guarantee in Monero.

Algorithm: Extended Heuristic II

```
Data: condidateSet  #Candidate set return from Attack II
Data: ringctIdList #RingCT transaction ID list

ringct_linkable_set = dict()
nonringct_linkable_set = dict()

for set in condidateSet do
   #Retrieve candidate set one by one
   if set in ringctIdList then
      #This is a RingCT transaction candidate set
      if set in ringct_linkable_set then
         #Add candidate set as RingCT
         ringct_linkable_set[set] += condidateSet[set]
      else:
         ringct_linkable_set[set] = condidateSet[set]
   else:
      #This is a non RingCT transaction candidate set
      if set in nonringct_linkable_set then
         #Add candidate set as non RingCT
         nonringct_linkable_set[set] += condidateSet[set]
      else:
         nonringct_linkable_set[set] = condidateSet[set]

#Two linkable Graphs of RingCT and non-RingCT Txes
return generateGraph(ringct_linkable_set),
       generateGraph(nonringct_linkable_set)

def generateGraph(candidateSet) as
   nodes = {}
   relationships = {}

   for set in candidateSet do
      #Retrieve key and values
      nodes.add(set)
      #load values of the key
      for value in candidateSet[set] do
          relationships.add(set,value)

   #Initialize graph instance using nodes and relationships
   graph = nx.Graph(nodes,relationships)
return graph
```

Following figure illustrate the structure of generating linked transaction graphs using extended attack. Nodes(Tx) represents the transactions and edges(t) represents the linkage between two transactions with timestamp as the weight of it.

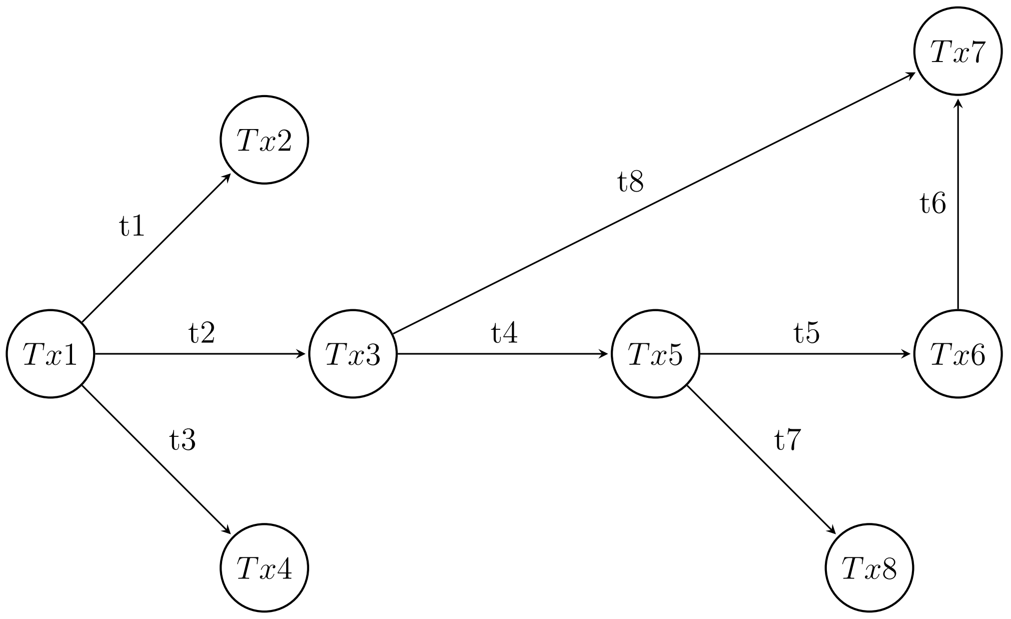


Figure 3.5: Structure of the linked transaction graph

3.4 Summary

This chapter provided a detailed description on the research design. The research design consists two main designs. First one is the design of the MoneroSci tool which is an optimized environment to explore and analyze the Monero blockchain. The second design consist set of attacks which can use to quantify the unlinkability and untraceability guarantee in Monero blockchain. These attacks designs are compatible to implement on top of the MoneroSci tool.

Chapter 4

Implementation

4.1 Introduction

This section elaborates the implementation details of the proposed solution of optimized environment to explore and analyze the Monero blockchain, implementation of traceability attacks conducted by Amrit Kumar et al. [17] and implementation details of the extended attack to evaluate linkability guarantee of Monero blockchain.

4.2 Software Tools

For the import Monero blockchain data to the parser, Monero Daemon uses for it. Monero Daemon acts as the synchronizer between Monero peer to peer network and the MoneroSci analysis tool. The proposed MoneroSci was implemented using C++ 11 programming language, SQLite3 database management system and RocksDB an embeddable persistent key-value storage. C++ 11 programming language was used to develop whole the MoneroSci analysis tool including the MoneroSci parser and MoneroSci Analysis Library. C++ Boost library was used to develop the python interface of MoneroSci Analysis Library. C++ Boost library creates a shared library for python called ”monerosci” which can import as a library in python to access the MoneroSci Core Blockchain Data. RocksDB storage used to store the ID-Hash mapping key-pair values in every element in the Blockchain as discussed in the design chapter. Such as blocks, transactions, inputs, outputs, ring members and last ids. SQLite3 was used to store all the data of the blockchain in a relational way. It contains indexes databases and detail databases separately. Jupyter Notebook uses to expose the analysis library for the end users. It compatible to works with Python 2.x or 3.x versions.

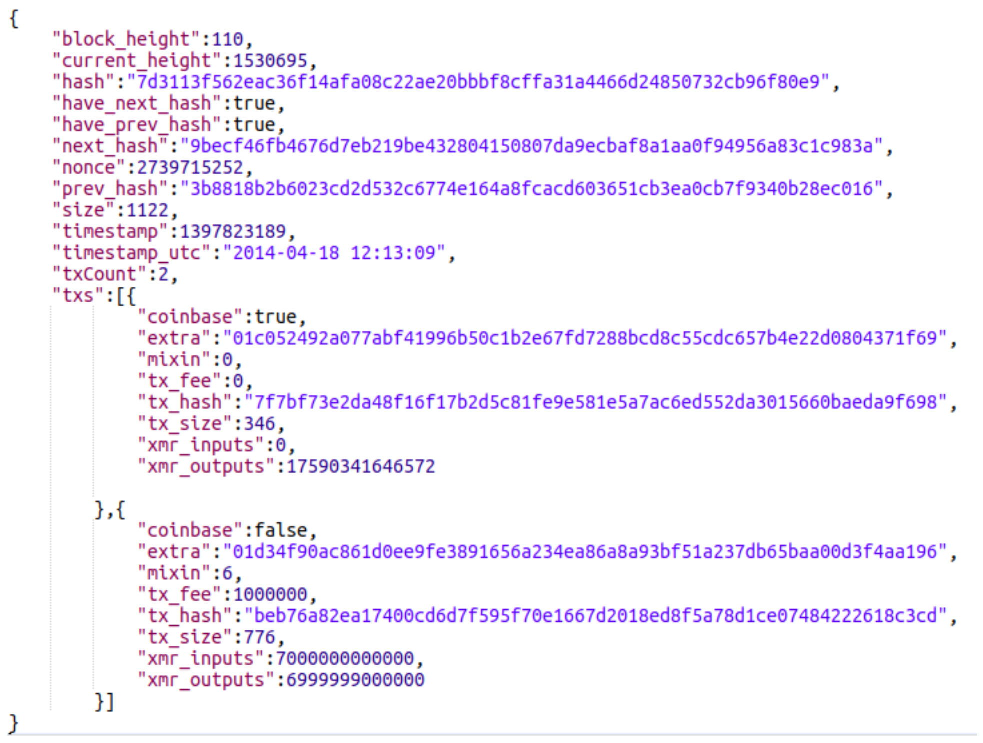
4.3 Implementation Details - MoneroSci Tool

As mentioned in chapter 3, MoneroSci tool consists of three main components. MoneroSci parser, Core Blockchain Data and MoneroSci Analysis Library.

4.3.1 Parser

MoneroSci parser is a multithreaded application which connects with Monero Daemon to retrieve the raw blockchain data from the Monero peer to peer network or read the raw blockchain data from lmdb files. When running the parser, first it checks the connection of Monero Daemon, if the daemon is not found parser proceed ahead with the lmdb data storage. MoneroSci parser retrieves the raw blockchain data as in JSON format by using C++ nlohmann::json structs. Following figures illustrate the JSON format of block data, transaction data, output data and input data retrieve from the Monero raw blockchain.

JSON format of block data:-



```
{
    "block_height":110,
    "current_height":1530695,
    "hash":"7d3113f562eac36f14afa08c22ae20bbbf8cffa31a4466d24850732cb96f80e9",
    "have_next_hash":true,
    "have_prev_hash":true,
    "next_hash":"9becf46fb4676d7eb219be432804150807da9ecbaf8a1aa0f94956a83c1c983a",
    "nonce":2739715252,
    "prev_hash":"3b8818b2b6023cd2d532c6774e164a8fcacd603651cb3ea0cb7f9340b28ec016",
    "size":1122,
    "timestamp":1397823189,
    "timestamp_utc":"2014-04-18 12:13:09",
    "txCount":2,
    "txs":[{
            "coinbase":true,
            "extra":"01c052492a077abf41996b50c1b2e67fd7288bcd8c55cdc657b4e22d0804371f69",
            "mixin":0,
            "tx_fee":0,
            "tx_hash":"7f7bf73e2da48f16f17b2d5c81fe9e581e5a7ac6ed552da3015660baeda9f698",
            "tx_size":346,
            "xmr_inputs":0,
            "xmr_outputs":17590341646572

        },{
            "coinbase":false,
            "extra":"01d34f90ac861d0ee9fe3891656a234ea86a8a93bf51a237db65baa00d3f4aa196",
            "mixin":6,
            "tx_fee":1000000,
            "tx_hash":"beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd",
            "tx_size":776,
            "xmr_inputs":7000000000000,
            "xmr_outputs":6999999000000
        }]
}
```

To retrieve the block data, it only requires the block height of block the block. To retrieve the transaction data, it only requires the transaction hash of the transaction.

JSON format of transaction data:-

```
{
    "blk_height":["110"],
    "blk_timestamp":["2014-04-18 12:13:09"],
    "blk_timestamp_uint":[1397823189],
    "confirmations":[1530585],
    "extra":["01d34f90ac861d0ee9fe3891656a234ea86a8a93bf51a237db65baa00d3f4aa196"],
    "from_cache":[false],
    "has_inputs":[true],
    "inputs":[↔],
    "inputs_no":[1],
    "inputs_xmr_sum":["7.000000000000"],
    "is_ringct":[false],
    "outputs":[↔],
    "outputs_no":[8],
    "outputs_xmr_sum":["6.999999000000"],
    "rct_type":[0],
    "tx_blk_height":[110],
    "tx_fee":["0.000001000000"],
    "tx_hash":["beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd"],
    "tx_prefix_hash":["ccabefb57635c09cfe66af861f11e1a379cd0de0e030409ab3c26418cf302166"],
    "tx_pub_key":["d34f90ac861d0ee9fe3891656a234ea86a8a93bf51a237db65baa00d3f4aa196"],
    "tx_size":["0.7578"],
    "tx_version":[1]
}
```

Transaction dataset includes the details of key images (inputs), ring members (mix-ins) and the stealth addresses (outputs).

JSON format of output data:-

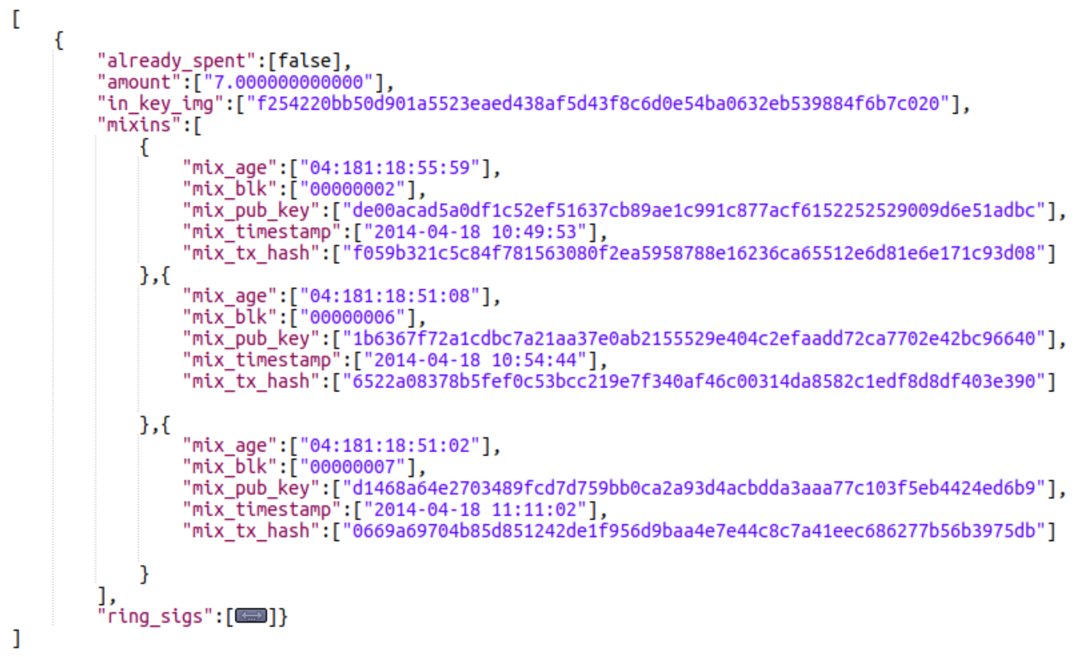
```
[
    {
        "amount":["0.000000646572"],
        "amount_idx":["0"],
        "num_outputs":[1],
        "out_pub_key":["eab03067870349139bee7eab2ca2e030a6bb73d4f68ab6a3b6ca937214054cda"],
        "output_idx":["00"],
        "unformated_output_idx":[0]

    },{
        "amount":["0.000001000000"],
        "amount_idx":["12"],
        "num_outputs":[42333],
        "out_pub_key":["8bbe23b57ea9bae53f12da93bb57bf8a2e40598d9fccd10c2921576e987d93cd"],
        "output_idx":["01"],
        "unformated_output_idx":[1]

    },{
        "amount":["0.000040000000"],
        "amount_idx":["14"],
        "num_outputs":[58711],
        "out_pub_key":["468738e391f07c4f2b356f7957160968e0bfef6e907c3cee2d8c23cbf04b0896"],
        "output_idx":["02"],
        "unformated_output_idx":[2]

    }
]
```

JSON format of input data:-



```
[
    {
        "already_spent":[false],
        "amount":["7.000000000000"],
        "in_key_img":["f254220bb50d901a5523eaed438af5d43f8c6d0e54ba0632eb539884f6b7c020"],
        "mixins":[
            {
                "mix_age":["04:181:18:55:59"],
                "mix_blk":["00000002"],
                "mix_pub_key":["de00acad5a0df1c52ef51637cb89ae1c991c877acf6152252529009d6e51adbc"],
                "mix_timestamp":["2014-04-18 10:49:53"],
                "mix_tx_hash":["f059b321c5c84f781563080f2ea5958788e16236ca65512e6d81e6e171c93d08"]
            },{
                "mix_age":["04:181:18:51:08"],
                "mix_blk":["00000006"],
                "mix_pub_key":["1b6367f72a1cdbc7a21aa37e0ab2155529e404c2efaadd72ca7702e42bc96640"],
                "mix_timestamp":["2014-04-18 10:54:44"],
                "mix_tx_hash":["6522a08378b5fef0c53bcc219e7f340af46c00314da8582c1edf8d8df403e390"]

            },{
                "mix_age":["04:181:18:51:02"],
                "mix_blk":["00000007"],
                "mix_pub_key":["d1468a64e2703489fcd7d759bb0ca2a93d4acbdda3aaa77c103f5eb4424ed6b9"],
                "mix_timestamp":["2014-04-18 11:11:02"],
                "mix_tx_hash":["0669a69704b85d851242de1f956d9baa4e7e44c8c7a41eec686277b56b3975db"]

            }
        ],
        "ring_sigs":[↔]}
]
```

In the beginning, MoneroSci parser checks the number of cores available in the server and divide workload among the cores by creating threads. To achieve this task C++ threads are used. As discussed in design chapter, overall parser activity is like as producer-consumer scenario.

The main thread went through all the blocks, transaction, inputs, outputs and ring members and its loads the data into the buffers. MoneroSci parser maintains a set of buffers to store the blocks, transactions, inputs, key images, stealth addresses and ring members data in the memory. If the buffer filled with the relevant data, main thread signals for the particular thread which is responsible for that buffer to consume the data from the buffer and store the data inside the SQLite database as batch insert mechanism. Each thread stores the ID-Hash mapping of each element by giving a unique ID for each item. Those ID-Hash key-value pairs stored in the persistent key-value storage called RocksDB developed by Facebook. Following code segment illustrate the process of the main thread.

```
void mainFunction(page& monerosci,int startBlock,int endBlock,uint64_t
     blk_height){
     //load last ids to global variables
     getLastIdList();    //load the last blockchain ids
     time_t Start, End;  // initialize the start and end time
    variables
     time (& Start);     // set start time

     int x;
     //Load the data sequentially into the memory
     for(x = startBlock; x <= endBlock; x++){
         xmrProcessor(monerosci, to_string(x)); //current block.
         cout << "MoneroSci-parser has parsed blockchain data of "+
           std::to_string(x)+ "/" << blk_height << endl;

         //benchmark the performance after 1000 blocks completed
         if(x%1000 == 0){
             time (& End);
             double timeUsage = difftime (End, Start);
             //trace benchmarks
             performanceAnalyser(timeUsage);
         }
     }

     //store the last id in rocksdb hash-id mapping
     storeLastIdList();
     //close the bloom-filter ring member connection
     ringMemberHashMap.close();

     //print the progress of the parser
     showCurrentStatus(x);
     // print the performance benchmarks of the parser
     printPerformance();
     // indicate all the blocks are loaded to the memory for the
    slaves
     isAllBlockDone = true;
}
```

4.3.2 Core Blockchain Data

Core Blockchain Data is the output of the MoneroSci parser. It is the dataset of MoneroSci which are used to analyze and query the Monero Blockchain. SQLite and RocksDB hard disk data storages are used to store the parsed blockchain data. Core Blockchain Data has three main data sections. ID-Hash mapping, Monero-indexes and Monero-data.

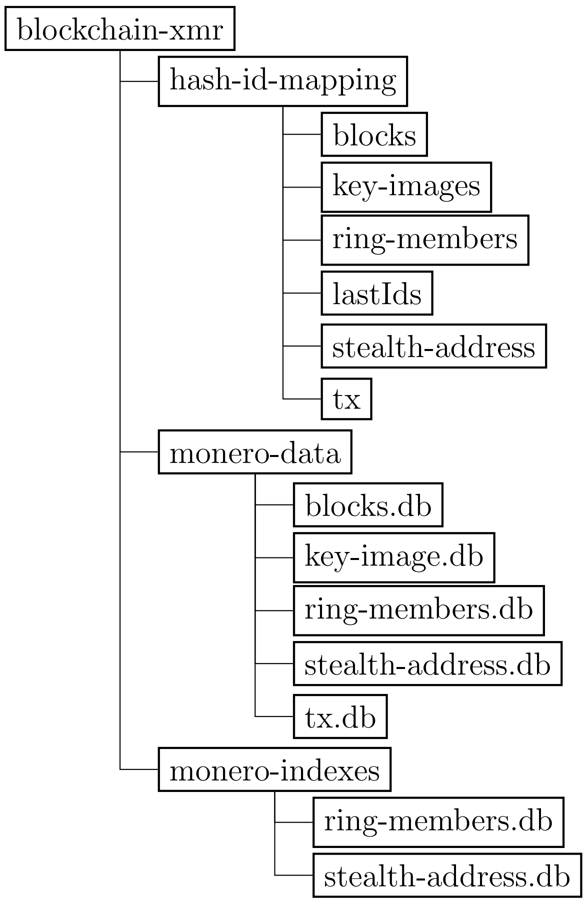


Figure 4.1: Structure of Core Blockchain Data

In above diagram, ”blockchain-xmr” is the root directory of Core Blockchain Data. ID-Hash mapper contains a unique id for every element in the blockchain. Generating a unique id for every element is doing by MoneroSci parser. In the analysis, MoneroSci Analysis Library first finds the id for a given hash or public key to reduce the complexity of the query. That approach makes analysis much easier than using long cryptographic hashes. The interface of the ID-hash mapper as follows,

```
using namespace std;

namespace hashmapper {

    class idHashMapperDB{
        //attributes
        string homeDirName;
        string subDirName;
        string storeDirName;
        string finalPath;
        rocksdb::DB* db;
        rocksdb::Options options;
        rocksdb::Status status;

    public:
        //constructor
        idHashMapperDB(string homePath,string subDir, string storeDir)
    ;

        //methods
        void insertKey(string key, string value);

        string getValueFromKey(string key);

        void deleteFromKey(string key);

        string isKeyExist(string key);

        void close();
    };
}
```

This contains set of methods to add new hash-id map, get value from hash, delete key-value pair and check if the hash is already exist in the database. Considering the monero-indexes and monero-data, SQLite database is used. It contains a set of tables which are used to store the data separately. SQL queries of those tables as follows,

```

//block detail table
char const *blockDetailsTable = "CREATE TABLE IF NOT EXISTS BLOCK_DATA
    ( DATA_ID INT PRIMARY KEY NOT NULL,HEIGHT INT NOT NULL,HASH TEXT
    NOT NULL, TIMESTAMP INT NOT NULL, TIMESTAMP_UTC TEXT NOT NULL, SIZE
    INT NOT NULL,TX_COUNT INT NOT NULL, NONCE INTEGER, PREV_HASH TEXT,
    NEXT_HASH TEXT);";
//block index table
char const *blockIndexTable = "CREATE TABLE IF NOT EXISTS BLOCK_INDEX(
    ID INT PRIMARY KEY NOT NULL,HASH_ID INT NOT NULL,DATA_ID INT NOT
    NULL);CREATE INDEX blk_index ON BLOCK_INDEX (HASH_ID);";
```

```

//stealth-address table
char const *saDetailTable = "CREATE TABLE IF NOT EXISTS SA_DATA (
    DATA_ID INTEGER PRIMARY KEY NOT NULL,P_KEY TEXT NOT NULL, AMOUNT
    TEXT,AMOUNT_INDEX TEXT,OUTPUT_IDX TEXT,NUM_OUTPUTS INTEGER);";

//stealth-address table
char const *saIndexTable = "CREATE TABLE IF NOT EXISTS SA_INDEX (
    DATA_ID INT PRIMARY KEY NOT NULL,BLOCK_ID INT NOT NULL,TX_ID INT
    NOT NULL, HASH_ID INT NOT NULL);CREATE INDEX output_index ON
    SA_INDEX (HASH_ID);";

//ring-member detail table
char const *rmDetailTable = "CREATE TABLE IF NOT EXISTS RM_DATA (
    DATA_ID INTEGER PRIMARY KEY NOT NULL,P_KEY TEXT NOT NULL,
    MIX_BLOCK_HEIGHT  INT, MIX_IDX TEXT, TX_HASH TEXT, MIX_TIMESTAMP
    TEXT, OUTPUT_INDEX INT,MIX_AGE TEXT);";

//ring-members index table
char const *rmIndexTable = "CREATE TABLE IF NOT EXISTS RM_INDEX (
    DATA_ID INT PRIMARY KEY NOT NULL,BLOCK_ID INT NOT NULL,TX_ID INT
    NOT NULL, KEY_IMAGE INT NOT NULL,HASH_ID INT NOT NULL);CREATE INDEX
      input_index ON RM_INDEX (HASH_ID);";

//tx details table
char const *txDetailsTable = "CREATE TABLE IF NOT EXISTS TX_DATA(
    DATA_ID INTEGER PRIMARY KEY NOT NULL, COIN_BASE INT NOT NULL,HEIGHT
     INT NOT NULL,HASH TEXT NOT NULL, CONFIRMATIONS INT, EXTRA TEXT,
    INPUT_COUNT INT NOT NULL, MIXIN INT, OUTPUT_COUNT INT NOT NULL,
    PAYMENT_ID TEXT, RCT_TYPE INT, TX_FEE TEXT,TX_VERSION INT,XMR_INPUT
     TEXT, XMR_OUTPUT  TEXT, TIMESTAMP INT NOT NULL,TIMESTAMP_UTC TEXT
    NOT NULL, SIZE TEXT  NOT NULL, IS_RINGCT INT, TX_PUB_KEY TEXT,
    TX_PREFIX_HASH TEXT);";

//tx index table
char const *txIndexTable = "CREATE TABLE IF NOT EXISTS TX_INDEX (
    DATA_ID INT PRIMARY KEY NOT NULL,BLOCK_ID INT NOT NULL,HASH_ID INT
    NOT NULL );CREATE INDEX transaction_index ON TX_INDEX (HASH_ID);";

char const *keyImageTable = "CREATE TABLE IF NOT EXISTS KI_DATA (
    DATA_ID INTEGER PRIMARY KEY NOT NULL,K_HASH TEXT NOT NULL,BLOCK_ID
    INT NOT  NULL,TX_ID INT NOT NULL,INPUT_AMOUNT TEXT, INPUT_IDX TEXT,
      MIXIN_COUNT INT);";
```

4.3.3 Analysis Library

MoneroSci Analysis Library is a python interface for analyzing the CBD. C++ Boost mainly used to implement this library. It creates a shared library called "monerosci" which can import to python interpreters to analyze the Monero blockchain data as discussed in design chapter. Analysis library contains six classes to access the CBD.

Blockchain class interface -

```cpp
class Blockchain{

public:
    // Constructor
    Blockchain(std::string url);

    // Methods
    int blockCount();

    int txCount();

    int keyImageCount();

    int key_image_range_total(int fromBlock, int toBlock);

    boost::python::list blocks();

    boost::python::list range(int from, int to);

    Tx tx_with_hash(string hash);

    Tx tx_with_index(int index);

    boost::python::list inputs_with_mixins(int mix_number);

    boost::python::list all_outputs(int startBlk, int endBlk);

    boost::python::dict outputs_with_spents(int sBLock,int eBlock);

    boost::python::list all_inputs(int startBlk, int endBlk);

    boost::python::list all_keyimages(int startBlk, int endBlk);

    boost::python::list all_ringCT(int startBlk, int endBlk);

    boost::python::dict all_txes_timestamps(int startBlk, int endBlk);

};
```

Block class interface -

```cpp
class Block{

public:
    //attributes
    int height;
    string hash;
    string timestamp;
    string timestamp_utc;
    int size;
    int tx_count;
    int block_index;
    int nonce;
    string prev_block_hash;
    string next_block_hash;

    // Constructor
    Block();
    Block(std::map<string,string> blockData);
    Block(int index);
    Block(string hashVal);

    // Methods
    string str();
    Block next_block();
    Block prev_block();
    boost::python::list coinbase_tx();
    long double fee();
    boost::python::list txes();

};
```

Tx class interface -

```cpp
class Tx {

public:
    //attributes
    bool is_coinbase;
    int block_height;
    string hash;
    int confirmation;
    string extra;
    int input_count;
    int mixins;
    bool is_ringct;
    int output_count;
    string payment_id;
    double fee;
    int version;
    double xmr_input;
    double xmr_output;
    int timestamp;
    double size;
    int tx_index;
    string prefix_hash;
    string public_key;

    // Constructor
    Tx();
    Tx(std::map<string, string> txData);
    Tx(int index);
    Tx(string hashVal);

    // Methods
    string str();
    boost::python::list outputs();
    boost::python::list inputs();
    Block blockObj();

};
```

Input class interface -

```
class Input {

public:
    //attributes
    int block_index;
    string public_key;
    int tx_index;
    int key_image_index;
    int mixin_count;
    double amount;

    // Constructors
    Input(int key_image_id);
    Input(std::map<string,string> inputData);
    Input(string key_image_key);

    // Methods
    string str();
    boost::python::list rings();
};
```

Output class interface -

```
class Output{

public:
    //attributes
    int block_index;
    string public_key;
    double amount;
    int tx_index;
    int output_index;
    int output_idx;
    string rm_key_id;

    // Constructors
    Output(int req_tx_index,int req_output_index);
    Output(std::map <string, string> outputData);

    // Methods
    string str();
    boost::python::list mixing_txes();
    void setRmKey(string key);
};
```

Ring class interface -

```
class Ring{

public:
    //attributes
    string public_key;
    int block_index;
    int tx_index;
    int mix_block_index;
    string mix_tx_hash;
    string timestamp;
    int mix_output_index;
    string mix_age;
    int ring_index;
    int key_image_index;
    int ring_data_id;

    //constructors
    Ring(int ki_id, int rm_id);
    Ring(std::map<string,string> rmData);

    // Methods
    string str();
    Tx origin_tx();
};
```

All these classes defined inside the PYTHON_MODULE method of C++ boost library. Boost enables to create python data structures in C++. Therefore, library methods can return data compatible with python environment.

4.4 Implementation Details - Traceability Attacks

As mentioned in chapter 3, there are two main attacks used to quantify the untraceability guarantee in Monero proposed by Amrit Kumar et al. [17].

4.4.1 Attack I: Leveraging Zero Mix-ins

Attack I has implemented using MoneroSci tool. This attack has two main phases. The first phase is finding the percentage of zero mix-ins in the Monero blockchain. The second phase is finding the cascade effect of zero mix-ins for the other mix-in strategies which reveal the actual key being spent in the transaction. Finally get the total of zero mixin percentage and percentage of cascade effect of zero mix-ins.

Following code, segment illustrate the way of finding zero mixin percentage using MoneroSci analysis environment. Between block 1 to block 1240503.

```
import monerosci
from __future__ import division

chain = monerosci.Blockchain("location to CBD")

mixin_zero_inputs = chain.inputs_range_mixins(0,1,1240503)
count_mixin_zero_inputs = len(mixin_zero_inputs)
total_inputs = chain.key_image_range_total(1,1240503)

zero_mixin_percentage=(count_mixin_inputs*100)/
    total_inputs
print("Zero mixin percentage-%f" %zero_mixin_percentage)+"
    %"
```

To calculate the cascade effect of zero mix-ins its need to get other mix-in inputs. In [17], it only finds the results up to 10 mix-in inputs. Retrieve inputs which include 1 mixin to 10 mixin from MoneroSci as follows,

```
import monerosci
chain = monerosci.Blockchain("location to CBD")

all_inputs = []
for x in range(10):
    mixin_inputs = chain.inputs_range_mixins((x+1)
  ,1,1240503)
    all_inputs.insert(x,mixin_inputs)
    print str(x+1)+"-mixin input count-"+ str(len(
  mixin_inputs))
```

Implementation of finding the cascade effect of zero mix-ins to other mixin levels as follows,

```
def find_cascade_effect_mixins():
 p = 1-5 loop number
 #iteration of 1-10 mixin inputs
 for x in range(len(not_traceable_inputs[p])):
  current_input_set = not_traceable_inputs[p][x]
  #print len(current_input_set)
  current_not_traceable_inputs = []
  traceable_count = 0

  #selected mixin-value public-key iteration
  for y in range(len(current_input_set)):
   get_all_ring_members = current_input_set[y].rings()
   public_keys = []

   for z in range(len(get_all_ring_members)):
    current_pub_key = get_all_ring_members[z].public_key
    if current_pub_key in traceable_inputs:
      public_keys.insert(z, 1)
    else:
      public_keys.insert(z, 0)
      sum_public_keys = sum(public_keys)
   if sum_public_keys == (x+1):
    spent_key_index = public_keys.index(0)
    traceable_inputs
    [get_all_ring_members[spent_key_index].public_key]=z
    traceable_count = traceable_count + 1
   else:
    current_not_traceable_inputs.append
    (current_input_set[y])

  current_loop_not_traceable_set.insert
  (x,current_not_traceable_inputs)
  traceable_counts[x][p] = traceable_count
  current_percentage = 0

  if (p==0):
   current_percentage = (traceable_count*100/
                     len(not_traceable_inputs[0][x]))
   overall_percentage[x][p] = current_percentage
  else:
   current_percentage = overall_percentage[x][p-1] +
     (traceable_count*100/len(not_traceable_inputs[0][x]))
            overall_percentage[x][p] = current_percentage

  print "Traceable percentage for"+str(x+1)+"-mixin is-"
             +str(current_percentage)+"%"

  not_traceable_inputs.insert((p+1),not_traceable_set)
return overall_percentage
```

As shown in above, attack II loop 5 times to find the cascade effect of zero mix-ins with other mix-in values. In each iteration it prints the traceability percentage in each mix-in value.

4.4.2 Attack II: Leveraging Output Merging

Attack II functions under a heuristic of while creating a transaction, it is unlikely to choose more than one mix-ins that are outputs of a single previous transaction. The simple meaning of that is if there is a transaction used two or more outputs of same previous transaction as its inputs, those inputs can not be act as mix-ins. Therefore, the main objective of this attack is to find the set of transactions which are connected with more than one output as inputs. To move forward with the attack, it needs to retrieve all the outputs and inputs using MoneroSci.

```
def get_inputs_outputs():
    all_inputs = chain.all_inputs(1,1240503)
    all_outputs = chain.all_outputs(1,1240503)

    return all_inputs,all_outputs
```

Here all the inputs and all the outputs have bind with the public key which can use to link inputs and outputs together. Therefore, next step is for each output txes, find txes which are used outputs of a particular tx as its input. From this method, can construct a dictionary including output tx id as its key and list of used input txes of each output as the value. After that can use Counter library to count the number of inputs in the value section of the dictionary. After that can use this dictionary to find inputs and outputs which linked more than once to obtained the results of heuristic II.

```python
from collections import Counter

def heuristic_attack():
    total_txes = 0
    source_to_destination = dict()
    sourceTx_to_destinationTx = dict()
    destination_counts = dict()

    for key in texes_combination:
        currentList = Counter(texes_combination[key])

        more_than_two_destinations = 0
        hasDestination = False
        valid_tx = []
        current_destinations = dict()

        #check the inputs with Counter which has more than 2
        for x in currentList:
            if(currentList[x] >=2):
                hasDestination = True
                more_than_two_destinations =
              more_than_two_destinations + 1
                valid_tx.append(x)
                current_destinations[x] = currentList[x]

        #check output has a destination
        if(hasDestination):
            total_txes = total_txes + 1
            destination_counts[key] = current_destinations

        if(len(valid_tx) > 0):
            sourceTx_to_destinationTx[key] = valid_tx

        #calculate source and destination transaction counts
```

```
      if more_than_two_destinations in source_to_destination:
         source_to_destination[more_than_two_destinations]=
   source_to_destination[more_than_two_destinations]+1
      elif more_than_two_destinations > 0:
         source_to_destination[more_than_two_destinations]=1

#return dictionary of source and destination txes
#source as the key of the dictionary
#destinations list as the value of the dictionary
return source_to_destination
```

4.4.3 Extend Attack II for Unlinkability

Attack II is based on a heuristic which can use to analyze about the linkability of two transactions in Monero. In the attack II it only concerned about to find the real key being spent within the input transaction. But when thinking more about this situation, attack II can extend to track transaction links. To do that, extended attack used source and destination transaction dictionary as for it input data. Then attack decomposed the source and destination dictionary into two main datasets. One for RingCT transaction and another for non RingCT transactions. Then attack generate separate graph data structures for both type to do analysis on it. Due to large number of nodes and relationship it is hard to do it with normal graph databases such as Neo4j. Therefore used Gephi graph tool and Networkx python library to generate graphs via CSV files.

```
import csv

def generateGraphCSV(sourceTx_to_destinationTx):
  nodes = []
  relationships = []
  current_nodes = dict()
  increment_id = 0
  nodes.append(["ID","Label"])
  relationships.append(["Source","Target","Timestamp"])

  for key in sourceTx_to_destinationTx:
      current_key_id = 0
```

```
        current_key = []
        if key in current_nodes:
            current_key_id = current_nodes[key]
        else:
            increment_id += 1
            current_nodes[key] = increment_id
            current_key_id = increment_id

            #check if the tx is in RingCT
            if key in ringct_txes:
                nodes.append([increment_id, "RINGCT"])
            else:
                nodes.append([increment_id, "NON-RINGCT"])

        #get values of output tx
        values = sourceTx_to_destinationTx[key]
        for value in values:
            current_val_id = 0

            if value in current_nodes:
                current_val_id = current_nodes[value]
            else:
                increment_id += 1
                current_nodes[value] = increment_id
                current_val_id = increment_id

                #check if the tx is in RingCT
                if key in ringct_txes:
                    nodes.append([increment_id, "RINGCT"])
                else:
                    nodes.append([increment_id, "NON-RINGCT"
    ])

            #add current relationship
            relationships.append([current_key_id,
    current_val_id,txes_timestamps[current_key_id]])

#generate CSV files
with open("nodes.csv", "wb") as f:
    writer = csv.writer(f)
    writer.writerows(nodes)

with open("edges_timestamp.csv", "wb") as f:
    writer = csv.writer(f)
    writer.writerows(relationships)
```

Based on the generated graphs by the extended attack can analyze the linkability of Monero transactions. And can quantify how much of non-RingCT and RingCT transaction can be linked by the extended attack. Also can identify patterns and reactions of linked transactions over the time.

4.5 Summary

In this chapter, describes the implementation details of the optimized environment to analyze and explore the Monero blockchain called MoneroSci and attacks for quantifying untraceability and unlinkability guarantee of Monero blockchain. Implementation of MoneroSci tool includes details of the main three components. MoneroSci parser, Core Blockchain Data and MoneroSci Analysis Library with details of software tools used and code segments. Evaluation model of this implementations will be described in Chapter 5. More details about the methods and attributes of MoneroSci Analysis Library classes are attached in Appendix A, figure A.1-A.9. It include the implemented documentation for the MoneroSci which can use as a analysis tool for any of interest user.

Chapter 5

Results and Evaluation

5.1 Introduction

This chapter describes how results are evaluated and the success level of the proposed analysis tool and traceability and linkability attacks. As mentioned in the research methodology, there are two phases of evaluations in this research. The first phase is evaluating MoneroSci tool which is an optimized environment for exploring and analyzing the Monero blockchain. The second evaluation phase is based on the existing research untraceability attacks and its results presented by Amrit Kumar et al. [17] for the Monero blockchain developed on top of MoneroSci.

5.2 Evaluation Model

5.2.1 Performance of MoneroSci Parser

MoneroSci parser uses to parse the whole Monero blockchain data into Core Blockchain Data. Parsing the blockchain requires to be done only one time upon the configuration of the MoneroSci. The parser has been configured and tested on 8 vCPUs, 52 GB memory VM instance in Google Cloud Platform. Here 8 vCPU means only have 8 cores. This configured instance cost in GCP is $295.20 per month and $0.404 per hour. If it possible to increase the number of CPUs in the instance can increase the performance of the parser. But it increases the cost of the instance too. In the first stage of evaluation, it evaluates the performances of MoneroSci parser attributes such as parsing speed(time), consumption of virtual memory, consumption of physical memory and CPU utilization. For the moment, MoneroSci parser is the only parser for the Monero blockchain. Therefore performance has been illustrated as benchmark data.

5.2.2 Performance of MoneroSci Analysis Library

To access the Core Blockchain Data it needs the MoneroSci Analysis Library implemented using python and Boost. As mentioned in implementation section, MoneroSci Analysis Library includes many of syntax to access the Monero blockchain data. Such for access block data, transaction data, input data, output data and mixin data. As the part of the evaluation it considered the access time and the accuracy of retrieved data by MoneroSci Analysis Library. MoneroSci Analysis Library has been compared with the existing Monero data retrieving tool as part of the evaluation. And also test cases have run to validate the data retrieve from Analysis Library. Here access time means how much of time it required to retrieve the whole blockchain data.

5.2.3 Monero Traceability and Linkability Guarantee

In the second phase of the evaluation, Existing untraceability attacks have been evaluated. Attack I and attack II in [17] have been developed on top of the MoneroSci tool. Those attack results have compared and evaluated with past researcher's results. Here attack I results have been used as the ground truth data for the attack II because attack I and attack II both are developed to quantify the guarantee of Monero traceability. All the results compared as the percentages. To evaluate the attack II results accuracy, following terms are defined same as in [17].

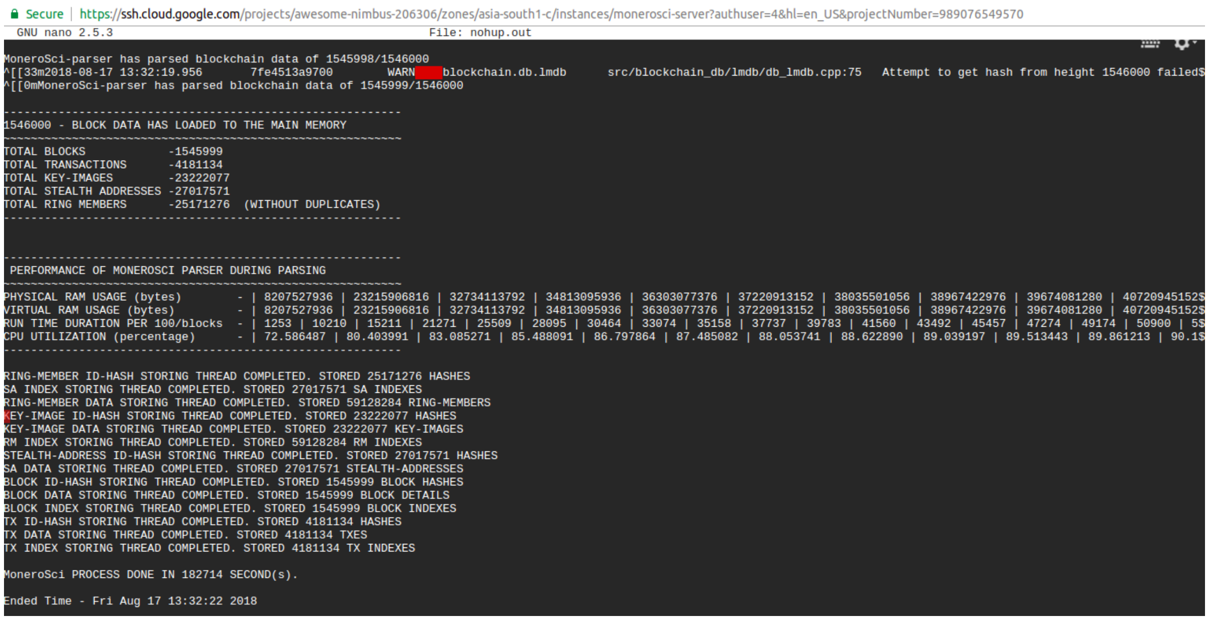
- **True positive (TP):** An input creates a true positive if: Attack II identifies a unique key as the one being spent in the input and, the key is the same as the one identified by Attack I. The two attacks are hence in agreement with the conclusion
- **False positive (FP):** An input creates a false positive if all the keys identified as being spent by Attack II were actually found to be spent in a different.
- **Unknown positive (UP):** An input creates an unknown positive if at least one of the keys identified by Attack II could not be identified as being spent in any input (of any transaction) by Attack I. The uncertainty comes from Attack I as it does not give ground truth for all inputs.

For the extended unlinkability attack for the Monero has been evaluated by generating a graph and this is the first study of research has been done for Monero linkability. Non-RingCT transactions have been verified using the attack I ground truth data.

5.3 Results

5.3.1 Performance of MoneroSci Parser

MoneroSci parser connects with Monero daemon to retrieve the raw Monero blockchain data into the local machine. Monero daemon stores blockchain data in the lmdb data files inside the .bitmonero directory. Monero Daemon takes several hours to download the blockchain data. Due to this reason initialization step is getting slow. But this is a one-time requirement. Then MoneroSci parser connected with lmdb file and parsed the raw blockchain data into the Core Blockchain Data. As mentioned in Evaluation model, MoneroSci parser has been configured in 8 vCPUs, 52 GB memory VM instance in Google Cloud Platform. It resulted in the runtime of around 50 hours (182714 seconds) to complete the whole Monero blockchain (up to 1546000 blocks). Following figure illustrate the final outputs preview of the MoneroSci parser.


Secure | https://ssh.cloud.google.com/projects/awesome-nimbus-206306/zones/asia-south1-c/instances/monerosci-server?authuser=4&hl=en_US&projectNumber=989076549570

```
  GNU nano 2.5.3                                        File: nohup.out

MoneroSci-parser has parsed blockchain data of 1545998/1546000
^[[33m2018-08-17 13:32:19.956       7fe4513a9700        WARN     blockchain.db.lmdb      src/blockchain_db/lmdb/db_lmdb.cpp:75   Attempt to get hash from height 1546000 failed$
^[[0mMoneroSci-parser has parsed blockchain data of 1545999/1546000

-----------------------------------------------------------
1546000 - BLOCK DATA HAS LOADED TO THE MAIN MEMORY
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
TOTAL BLOCKS            -1545999
TOTAL TRANSACTIONS      -4181134
TOTAL KEY-IMAGES        -23222077
TOTAL STEALTH ADDRESSES -27017571
TOTAL RING MEMBERS      -25171276  (WITHOUT DUPLICATES)
-----------------------------------------------------------


-----------------------------------------------------------
 PERFORMANCE OF MONEROSCI PARSER DURING PARSING
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
PHYSICAL RAM USAGE (bytes)        - | 8207527936 | 23215906816 | 32734113792 | 34813095936 | 36303077376 | 37220913152 | 38035501056 | 38967422976 | 39674081280 | 40720945152$
VIRTUAL RAM USAGE (bytes)         - | 8207527936 | 23215906816 | 32734113792 | 34813095936 | 36303077376 | 37220913152 | 38035501056 | 38967422976 | 39674081280 | 40720945152$
RUN TIME DURATION PER 100/blocks  - | 1253 | 10210 | 15211 | 21271 | 25509 | 28095 | 30464 | 33074 | 35158 | 37737 | 39783 | 41560 | 43492 | 45457 | 47274 | 49174 | 50900 | 5$
CPU UTILIZATION (percentage)      - | 72.586487 | 80.403991 | 83.085271 | 85.488091 | 86.797864 | 87.485082 | 88.053741 | 88.622890 | 89.039197 | 89.513443 | 89.861213 | 90.1$
-----------------------------------------------------------

RING-MEMBER ID-HASH STORING THREAD COMPLETED. STORED 25171276 HASHES
SA INDEX STORING THREAD COMPLETED. STORED 27017571 SA INDEXES
RING-MEMBER DATA STORING THREAD COMPLETED. STORED 59128284 RING-MEMBERS
KEY-IMAGE ID-HASH STORING THREAD COMPLETED. STORED 23222077 HASHES
KEY-IMAGE DATA STORING THREAD COMPLETED. STORED 23222077 KEY-IMAGES
RM INDEX STORING THREAD COMPLETED. STORED 59128284 RM INDEXES
STEALTH-ADDRESS ID-HASH STORING THREAD COMPLETED. STORED 27017571 HASHES
SA DATA STORING THREAD COMPLETED. STORED 27017571 STEALTH-ADDRESSES
BLOCK ID-HASH STORING THREAD COMPLETED. STORED 1545999 BLOCK HASHES
BLOCK DATA STORING THREAD COMPLETED. STORED 1545999 BLOCK DETAILS
BLOCK INDEX STORING THREAD COMPLETED. STORED 1545999 BLOCK INDEXES
TX ID-HASH STORING THREAD COMPLETED. STORED 4181134 HASHES
TX DATA STORING THREAD COMPLETED. STORED 4181134 TXES
TX INDEX STORING THREAD COMPLETED. STORED 4181134 TX INDEXES

MoneroSci PROCESS DONE IN 182714 SECOND(s).

Ended Time - Fri Aug 17 13:32:22 2018
```

Figure 5.1: Output of the MoneroSci parser

As results of the MoneroSci parser, it parsed entire history from the first transaction on April 18th 2014 up to April 06th 2018 including 1,546,000 blocks, transactions, key images, stealth addresses and ring members. Table 5.1 shows the details of parsed blockchain data counts.

Table 5.1: Parsed Blockchain Data Counts

Blockchain Data	**Count**
Blocks	1,546,000
Transactions	4,181,134
Stealth Addresses (outputs)	27,017,571
Key-Images (Inputs)	23,222,077
Ring Members (mixins)	40,389,128

Parsing the blockchain needs to be done only one time upon installation. Parser performance depends on the number of cores available in the server. While parsing, MoneroSci parser able to track the performance it using. Observed MoneroSci parser performance in consumption of physical memory, CPU utilization and speed illustrate on following Table 5.2. These values illustrate that how much of performance required when the parser parsing 100,000 of block ranges.

Table 5.2: Performance evaluation of MoneroSci Parser

Block Number	**Physical Memory Usage (Gega Byte)**	**CPU Utilization (%)**	**Time (seconds)**
100,000	21.621	80.403991	10210
200,000	32.422	85.488091	21271
300,000	34.665	87.485082	28095
400,000	36.291	88.622890	33074
500,000	37.924	89.513443	37737
600,000	39.023	90.144637	41560
700,000	40.189	90.713677	45457
800,000	41.421	91.197647	49174
900,000	42.491	91.626652	52833
1,000,000	43.897	92.153302	57878
1,100,000	46.999	93.139972	69430
1,200,000	45.326	94.304827	88404

Continued on next page...

Table 5.2: continued from previous page.

Block Number	**Physical Memory Usage (Gega Byte)**	**CPU Utilization (%)**	**Time (seconds)**
1,300,000	47.750	95.044464	105300
1,400,000	50.699	96.021972	137297
1,500,000	50.740	96.700470	171386

Following figures illustrate the monitored performance of MoneroSci Parser using above performance values.

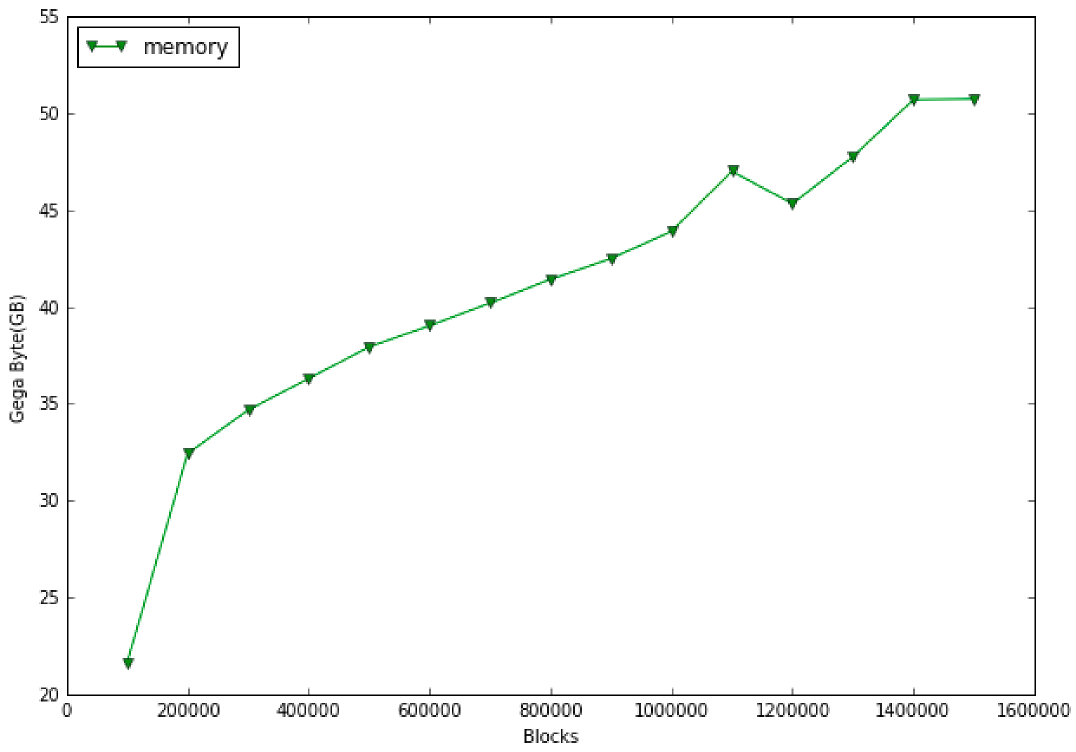


Figure 5.2: Memory usage of MoneroSci Parser

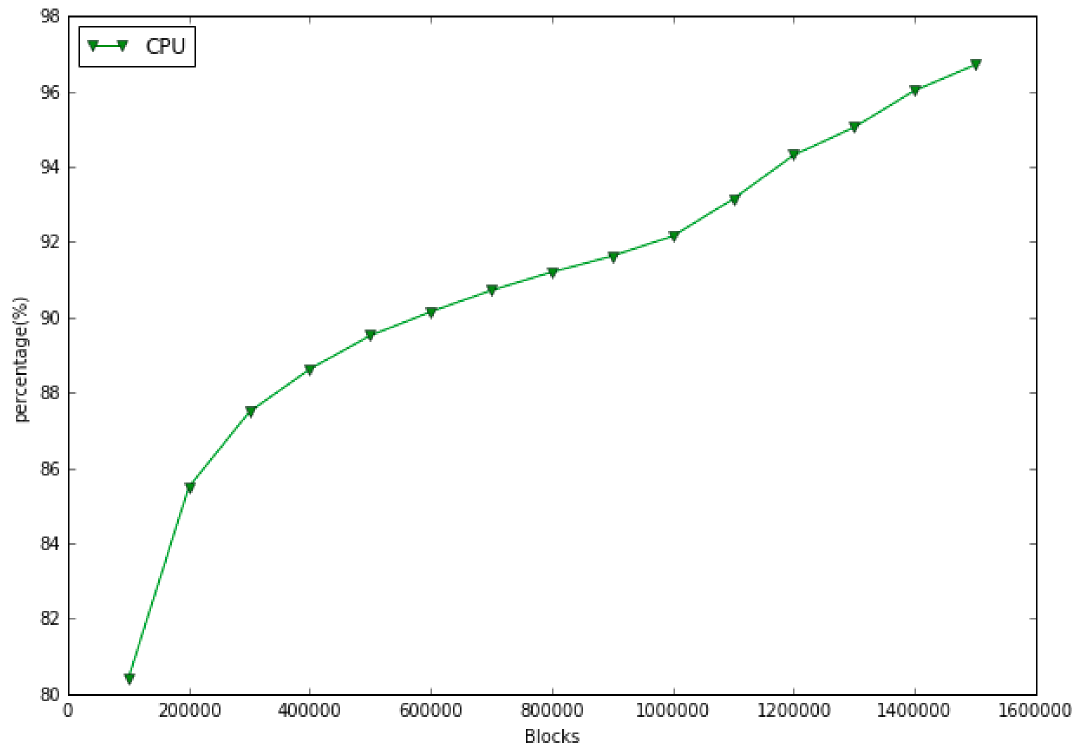


Figure 5.3: CPU Utilization of MoneroSci Parser

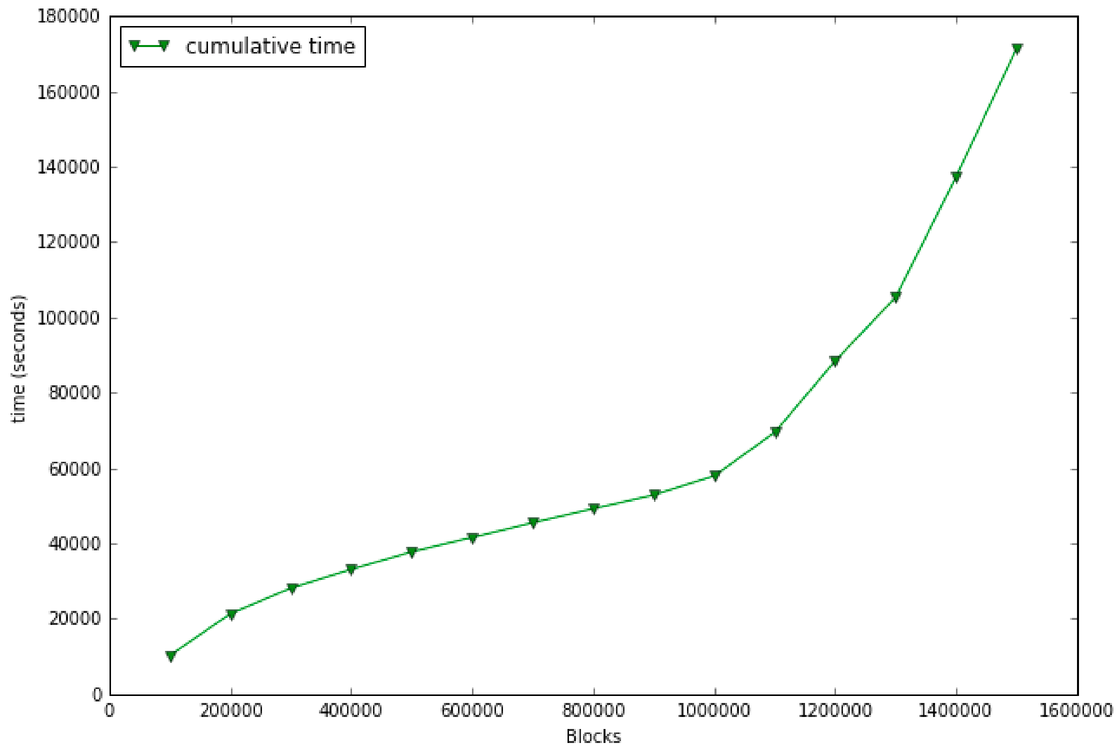


Figure 5.4: Block parsing time of MoneroSci Parser

5.3.2 Performance of MoneroSci Analysis Library

Comparing the MoneroSci analysis tool with other existing Monero data explorers, MoneroSci takes a very low amount of time to retrieve the whole blocks data from the blockchain. Following Figure 5.5 and 5.6 illustrate the block access time of the existing tool vs MoneroSci tool.

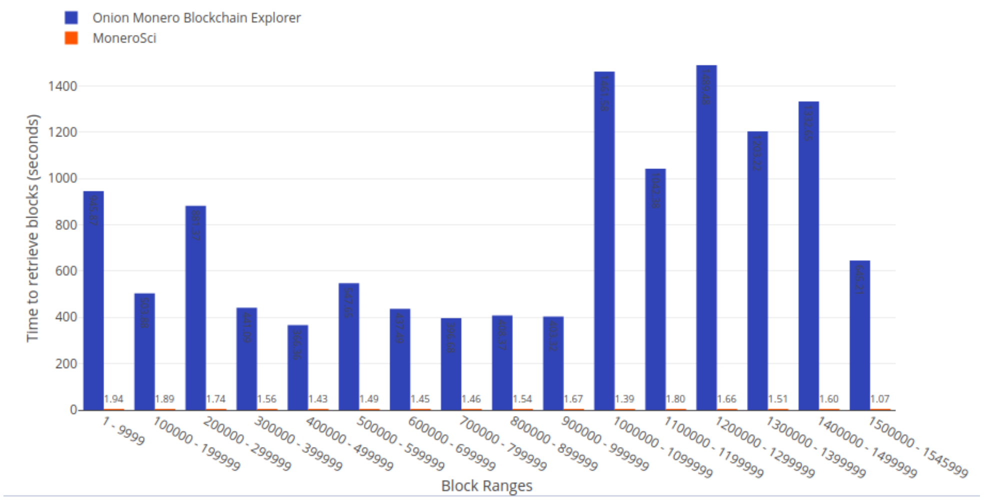


Figure 5.5: MoneroSci vs Onion Monero Blockchain Explorer block ranges access time

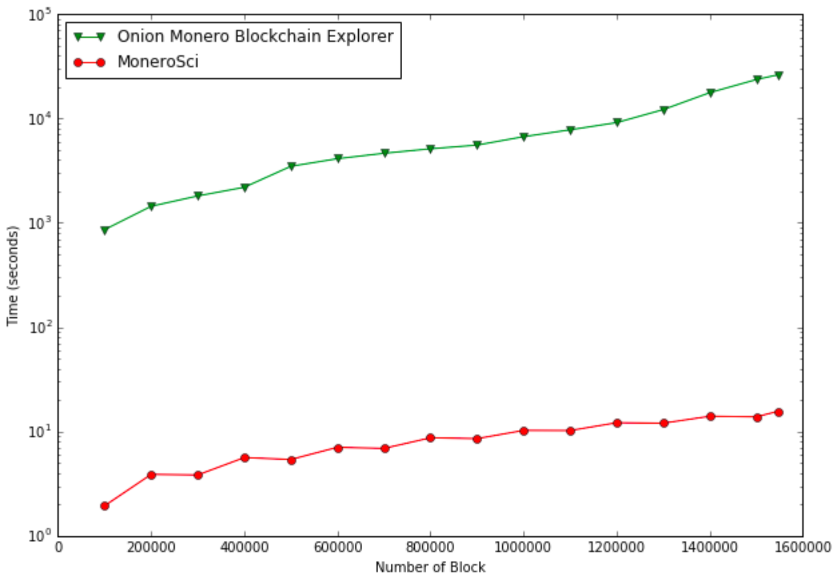
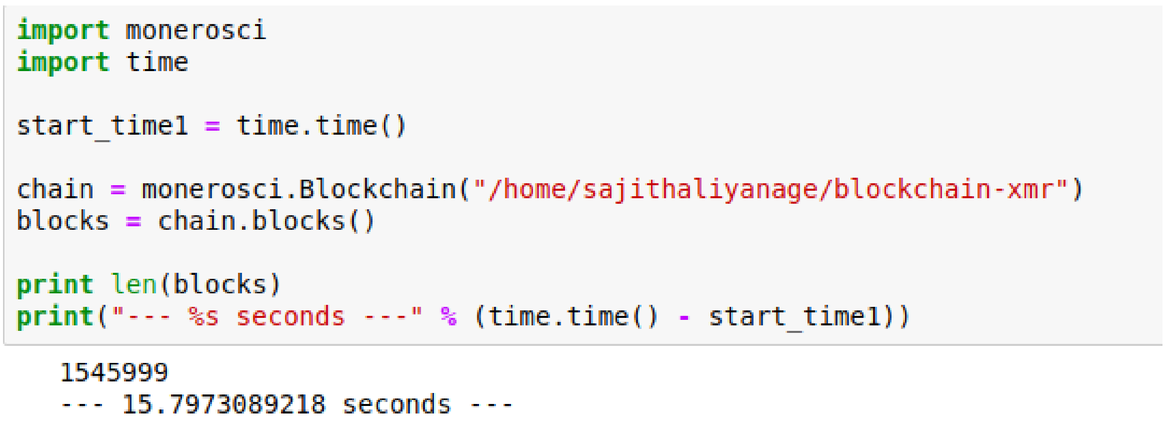


Figure 5.6: MoneroSci vs Onion Monero Blockchain Explorer cumulative block access time

Above diagrams show that existing Monero explore tool (Onion Monero Blockchain Explorer) requires more than 400 seconds to retrieve 100,000 blocks in any block range. But every 100,000 block ranges MoneroSci tool only requires less than 2 seconds to retrieve 100,000 blocks. Considering the overall access times, Onion Monero Blockchain Explorer needs 25604 seconds (around 7.1 hours) to retrieve whole blockchain and MoneroSci only requires 15 seconds to retrieve whole blockchain. In figure 5.5 clearly indicates that after 1 million blocks Onion Monero Blockchain Explorer requires more than 1100 seconds to retrieve data than previously block ranges. Main reason for this is Monero developers have maximized the block size after 1009827 block to force the transaction minimum mix-in value as 3 (hard fork). In data analyzing scenarios, loading blockchain data in faster way is very important because blockchain contains a huge number of data. Therefore, analyzing tool needs to perform very fast. Following Figure 5.7 illustrates the code segment and it result of how the MoneroSci performs to retrieve all the blocks in Monero blockchain.



```
import monerosci
import time

start_time1 = time.time()

chain = monerosci.Blockchain("/home/sajithaliyanage/blockchain-xmr")
blocks = chain.blocks()

print len(blocks)
print("--- %s seconds ---" % (time.time() - start_time1))
```

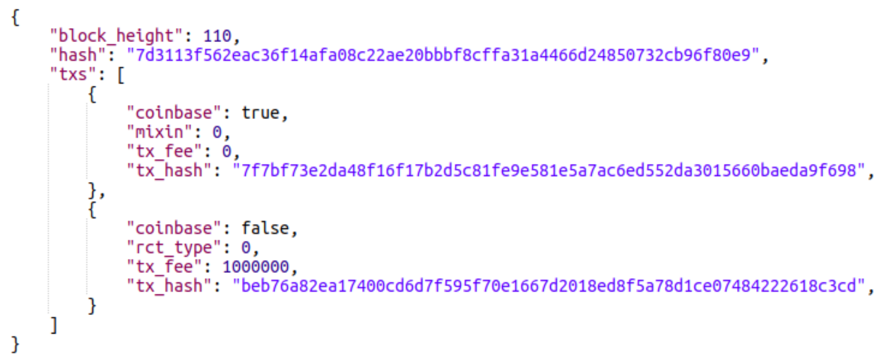
```
1545999
--- 15.7973089218 seconds ---
```

Figure 5.7: Retrieving all block data by MoneroSci

To validate the data of the MoneroSci Analysis Library it has been tested and evaluated with Onion Monero Blockchain Explorer existing tool. Following are the some of test cases used to compare and verify the reliability of blockchain data between the two tools.

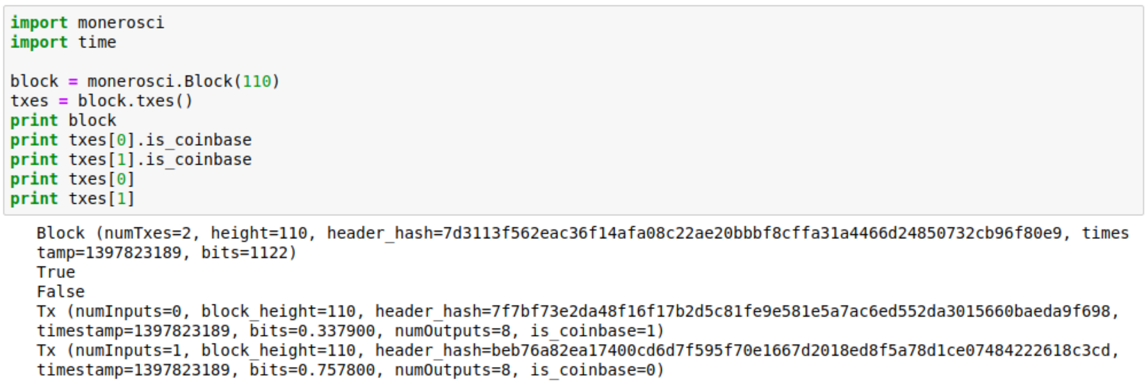
Test Case I - Retrieving Block Data

Retrieving Block 110 data using Onion Monero Blockchain Explorer



```
{
    "block_height": 110,
    "hash": "7d3113f562eac36f14afa08c22ae20bbbf8cffa31a4466d24850732cb96f80e9",
    "txs": [
        {
            "coinbase": true,
            "mixin": 0,
            "tx_fee": 0,
            "tx_hash": "7f7bf73e2da48f16f17b2d5c81fe9e581e5a7ac6ed552da3015660baeda9f698",
        },
        {
            "coinbase": false,
            "rct_type": 0,
            "tx_fee": 1000000,
            "tx_hash": "beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd",
        }
    ]
}
```

Retrieving Block 110 data using MoneroSci



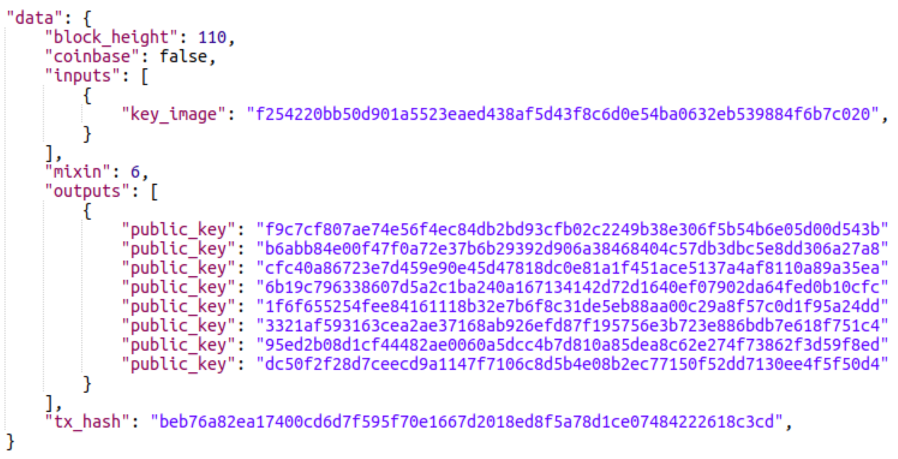
```
import monerosci
import time

block = monerosci.Block(110)
txes = block.txes()
print block
print txes[0].is_coinbase
print txes[1].is_coinbase
print txes[0]
print txes[1]
```

```
Block (numTxes=2, height=110, header_hash=7d3113f562eac36f14afa08c22ae20bbbf8cffa31a4466d24850732cb96f80e9, times
tamp=1397823189, bits=1122)
True
False
Tx (numInputs=0, block_height=110, header_hash=7f7bf73e2da48f16f17b2d5c81fe9e581e5a7ac6ed552da3015660baeda9f698,
timestamp=1397823189, bits=0.337900, numOutputs=8, is_coinbase=1)
Tx (numInputs=1, block_height=110, header_hash=beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd,
timestamp=1397823189, bits=0.757800, numOutputs=8, is_coinbase=0)
```

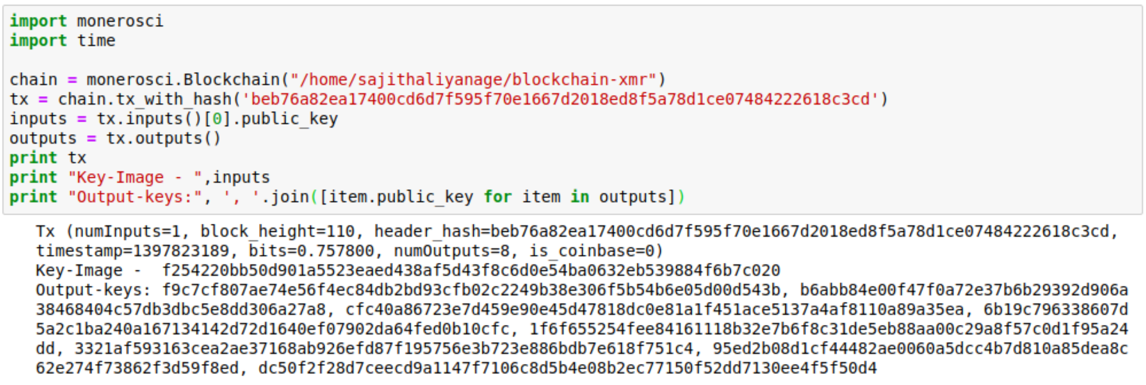
Test Case II - Retrieving Transaction Data

Retrieving Transaction with hash ”beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd” data using Onion Monero Blockchain Explorer



```
"data": {
    "block_height": 110,
    "coinbase": false,
    "inputs": [
        {
            "key_image": "f254220bb50d901a5523eaed438af5d43f8c6d0e54ba0632eb539884f6b7c020",
        }
    ],
    "mixin": 6,
    "outputs": [
        {
            "public_key": "f9c7cf807ae74e56f4ec84db2bd93cfb02c2249b38e306f5b54b6e05d00d543b"
            "public_key": "b6abb84e00f47f0a72e37b6b29392d906a38468404c57db3dbc5e8dd306a27a8"
            "public_key": "cfc40a86723e7d459e90e45d47818dc0e81a1f451ace5137a4af8110a89a35ea"
            "public_key": "6b19c796338607d5a2c1ba240a167134142d72d1640ef07902da64fed0b10cfc"
            "public_key": "1f6f655254fee84161118b32e7b6f8c31de5eb88aa00c29a8f57c0d1f95a24dd"
            "public_key": "3321af593163cea2ae37168ab926efd87f195756e3b723e886bdb7e618f751c4"
            "public_key": "95ed2b08d1cf44482ae0060a5dcc4b7d810a85dea8c62e274f73862f3d59f8ed"
            "public_key": "dc50f2f28d7ceecd9a1147f7106c8d5b4e08b2ec77150f52dd7130ee4f5f50d4"
        }
    ],
    "tx_hash": "beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd",
}
```

Retrieving Transaction with hash
"beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd" data using MoneroSci



```
import monerosci
import time

chain = monerosci.Blockchain("/home/sajithaliyanage/blockchain-xmr")
tx = chain.tx_with_hash('beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd')
inputs = tx.inputs()[0].public_key
outputs = tx.outputs()
print tx
print "Key-Image - ",inputs
print "Output-keys:", ', '.join([item.public_key for item in outputs])
```

```
Tx (numInputs=1, block_height=110, header_hash=beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd,
timestamp=1397823189, bits=0.757800, numOutputs=8, is_coinbase=0)
Key-Image -  f254220bb50d901a5523eaed438af5d43f8c6d0e54ba0632eb539884f6b7c020
Output-keys: f9c7cf807ae74e56f4ec84db2bd93cfb02c2249b38e306f5b54b6e05d00d543b, b6abb84e00f47f0a72e37b6b29392d906a
38468404c57db3dbc5e8dd306a27a8, cfc40a86723e7d459e90e45d47818dc0e81a1f451ace5137a4af8110a89a35ea, 6b19c796338607d
5a2c1ba240a167134142d72d1640ef07902da64fed0b10cfc, 1f6f655254fee84161118b32e7b6f8c31de5eb88aa00c29a8f57c0d1f95a24
dd, 3321af593163cea2ae37168ab926efd87f195756e3b723e886bdb7e618f751c4, 95ed2b08d1cf44482ae0060a5dcc4b7d810a85dea8c
62e274f73862f3d59f8ed, dc50f2f28d7ceecd9a1147f7106c8d5b4e08b2ec77150f52dd7130ee4f5f50d4
```

In above both test cases MoneroSci perform 100% accurately. MoneroSci retrieved true data without retrieving false data for every test cases.

5.3.3 Monero Traceability with Mix-ins

In Monero, traceability is directly link with the mix-in strategy of the transactions. Therefore how transactions used mix-in strategy has been evaluated using MoneroSci tool. Figure 5.8 shows the how many transactions used each mix-in values in Monero up to 10.

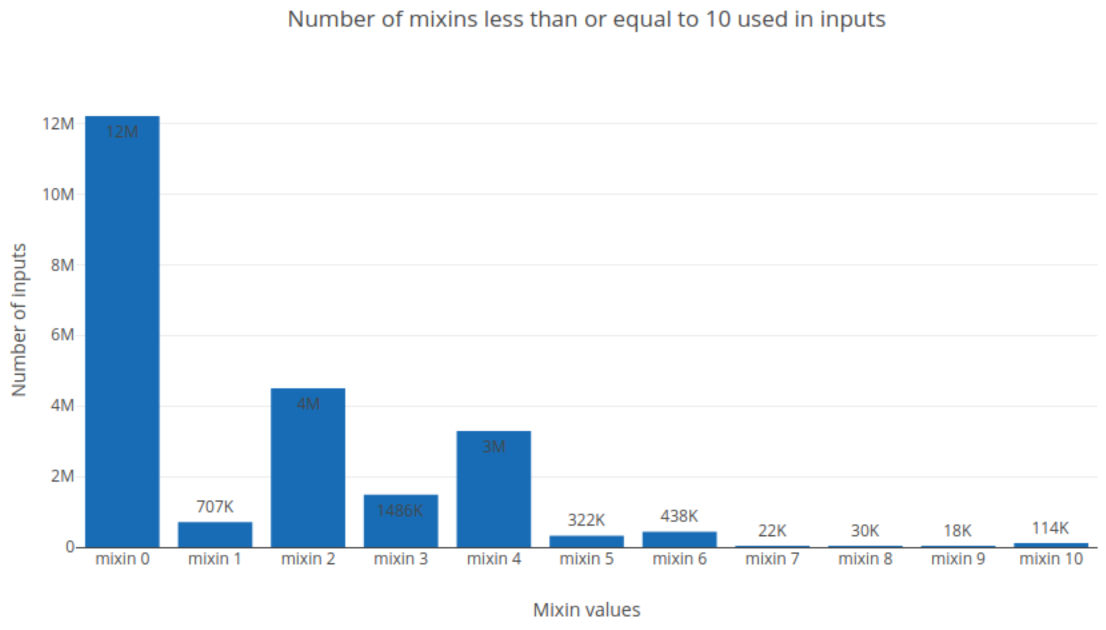


Figure 5.8: Monero mix-ins with frequencies

Based on the above illustration, can construct cumulative frequency table for mix-ins values in Monero blockchain.

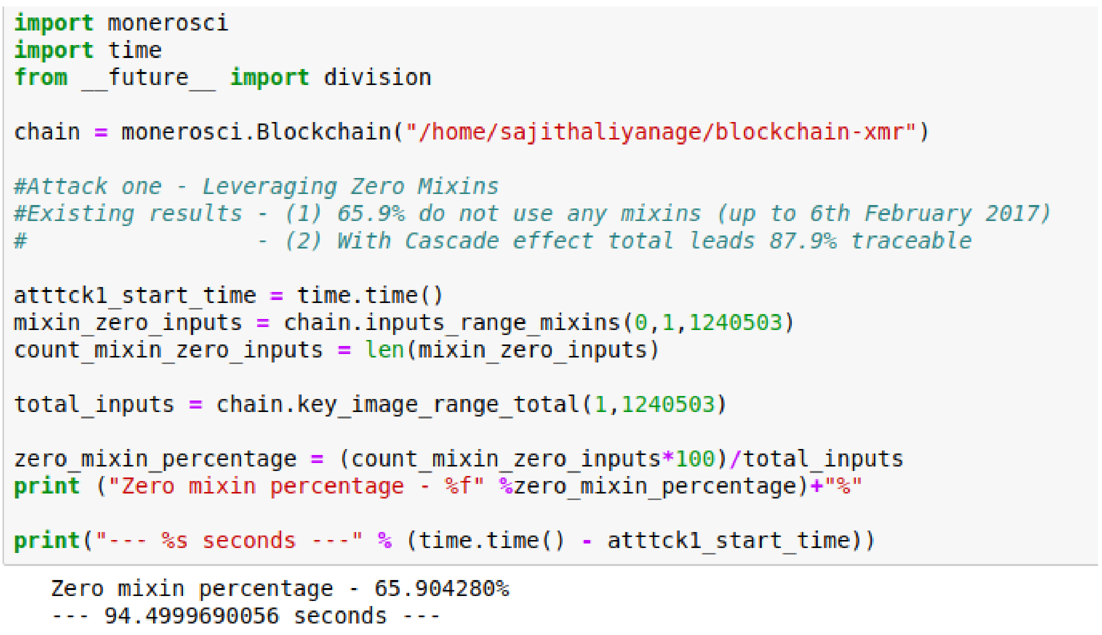
Table 5.3: Monero mix-ins with cumulative frequencies

Mix-in Number	**Frequency (inputs)**	**Cumulative Frequency (in %)**
0	12209849	52.57
1	707788	55.62
2	4496449	74.98
3	1486633	81.39
4	3287350	95.54
5	322845	96.93
6	438574	98.82
7	22147	98.92
8	30385	99.05
9	18259	99.13
10	114672	99.62

From the above table of figure, can observed that first transaction on April 18th 2014 up to and including the last transaction on April 6th 2018, most of them are used 0 to 10 mixin values. It means 99.62% of transactions used 0 to 10 mixin values within the transaction. The main reason could be for it is the transaction fee. If mixin value increases, it increase the transaction fee. And also more than 50% transactions are used 0 mixin which transactions are trivially traceable.

5.3.4 Traceability Attack I: Leveraging Zero Mix-ins

As mention in chapter 3 and chapter 4, existing traceability attacks conducted by Amrit Kumar et al. [17] and Malte Möser et al. [18] have implemented on top of the MoneroSci. In attack I, Amrit Kumar et al. have used the blockchain range of 1 to 1240503. First transaction on April 18th 2014 up to and including the last transaction on February 6th 2017. In [17], they mentioned that Monero blockchain includes 65.9% of inputs do not use any mix-ins and trivially traceable. It has been verified using MoneroSci by developing that statement on top of it.



```
import monerosci
import time
from __future__ import division

chain = monerosci.Blockchain("/home/sajithaliyanage/blockchain-xmr")

#Attack one - Leveraging Zero Mixins
#Existing results - (1) 65.9% do not use any mixins (up to 6th February 2017)
#                 - (2) With Cascade effect total leads 87.9% traceable

atttck1_start_time = time.time()
mixin_zero_inputs = chain.inputs_range_mixins(0,1,1240503)
count_mixin_zero_inputs = len(mixin_zero_inputs)

total_inputs = chain.key_image_range_total(1,1240503)

zero_mixin_percentage = (count_mixin_zero_inputs*100)/total_inputs
print ("Zero mixin percentage - %f" %zero_mixin_percentage)+"%"

print("--- %s seconds ---" % (time.time() - atttck1_start_time))
```

```
Zero mixin percentage - 65.904280%
--- 94.4999690056 seconds ---
```

Figure 5.9: Results of finding zero mixin percentage

And later part of the attack I, they have mentioned zero mix-in cascades to traceability of another 22% of the inputs, leading to a total of 87% of traceable inputs. Therefore, the cascade effect of the zero mix-ins has been evaluated using MoneroSci. Attack used 5 iterations(n) as they done in [17]. It gave the results as follows,

Table 5.4: Attack I zero mixin cascade effect in percentage

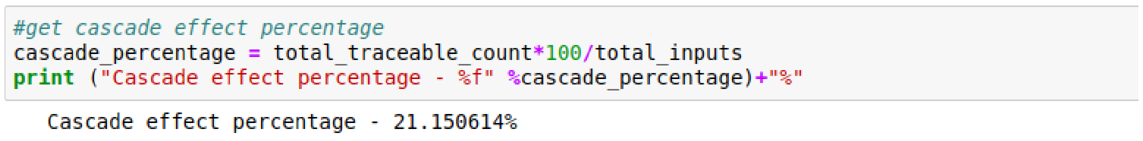
Mixin Number	**Percentage of traceable inputs in nth iteration (in %)**				
	n = 1	**n = 2**	**n = 3**	**n = 4**	**n = 5**
1	82.11	87.03	87.58	87.65	87.66
2	45.67	56.72	58.51	58.80	58.85
3	58.75	67.51	68.82	69.04	69.08
4	42.17	50.37	51.90	52.18	52.23
5	42.33	48.35	49.26	49.38	49.41
6	43.31	50.46	51.58	51.76	51.79
7	30.16	35.77	36.79	36.96	37.01

Continued on next page...

Table 5.4: continued from previous page.

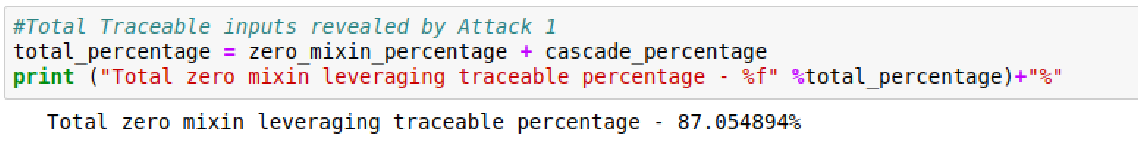
Mixin Number	**Percentage of traceable inputs in nth iteration (in %)**				
	n = 1	**n = 2**	**n = 3**	**n = 4**	**n = 5**
8	27.26	35.00	36.52	36.83	36.90
9	26.61	33.86	35.31	35.64	35.72
10	19.79	25.43	26.53	26.7	26.76

Here can observed that after some iteration level traceable inputs almost reaches a constant value. After first iteration (n=1) the number of traceable inputs of one mix-in reaches the values as 82%. After fifth iteration (n=5) this percentage becomes 87% like Amrit Kumar et al observed. Above table shows that the cascade effect for using high number mix-in value such as 10 also have a considerable percentage of traceable inputs. 27% after fifth iteration. End of the attack, observed that almost 22% of inputs are traceable due to the cascade effect and it leading to a total of 87% of traceable inputs.



```
#get cascade effect percentage
cascade_percentage = total_traceable_count*100/total_inputs
print ("Cascade effect percentage - %f" %cascade_percentage)+"%"
```

```
Cascade effect percentage - 21.150614%
```



```
#Total Traceable inputs revealed by Attack 1
total_percentage = zero_mixin_percentage + cascade_percentage
print ("Total zero mixin leveraging traceable percentage - %f" %total_percentage)+"%"
```

```
Total zero mixin leveraging traceable percentage - 87.054894%
```

5.3.5 Traceability Attack II: Leveraging Output Merging

As discussed in the implementation chapter this attack is based on a heuristic. Based on that heuristic attack observed these results. It found 882,102 different source transactions connects with one or more destination transactions. Here 881,079 source transactions were non-RingCT and 1023 transactions were RingCT. Considering the non-RingCT transactions 75% of all source transactions have only one destination. There were two transaction which had 147 destinations as the maximum number of destinations. Other source transaction destination counts varied between 1 and 147. Following figure illustrate the source and destination counts of transactions found by attack II.

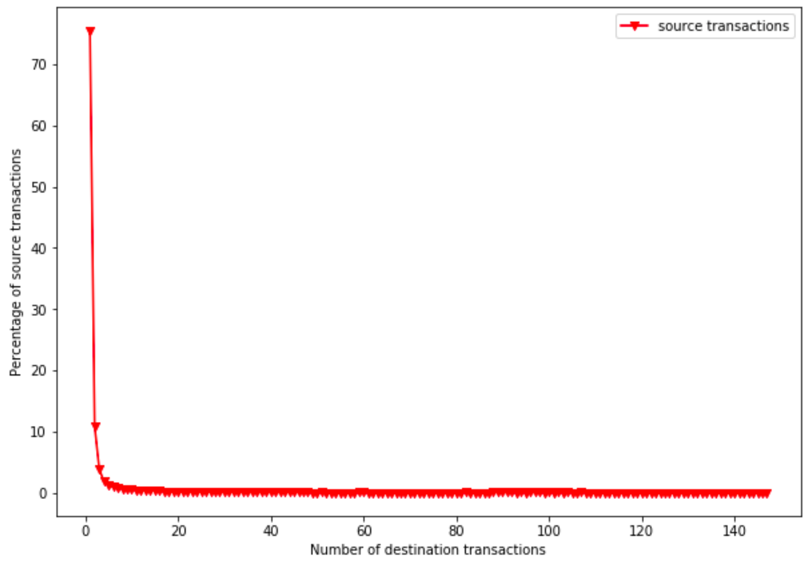


Figure 5.10: Result of employing Attack II on non-RingCTs

To evaluate the attack II, it used the results obtained by attack I that yield as ground truth. As discussed in the evaluation model it calculated the true positive, false positive and unknown positive rates of non-RingCT source transactions. The calculated results shows attack II has an overall true positive rate of 74.6% and 0.83% rate as false positive. 22.2% of unknown positive rate.

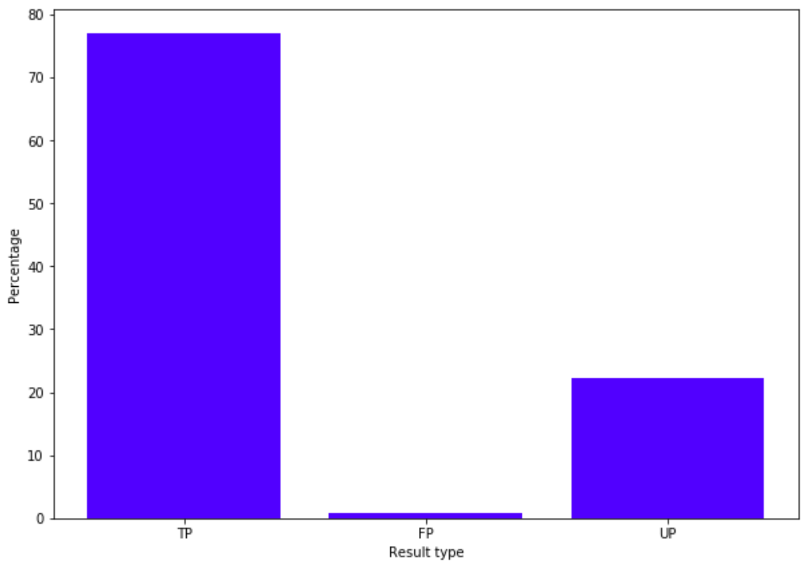
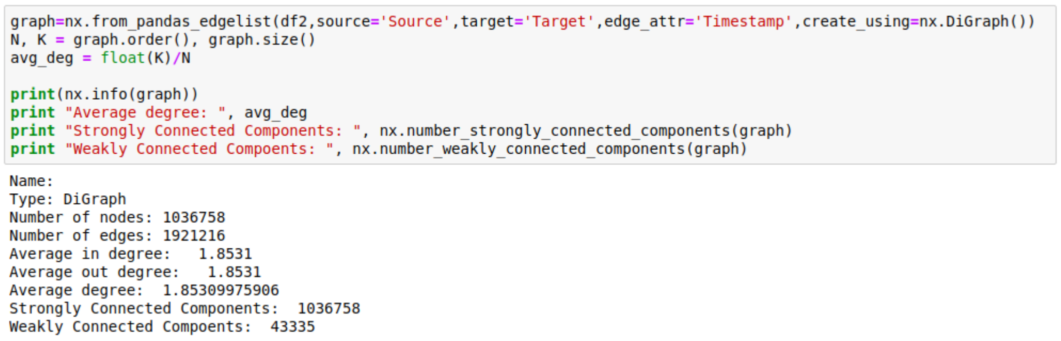


Figure 5.11: Attack II: Overall observed percentage of TP, FP and UP

5.3.6 Extend Attack II for Unlinkability

As discussed in design and implementation sections, this attack is based on the same heuristic used for the attack II: leveraging output merging. This attack was extend to the block height 1546000 which include first transaction on April 18th 2014 up to and including the transaction on April 6th 2018. This attack has been generated two main graphs. One for non-RingCT and one for RingCT. First considering the non-RingCT transaction graph, it is a linked non-RingCT transaction graph with 1036758 nodes and 1921216 edges.



```
graph=nx.from_pandas_edgelist(df2,source='Source',target='Target',edge_attr='Timestamp',create_using=nx.DiGraph())
N, K = graph.order(), graph.size()
avg_deg = float(K)/N

print(nx.info(graph))
print "Average degree: ", avg_deg
print "Strongly Connected Components: ", nx.number_strongly_connected_components(graph)
print "Weakly Connected Compoents: ", nx.number_weakly_connected_components(graph)
```

```
Name:
Type: DiGraph
Number of nodes: 1036758
Number of edges: 1921216
Average in degree:   1.8531
Average out degree:   1.8531
Average degree:  1.85309975906
Strongly Connected Components:  1036758
Weakly Connected Compoents:  43335
```

For the analysis of this graph, created a graph using NetworkX and plotted transactions degree distribution. For the degree distribution it considered the in-degree, out-degree and overall degree counts for each node.

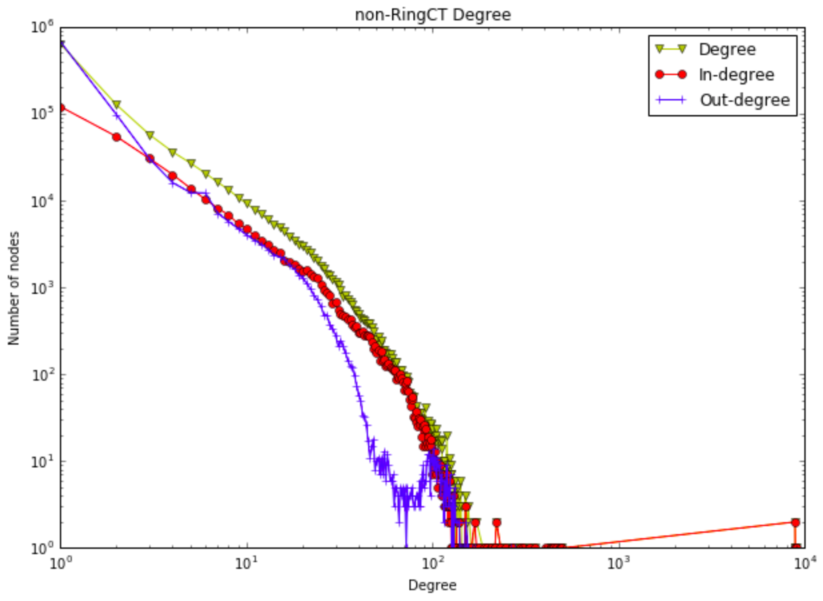
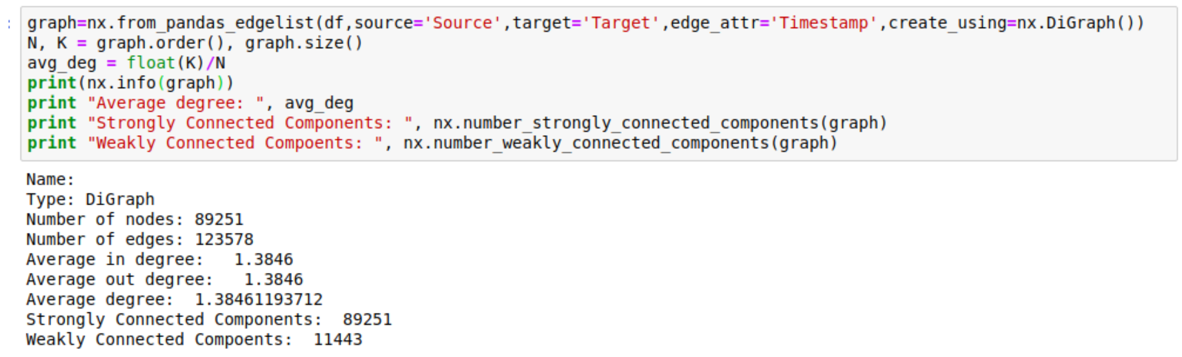


Figure 5.12: Degree distribution of non-RingCT linked graph

Considering the RingCT transaction graph, it is a linked RingCT transaction graph with 89251 nodes and 123578 edges. The one of main observation here is, RingCT transitions are the second level transactions which are now mandatory to use in Monero transactions. But those transactions also give results to this attack by joining with linked transaction graph.



```
graph=nx.from_pandas_edgelist(df,source='Source',target='Target',edge_attr='Timestamp',create_using=nx.DiGraph())
N, K = graph.order(), graph.size()
avg_deg = float(K)/N
print(nx.info(graph))
print "Average degree: ", avg_deg
print "Strongly Connected Components: ", nx.number_strongly_connected_components(graph)
print "Weakly Connected Compoents: ", nx.number_weakly_connected_components(graph)
```

```
Name:
Type: DiGraph
Number of nodes: 89251
Number of edges: 123578
Average in degree:   1.3846
Average out degree:   1.3846
Average degree:  1.38461193712
Strongly Connected Components:  89251
Weakly Connected Compoents:  11443
```

For the analysis of this graph, created a graph using NetworkX and plotted transactions degree distribution. For the degree distribution it considered the in-degree, out-degree and overall degree counts for each node.

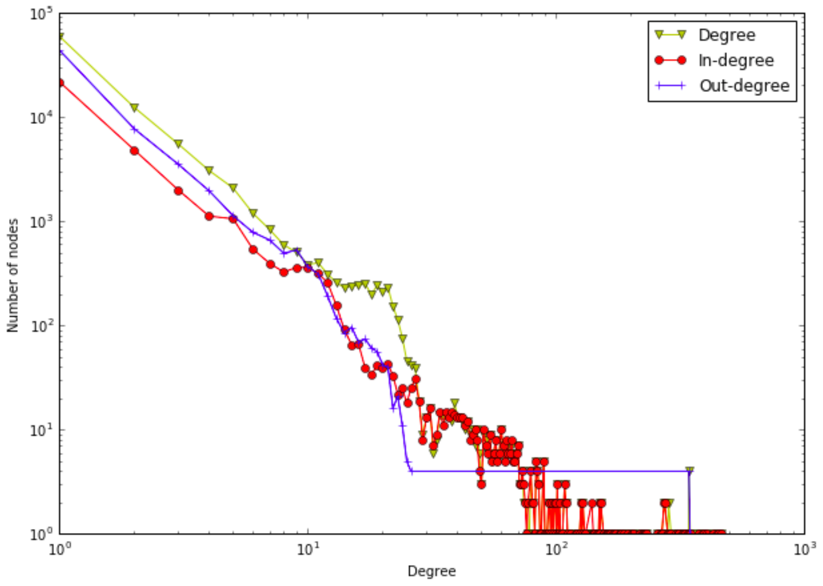


Figure 5.13: Degree distribution of RingCT linked graph

For the further analysis of this RingCT linked transaction graph, checked the linkability over the time. The time series used for the analysis are follows,

- **Time Series 1** - Tuesday, 31-Oct-17 00:00:00 UTC (1509408000)
- **Time Series 2** - Thursday, 30-Nov-17 00:00:00 UTC (1512000000)
- **Time Series 3** - Sunday, 31-Dec-17 00:00:00 UTC (1514678400)
- **Time Series 4** - Wednesday, 31-Jan-18 00:00:00 UTC (1517356800)
- **Time Series 5** - Wednesday, 28-Feb-18 00:00:00 UTC (1519776000)
- **Time Series 6** - Saturday, 31-Mar-18 00:00:00 UTC (1522454400)

For the above time series, the attack has been generated sub linked transaction graphs and analyzed the node-edges changes and degree distribution of nodes. From these plots can be observed that over the time series linked transaction graph edges and nodes count are getting increased and degree distribution is getting decreased. Following figures 5.14 and 5.15 elaborate the results over the time series.

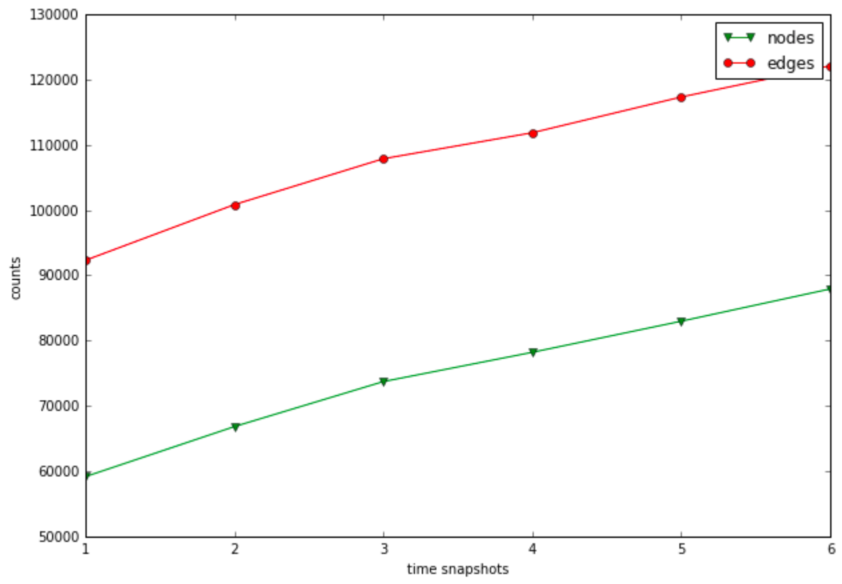


Figure 5.14: Number of nodes and edges in RingCT linked graph over the time.

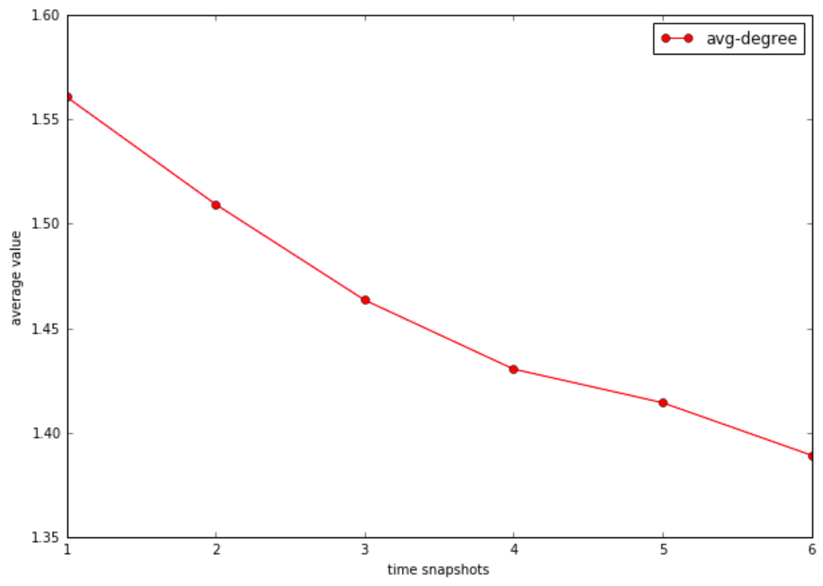


Figure 5.15: Degree distribution in RingCT linked graph over the time.

5.4 Summary

This chapter elaborated on the evaluation model of optimized environment for analyzing and exploring the Monero blockchain called MoneroSci and traceability attacks presented by Amrit Kumar et al. and Malte Möser et al. The results obtained for the MoneroSci showed that the proposed tool performs well over the existing tools. It only needs 15 seconds to load the whole blocks in the blockchain. And MoneroSci verified the existing attack results conducted in [17] as 87% non-RingCT transactions are traceable to attack I and 75% non-RingCT transactions are traceable to attack II. Extended attack shows that both non-RingCT and RingCT transactions can be linked using it and over the time linking nodes and edges are getting increased.

Chapter 6

Conclusions

6.1 Introduction

This chapter includes a conclusion about the research aims and objectives, research problems, limitations of the current work and implications for further research.

6.2 Conclusions about research questions (aims/objectives)

As discussed in research methodology and the design, the main aim of the research was to create an optimized environment to explore and analyze the Monero blockchain called MoneroSci and quantify the unlinkability and untraceability guarantee of Monero on top of it. Therefore, MoneroSci were introduced during this study with set features can use to analyze and query the Monero blockchain faster than existing data retrieving tools. As mentioned in Chapter 2, MoneroSci is the only Monero blockchain analysis platform currently exists. Most of the Bitcoin researchers use BlockSci [13] but the main limitation of it is it does not support for the Monero due to the difference in blockchain structure. Introduced MoneroSci analysis tool includes three main components to achieve the optimized environment. MoneroSci Parser converts the raw blockchain data into the well-defined data structure by using several optimization techniques as discussed in chapter 3 and 4. Core Blockchain Data is the store of blockchain data. MoneroSci Analysis Library is a data interface which use Core Blockchain Data for the analysis queries. Analysis library includes a set of classes and attributes which can use for analysis scenarios such as for retrieving any of blockchain element data or query the blockchain elements in easy. As shown in chapter 5, MoneroSci analysis tool achieved faster performance than existing tools. It can loads the whole Monero blockchain in 15 seconds instead existing tool requires more than 7 hours. As defined two research questions of this study, Is there any optimized way to analyze the whole Monero blockchain? and Develop a tool for exploring and analyzing linkability and traceability of Monero blockchain and transactions? yes, MoneroSci is the proposed solution for this. As one of the objectives of this study, MoneroSci tool is developed as a free and open source tool and anyone can refer the source code and improve the performance of it. Appendix A listed the documentation of MoneroSci and it includes the how to configure the parser and analysis tool in any Linux server and details of analysis library class methods which are useful for any of interest person who willing to set up and learn about the MoneroSci. And also, MoneroSci tool ready to deploy as a forensic tool for the law enforcement duties.

As mentioned in chapter 2, Monero untraceability attacks presented by Amrit Kumar et al.[17] and Malte Möser et al.[18] are developed and verified on top of the MoneroSci. It verifies the one of a research question, Can verify proposed traceability attacks conducted by Amrit Kumar et al. and Malte Möser et al. for Monero blockchain? in this study. MoneroSci gained results for the attack I: Leveraging zero mix-ins as 87% of non-RingCT transactions were traceable due to the zero mix-ins as observed in [17]. For the attack II: Leveraging output merging it gained results as 75% of non-RingCT transactions traceable due to the heuristic. In attack II it finds more transactions than the [17]. Then attack II has extended to analyze the linkability of Monero transactions as one of an objective of this study. It concludes that 1036758 non-RingCT transactions and 89251 RingCT transactions linked together. Thus, it can be concluded that the proposed MoneroSci analysis tool can be used as an optimized environment for analyze the Monero blockchain and implemented attacks can be used to quantify and analyze the Monero untraceability and unlinkability guarantee.

6.3 Conclusions about research problem

This study introduced an optimized analysis environment for Monero blockchain as a solution to the problem of the research. Based on the observations in chapter 5, MoneroSci performs faster than existing data retrieving tools. It only needs 15 seconds to load the whole Monero blockchain data into the memory and it has an analysis interface with a set of methods and attributes. One of the main problems for the Monero blockchain researchers and law enforcement parties was the lack of a tool to analyze the blockchain. Therefore, it increases attacks and criminal uses in Monero as discussed in chapter one. This study contributed to the domain of these situations by introducing MoneroSci Analysis Tool. The study also contributed to the domain of Monero blockchain privacy by quantifying the unlinkability and untraceability guarantees of it.

6.4 Limitations

During the research, it only used the publicly available Monero blockchain data for the study and it not used any of user data bind with the malicious attacks to reveals user identities. And also this study used fix height of Monero blockchain data from block number 1 to block number 1546000 because public Monero blockchain growing faster in every moment. And, in this study, it only considers the analysis of blockchain data and mining pool analysis not been covered. Developed proposed optimized environment, MoneroSci Analysis tool not tested for all types of coins in CryptoNote cryptocurrency family.

6.5 Implications for further research

Considering the overall study of this research, there are main key points can be considered as future works. In the linkability analysis of Monero, the extended attack resulted from a graph with linked transactions. It shows that Monero transactions can be linked together to analyze the history of transactions. But it not linked those transaction chains to a specific user public address. This can be considered as future work of this study. How can we trace a user in pseudo-anonymously using linked transactions? And also as discussed in chapter 2, BlockSci [13] is the one of the popular and mostly using blockchain analysis platform among the researchers, but it does not support for Monero. Therefore MoneroSci tool can be integrated into the BlockSci as a new additional layer as a future work after done by some research on compatibility. Through this research, it introduced MoneroSci analysis tool which include the main three components. Here MoneroSci parser is the key component because it deals with the raw blockchain data. Therefore improving performance in speed or memory consumption of MoneroSci parser also can be considered as a future work of this study.

References

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

[2] M. Fleder, M. S. Kester, and S. Pillai, “Bitcoin transaction graph analysis,” *CoRR*, vol. abs/1502.01657, 2015. [Online]. Available: http://arxiv.org/abs/1502.01657

[3] S. Meiklejohn, M. Pomarole, G. Jordan, K. Levchenko, D. McCoy, G. M. Voelker, and S. Savage, “A fistful of bitcoins,” *Communications of the ACM*, vol. 59, no. 4, pp. 86–93, 2016.

[4] F. Reid and M. Harrigan, “An analysis of anonymity in the bitcoin system,” *CoRR*, vol. abs/1107.4524, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1107.html#abs-1107-4524

[5] D. Ron and A. Shamir, “Quantitative analysis of the full bitcoin transaction graph,” in *Financial Cryptography and Data Security*, A.-R. Sadeghi, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013, pp. 6–24.

[6] F. Reid and M. Harrigan, “An analysis of anonymity in the bitcoin system,” *CoRR*, vol. abs/1107.4524, 2011. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1107.html#abs-1107-4524

[7] M. Möser, “Anonymity of bitcoin transactions an analysis of mixing services,” 2013.

[8] “Cryptocurrency Market Capitalizations,” October 2018. [Online]. Available: https://coinmarketcap.com/currencies/monero/

[9] D. Chaum and E. van Heyst, “Group signatures,” *Advances in Cryptology - EUROCRYPT ’91, LNCS 547*, pp. 257–265, 1991.

[10] E. Fujisaki and K. Suzuki, “Traceable ring signature,” in *Proceedings of the 10th International Conference on Practice and Theory in Public-key Cryptography*,

ser. PKC'07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 181–200. [Online]. Available: http://dl.acm.org/citation.cfm?id=1760564.1760582

[11] R. L. Rivest, A. Shamir, and Y. Tauman, "How to leak a secret," in *PROCEEDINGS OF THE 7TH INTERNATIONAL CONFERENCE ON THE THEORY AND APPLICATION OF CRYPTOLOGY AND INFORMATION SECURITY: ADVANCES IN CRYPTOLOGY.* Springer-Verlag, 2001, pp. 554–567.

[12] "Monero Official Website," October 2018. [Online]. Available: https://getmonero.org

[13] H. A. Kalodner, S. Goldfeder, A. Chator, M. Möser, and A. Narayanan, "Blocksci: Design and applications of a blockchain analysis platform," *CoRR*, vol. abs/1709.02489, 2017. [Online]. Available: http://arxiv.org/abs/1709.02489

[14] "United States of America vs. Alexandre Cazes. Verified complaint for forfeiture In Rem," July 2017. [Online]. Available: https://www.justice.gov/opa/press-release/file/982821/download

[15] "The Shadow Brokers may have received up to 1500 Monero ($66,000) from their June "monthly dump service"," July 2017. [Online]. Available: https://steemit.com/shadowbrokers/@wh1sks/Theshadowbrokers-may-have-received-up-to-1500-monerousd66-000-from-their\-june-monthly-dump-service

[16] "Thomas Fox-Brewster. Wannacry hackers are using this Swiss company to launder $142,000 Bitcoin ransoms."," August 2017. [Online]. Available: https://www.forbes.com/sites/thomasbrewster/2017/08/03/wannacryhackers-use-shapeshift-to-launder-bitcoin

[17] A. Kumar, C. Fischer, S. Tople, and P. Saxena, "A traceability analysis of monero's blockchain," in *Computer Security - ESORICS 2017 - 22nd European Symposium on Research in Computer Security, Oslo, Norway, September 11-15, 2017, Proceedings, Part II*, 2017, pp. 153–173. [Online]. Available: https://doi.org/10.1007/978-3-319-66399-9_9

[18] M. Möser, K. Soska, E. Heilman, K. Lee, H. Heffan, S. Srivastava, K. Hogan, J. Hennessey, A. Miller, A. Narayanan, and N. Christin, "An empirical analysis of traceability in the monero blockchain," *PoPETs*, vol. 2018, no. 3, pp. 143–163, 2018. [Online]. Available: https://doi.org/10.1515/popets-2018-0025

[19] N. van Saberhagen, "Cryptonote v2.0," 2013. [Online]. Available: https://cryptonote.org/whitepaper.pdf

[20] S. Noether, S. Noether, and A. Mackenzie, "A note on chain reactions in traceability in cryptonote 2.0," *Monero Research Lab*, no. MRL-0001, 2014.

[21] A. Mackenzie and S. Noether, "Improving obfuscation in the cryptonote protocol," *Monero Research Lab*, no. MRL-0004, 2015.

[22] "Monero Research Lab," August 2018. [Online]. Available: https://getmonero.org/resources/research-lab

[23] A. Mackenzie and S. Noether, "Ring confidential transactions," *Monero Research Lab*, no. MRL-0005, 2016.

[24] J. Rubin, "Bitcoin spark framework," 2019. [Online]. Available: https://github.com/JeremyRubin/BTCSpark

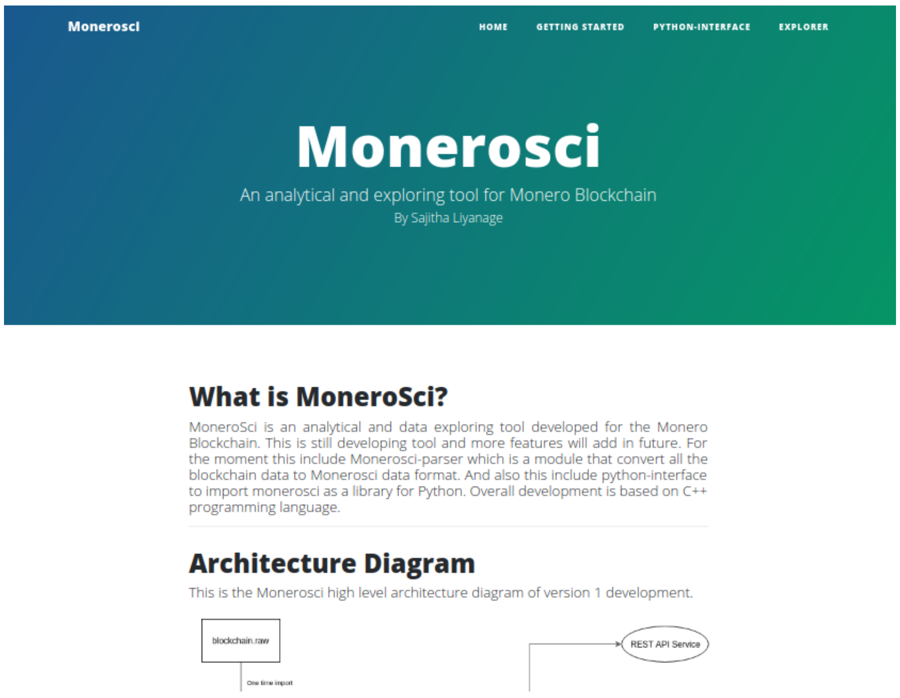
[25] M. Bartoletti, A. Bracciali, S. Lande, and L. Pompianu, "A general framework for bitcoin analytics," *CoRR*, vol. abs/1707.01021, 2017. [Online]. Available: http://arxiv.org/abs/1707.01021

[26] "Onion Monero Blockchain Explorer," August 2018. [Online]. Available: https://github.com/moneroexamples/onion-monero-blockchain-explorer

Appendix A

MoneroSci Documentation

Figures A.1 - A.9 showcases the MoneroSci Documentation created including the all the information about MoneroSci parser, Analysis Library and configuration steps of it.



Monerosci

An analytical and exploring tool for Monero Blockchain

By Sajitha Liyanage

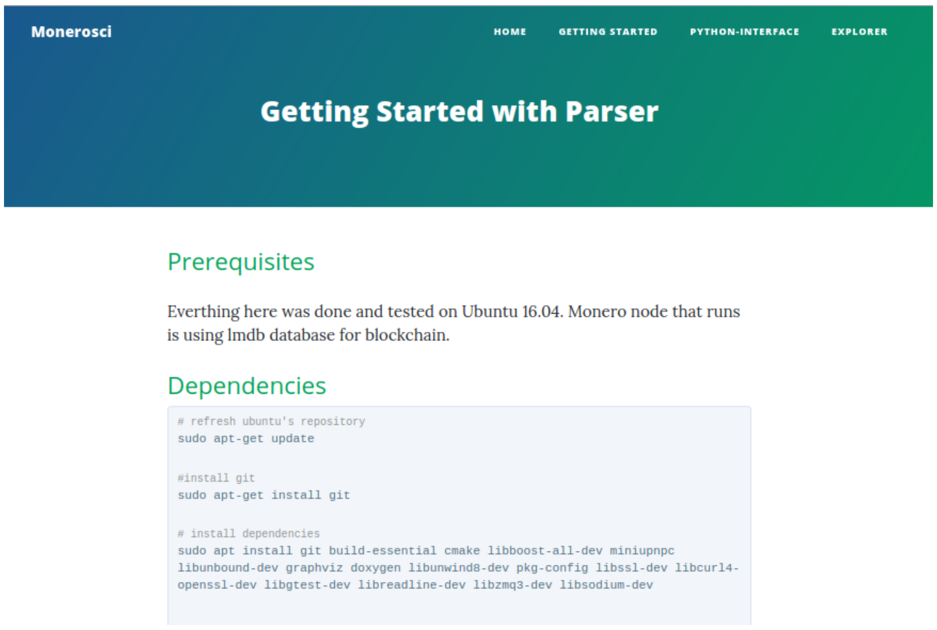
What is MoneroSci?

MoneroSci is an analytical and data exploring tool developed for the Monero Blockchain. This is still developing tool and more features will add in future. For the moment this include Monerosci-parser which is a module that convert all the blockchain data to Monerosci data format. And also this include python-interface to import monerosci as a library for Python. Overall development is based on C++ programming language.

Architecture Diagram

This is the Monerosci high level architecture diagram of version 1 development.

Figure A.1: Homepage of the MoneroSci documentation



Getting Started with Parser

Prerequisites

Everthing here was done and tested on Ubuntu 16.04. Monero node that runs is using lmdb database for blockchain.

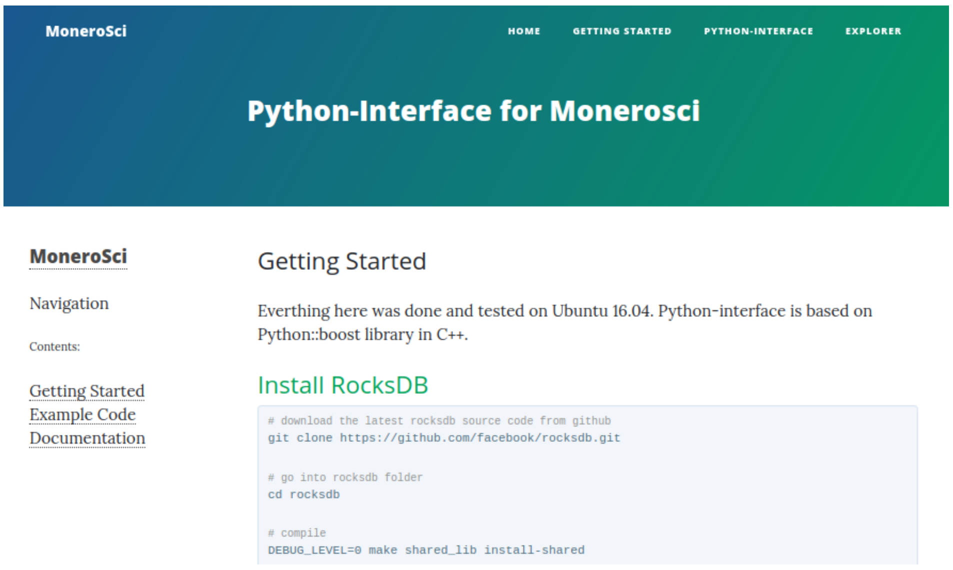
Dependencies

```
# refresh ubuntu's repository
sudo apt-get update

#install git
sudo apt-get install git

# install dependencies
sudo apt install git build-essential cmake libboost-all-dev miniupnpc
libunbound-dev graphviz doxygen libunwind8-dev pkg-config libssl-dev libcurl4-
openssl-dev libgtest-dev libreadline-dev libzmq3-dev libsodium-dev
```

Figure A.2: Configuration details of MoneroSci parser in the documentation



Python-Interface for Monerosci

Getting Started

Everthing here was done and tested on Ubuntu 16.04. Python-interface is based on Python::boost library in C++.

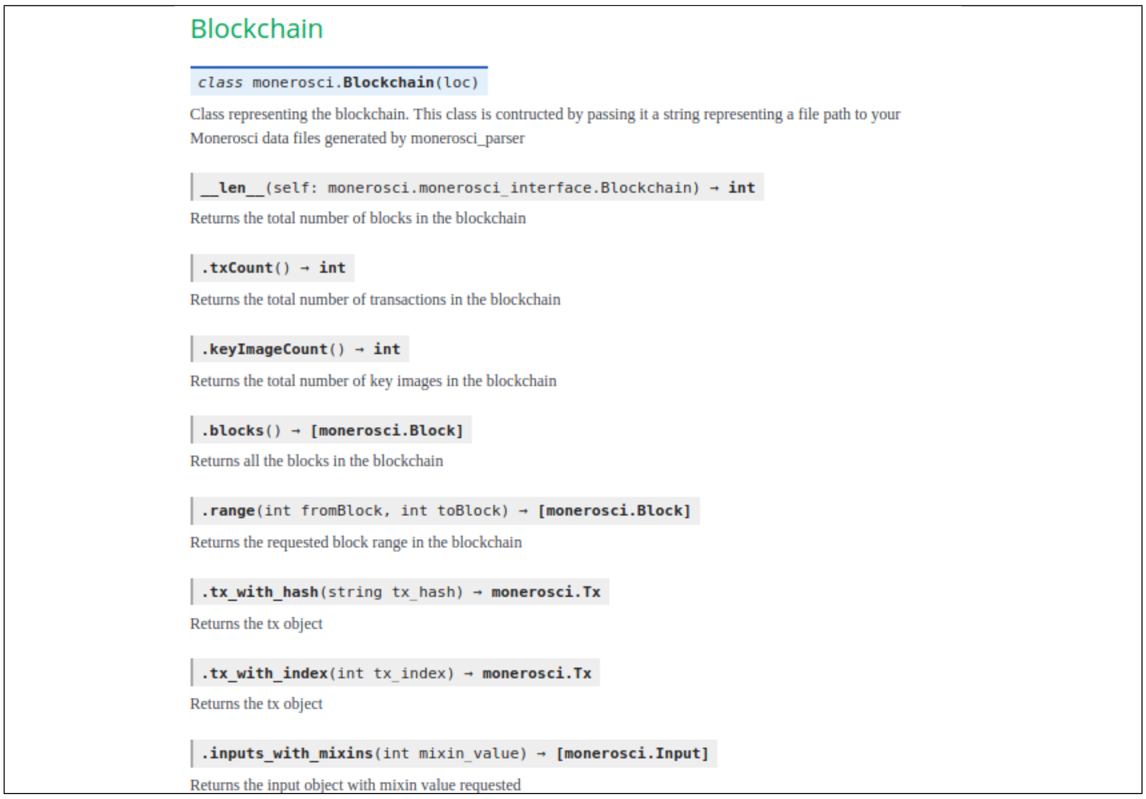
Install RocksDB

```
# download the latest rocksdb source code from github
git clone https://github.com/facebook/rocksdb.git

# go into rocksdb folder
cd rocksdb

# compile
DEBUG_LEVEL=0 make shared_lib install-shared
```

Figure A.3: Configuration details of MoneroSci Analysis Library in the documentation



Blockchain

`class monerosci.Blockchain(loc)`

Class representing the blockchain. This class is contructed by passing it a string representing a file path to your Monerosci data files generated by monerosci_parser

`__len__(self: monerosci.monerosci_interface.Blockchain) → int`

Returns the total number of blocks in the blockchain

`.txCount() → int`

Returns the total number of transactions in the blockchain

`.keyImageCount() → int`

Returns the total number of key images in the blockchain

`.blocks() → [monerosci.Block]`

Returns all the blocks in the blockchain

`.range(int fromBlock, int toBlock) → [monerosci.Block]`

Returns the requested block range in the blockchain

`.tx_with_hash(string tx_hash) → monerosci.Tx`

Returns the tx object

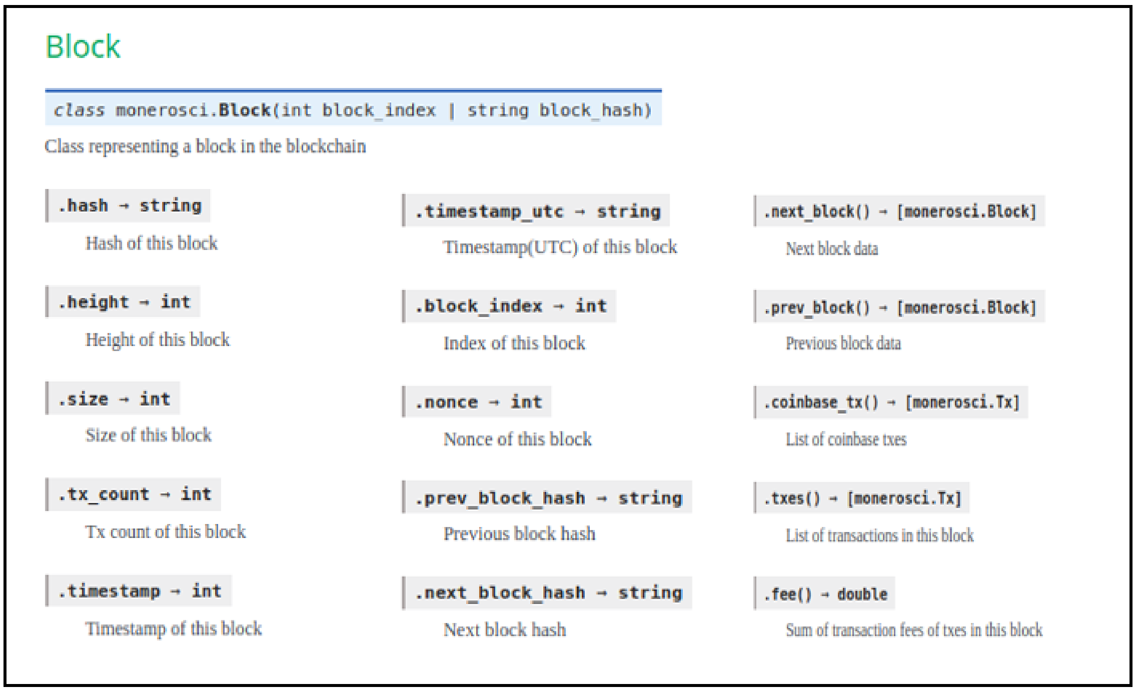
`.tx_with_index(int tx_index) → monerosci.Tx`

Returns the tx object

`.inputs_with_mixins(int mixin_value) → [monerosci.Input]`

Returns the input object with mixin value requested

Figure A.4: Blockchain class of MoneroSci Analysis Library in the documentation



Block

`class monerosci.Block(int block_index | string block_hash)`

Class representing a block in the blockchain

`.hash → string`

Hash of this block

`.height → int`

Height of this block

`.size → int`

Size of this block

`.tx_count → int`

Tx count of this block

`.timestamp → int`

Timestamp of this block

`.timestamp_utc → string`

Timestamp(UTC) of this block

`.block_index → int`

Index of this block

`.nonce → int`

Nonce of this block

`.prev_block_hash → string`

Previous block hash

`.next_block_hash → string`

Next block hash

`.next_block() → [monerosci.Block]`

Next block data

`.prev_block() → [monerosci.Block]`

Previous block data

`.coinbase_tx() → [monerosci.Tx]`

List of coinbase txes

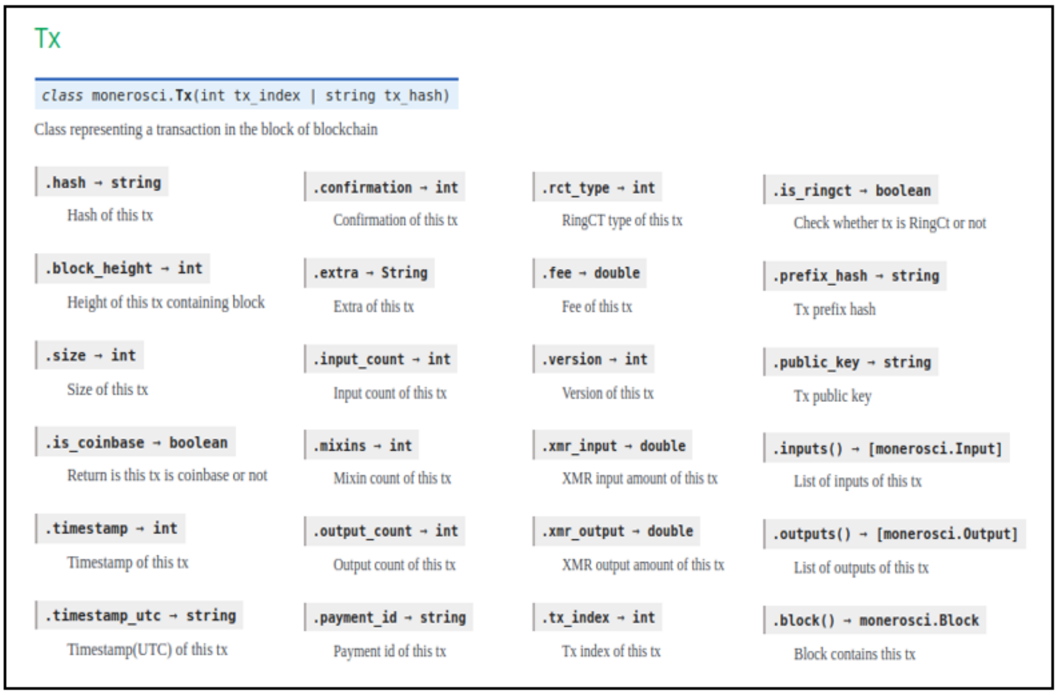
`.txes() → [monerosci.Tx]`

List of transactions in this block

`.fee() → double`

Sum of transaction fees of txes in this block

Figure A.5: Block class of MoneroSci Analysis Library in the documentation



Tx

`class monerosci.Tx(int tx_index | string tx_hash)`

Class representing a transaction in the block of blockchain

`.hash → string`
Hash of this tx

`.block_height → int`
Height of this tx containing block

`.size → int`
Size of this tx

`.is_coinbase → boolean`
Return is this tx is coinbase or not

`.timestamp → int`
Timestamp of this tx

`.timestamp_utc → string`
Timestamp(UTC) of this tx

`.confirmation → int`
Confirmation of this tx

`.extra → String`
Extra of this tx

`.input_count → int`
Input count of this tx

`.mixins → int`
Mixin count of this tx

`.output_count → int`
Output count of this tx

`.payment_id → string`
Payment id of this tx

`.rct_type → int`
RingCT type of this tx

`.fee → double`
Fee of this tx

`.version → int`
Version of this tx

`.xmr_input → double`
XMR input amount of this tx

`.xmr_output → double`
XMR output amount of this tx

`.tx_index → int`
Tx index of this tx

`.is_ringct → boolean`
Check whether tx is RingCt or not

`.prefix_hash → string`
Tx prefix hash

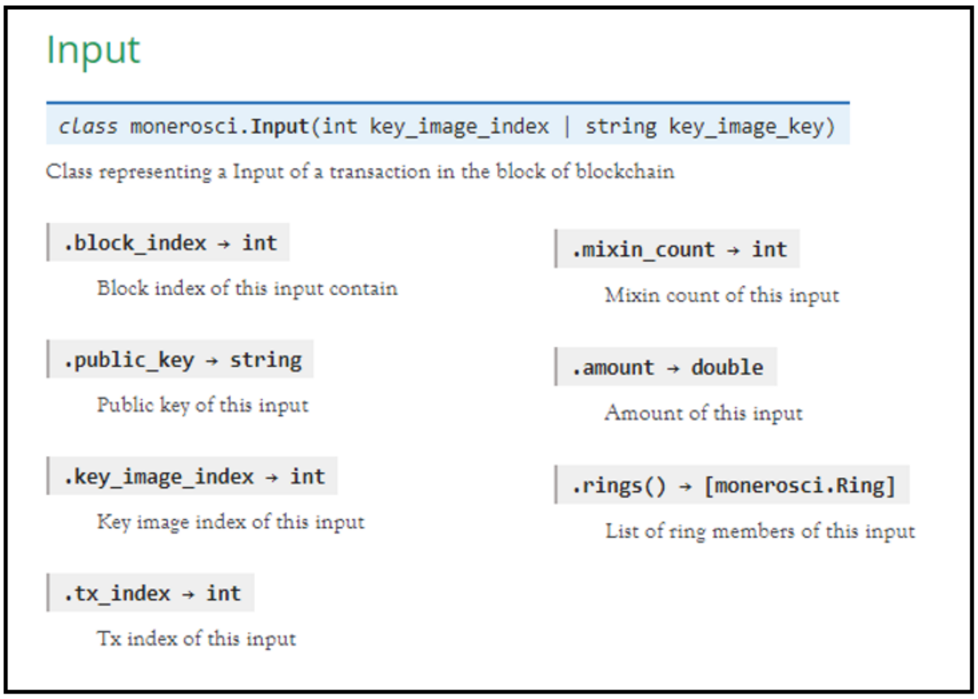
`.public_key → string`
Tx public key

`.inputs() → [monerosci.Input]`
List of inputs of this tx

`.outputs() → [monerosci.Output]`
List of outputs of this tx

`.block() → monerosci.Block`
Block contains this tx

Figure A.6: Tx class of MoneroSci Analysis Library in the documentation



Input

`class monerosci.Input(int key_image_index | string key_image_key)`

Class representing a Input of a transaction in the block of blockchain

`.block_index → int`
Block index of this input contain

`.public_key → string`
Public key of this input

`.key_image_index → int`
Key image index of this input

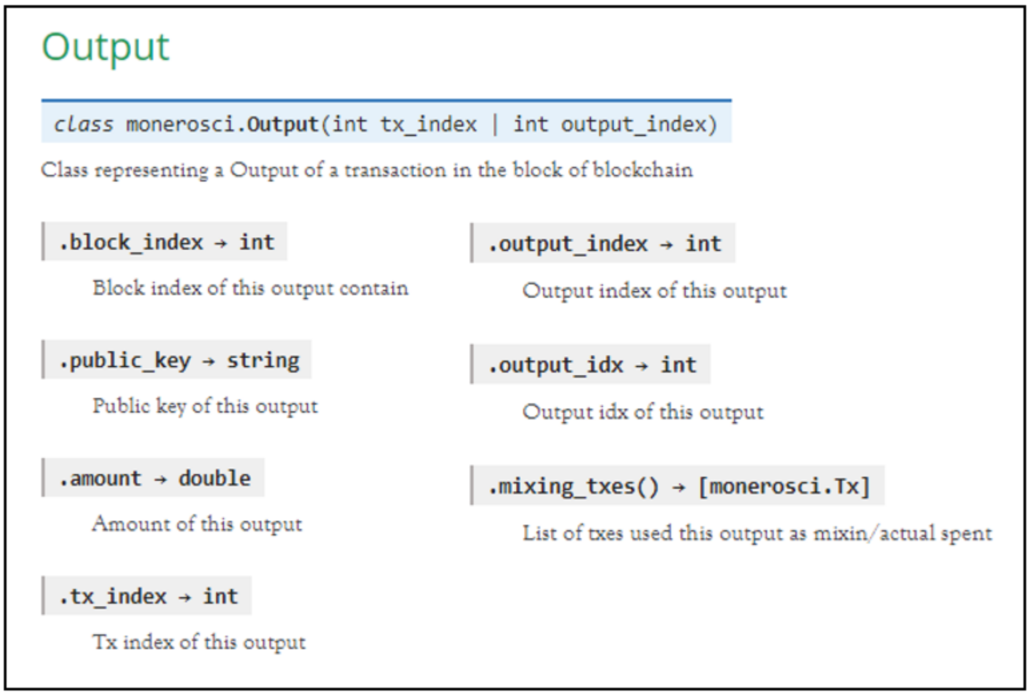
`.tx_index → int`
Tx index of this input

`.mixin_count → int`
Mixin count of this input

`.amount → double`
Amount of this input

`.rings() → [monerosci.Ring]`
List of ring members of this input

Figure A.7: Input class of MoneroSci Analysis Library in the documentation



Output

```
class monerosci.Output(int tx_index | int output_index)
```

Class representing a Output of a transaction in the block of blockchain

`.block_index → int`

Block index of this output contain

`.public_key → string`

Public key of this output

`.amount → double`

Amount of this output

`.tx_index → int`

Tx index of this output

`.output_index → int`

Output index of this output

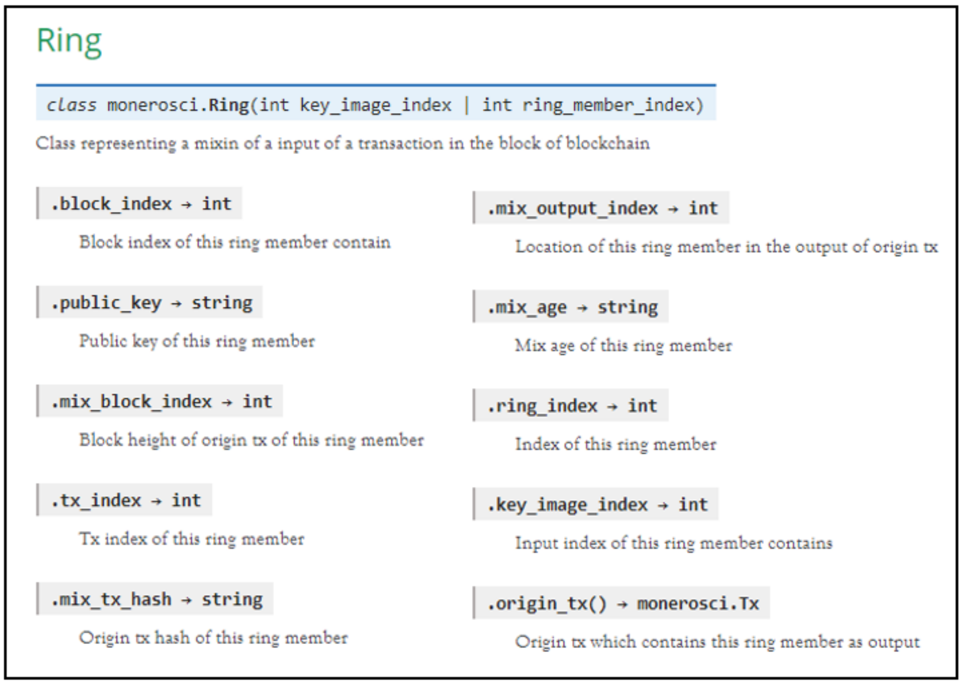
`.output_idx → int`

Output idx of this output

`.mixing_txes() → [monerosci.Tx]`

List of txes used this output as mixin/actual spent

Figure A.8: Output class of MoneroSci Analysis Library in the documentation



Ring

```
class monerosci.Ring(int key_image_index | int ring_member_index)
```

Class representing a mixin of a input of a transaction in the block of blockchain

`.block_index → int`

Block index of this ring member contain

`.public_key → string`

Public key of this ring member

`.mix_block_index → int`

Block height of origin tx of this ring member

`.tx_index → int`

Tx index of this ring member

`.mix_tx_hash → string`

Origin tx hash of this ring member

`.mix_output_index → int`

Location of this ring member in the output of origin tx

`.mix_age → string`

Mix age of this ring member

`.ring_index → int`

Index of this ring member

`.key_image_index → int`

Input index of this ring member contains

`.origin_tx() → monerosci.Tx`

Origin tx which contains this ring member as output

Figure A.9: Ring class of MoneroSci Analysis Library in the documentation