Network Analysis and Traceability on Monero Blockchain

SCSE20-0548

Submitted by: Phua Jia Sheng

Supervisor: Dr Sourav Sen Gupta

School of Computer Science and Engineering

Submitted in Partial Fulfilment of the Requirements

for the Degree of Bachelor of Computer Science

of the Nanyang Technological University

2020

Abstract

Monero was introduced as a privacy-focused alternative to Bitcoin in 2014. Based on privacy enhancing features such as ring signatures, stealth addresses and RingCT, it is supposed to protect the privacy of users while still solving the issue of double spending, which is rare in crypto currency. However, Monero is not foolproof when it comes to privacy, as previous studies have shown that it is vulnerable to weakness such as the 0-mixin attack. In this study, we explore just how much of Monero transactions are traceable, using previously known exploits to de-anonymize Monero inputs. The end goal of this study is to show that Monero is not as untraceable as thought to be, and that for Monero to achieve its privacy goals, it has to enforce privacy-by-default measures, such as using a minimum number of mixins.

Acknowledgements

I would like to thank my FYP mentor, Dr Sourav, for his guidance and understanding throughout this project. I would also like to thank my family, for their support throughout this project, as well as through my undergraduate studies in general.

Phua Jia Sheng

April 2021

List of Figures

Figure 1: An example of a Merkle Tree 10
Figure 2: The Monero transaction flow, along with the degrees of relationship, illustrated. 15
Figure 3: Sample block JSON object - block 1 JSON data 17
Figure 4: File structure for storing JSON files, pre-archival 18
Figure 5: File structure for storing JSON files, post-archival 19
Figure 6: ArangoDB Schema V1 20
Figure 7: ArangoDB Schema V2 23
Figure 8: ArangoDB Schema V3 25
Figure 9: Reading start index from text file 26
Figure 10: Connecting to ArangoDB 26
Figure 11: Re-archiving JSON files, writing index to text file 26
Figure 12: Workflow of json_to_adb.py 28
Figure 13: ArangoDB Schema V4 30
Figure 14: Creating the relevant ArangoDB documents 31
Figure 15: The overall setup of performing primary analysis, ... 33
Figure 16: Cumulative Deanonymization Ratio vs Block Height 34
Figure 17: CDR vs Block height, Round 1 vs Round 2 35

List of Tables

Table 1: Complete List of Zero Output Transactions 48

Table of Contents

Contents

Abstract 2

Acknowledgements 3

List of Figures 4

List of Tables 4

Table of Contents 5

1 Introduction 7

1.1 Objectives and Scope 7

1.2 Motivation 7

2 Blockchain Technology 8

2.1 Blocks 9

2.2 Block Time 10

2.3 Consensus Mechanisms 11

2.3.1 Proof-of-Work (PoW) 11

3 Monero 12

3.1 Privacy Features 12

3.1.1 Stealth Addresses 13

3.1.2 Ring Signatures 13

3.1.3 RingCT 14

3.2 Limitations 14

3.3 Flow of Monero Transactions 15

4 Data Collection 16

4.1 Considerations 16

4.2 Scraping JSON objects from xmrchain.net 16

4.3 File structure and archiving txs Directories 17

5 Building the Graph Database 19

5.1 Initial Schema Design 20

5.2 Anomalies in Data 21

5.2.1 Repeated Output Pubkey 22

5.2.2 Zero Output Transactions 23

5.2.3 Repeated Mixin in Same Input 23

5.3 json_to_adb.py Design Justifications and Code Snippets 25

6 Anonymity Analysis 28

6.1 Preparation 29
6.2 Primary Analysis 31
6.2.1 Setup 32
6.2.2 Results 34
6.3 Secondary Analysis 35
6.3.1 Results 35
7 Conclusion 36
References 36
Appendix 38
1.1 Sample Block JSON data 38
1.2 Sample Transaction JSON data 38
2 Source Code for get_json.py 40
3.1 Source Code for json_to_adb.py 41
3.2 Source Code for cleanup.py 44
4 Zero Output Transactions 48
5.1 Source Code for update_adb.py 60
5.2 Source Code for update_adb_cleanup.py 62
5.3 Source Code for check_status_restart.py 64
5.4 Source Code for bot.py 65
6.1 Source Code for anon_analysis.py 67
6.2 Source Code for anon_analysis_cleanup.py 69

1 Introduction

1.1 Objectives and Scope

The objective of this project would be to explore the extent to which the privacy features of Monero actually protects the privacy of its users. This will be done by first scraping the data of Monero transactions from publicly available block explorers, such as xmrchain.net. The data will then be imported into a graph-based database, and a tool will be built to help traverse, analyze and visualize the graph produced. One can then determine the extent to which the transactions on the Monero blockchain can be traced.

1.2 Motivation

Blockchain technology was introduced to the world when Satoshi Nakomoto wrote the Bitcoin white paper in 2009. A blockchain is an ever-growing ledger of data, which are stored in blocks, that are linked together cryptographically, ensuring the immutability of the data stored within. It has many applications, including cryptocurrency, which is favored by some due to its decentralized nature which renders it immune from government intervention, monetary policies, and censorship. Blockchain also has more commercial applications, such as inventory and supply chain management.

However, the openness of blockchain has a downside: a lack of privacy in certain use cases where privacy is important. For example, in cryptocurrency, the pseudonymous nature of wallets means that once a wallet is connected to a real-world entity, all the previous transactions made by that wallet can then be traced back to the real-world entity. The open nature of blockchain also results in a decrease in fungibility, since coins that were part of illicit transactions can be ‘blacklisted’. Another example would be a blockchain based voting system, where voting records should be kept secret.

To solve the issue of a lack of privacy, a few privacy-oriented blockchains have been created. More specifically, privacy-oriented cryptocurrency such as ZCash, which uses zero-knowledge proofs, and Monero, which has built-in, privacy-by-default features such as RingCT and stealth addresses, have emerged.

However, such features are not perfect. There has been previous research that has pointed out weaknesses in the design of Monero, such as the 0-mixin attack against older transactions when no minimum number of mixins were enforced on users, as well as temporal analysis, which is based on the theory that the mixin with the highest block number is the most likely to be the real input to a transaction.

2 Blockchain Technology

A blockchain is a growing list of records, individually known as blocks, that are linked together cryptographically. Every block typically contains a cryptographic hash digest of the pervious block it was built on, a timestamp of when the block is mined, as well as transaction data. [1] The transaction data in blocks are immutable by design. This is because once data is recorded in a specific block, it cannot be altered without changing all subsequent blocks. A change in data results in a change in hash digest of the block it is contained in, and since a subsequent block references previous block through its hash digest, a change in data results in a change in hash digest for all subsequent blocks. For a blockchain to be used as a distributed ledger, as is the case in cryptocurrencies such as Bitcoin, the blockchain is usually managed by a peer-to-peer network where all nodes adhere to a consensus protocol for validating new blocks. A blockchain can be considered to exemplify a distributed computing system with high Byzantine fault tolerance.

2.1 Blocks

Data in blocks include a list of valid transactions, that are hashed and encoded into a Merkle Tree, which is illustrated in the figure below.

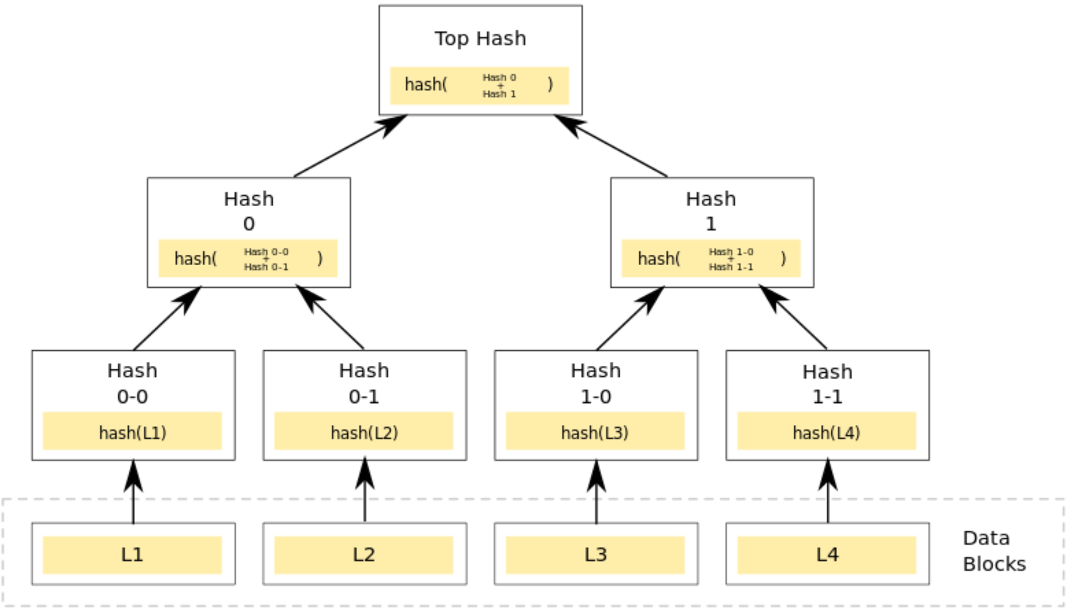


Figure 1: An example of a Merkle Tree

A Merkle tree is a data structure in which every leaf node is labelled with the cryptographic hash of a data block (in this case, blocks L1-L4), and every non-leaf node is labelled with the cryptographic hash of the labels of its child nodes. Merkle trees allow for efficient and secure verification of the contents of large data structures. [2]

Each block also contains the cryptographic hash of the previous block, which links the 2 blocks. This process of referencing the previous block is what maintains the immutability and integrity of data in a blockchain.

2.2 Block Time

The block time is the average time it takes for the network to generate one extra block in the blockchain, and it differs for every blockchain. Block time is usually determined by the consensus protocol, but it can also be affected by factors such as total hash power in the network, as well as the mining difficulty at the current block. The mining difficulty of a blockchain is periodically adjusted based on the total hash power in the network to keep the block time near a target time. For cryptocurrencies, block time is significant as shorter block times results in faster transactions on average. The block time for Ethereum is set to between 14 and 15 seconds, while for Bitcoin it is on average 10 minutes. [3]

2.3 Consensus Mechanisms

A network of blockchain nodes, like any other distributed computing systems, need a way of achieving and maintaining consensus. There are 2 popular consensus mechanisms in blockchain, namely Proof-of-Work (PoW) and Proof-of-Stake (PoS).

2.3.1 Proof-of-Work (PoW)

Proof of work (PoW) is a form of cryptographic proof in which one party (the prover) proves to others (the verifiers) that a certain amount of computational effort has been expended for some purpose. Usually, it involves producing some data (often a cryptographic hash) that is difficult, in terms of computational power and time, to produce, but is easy to verify. The piece of data produced must meet some pre-determined requirement. Bitcoin uses the Hashcash PoW system, along with the SHA-256 hashing algorithm, to secure the Bitcoin blockchain.

In Bitcoin, in order for a block to be accepted by the network, the miner of the block needs to produce a valid PoW – the SHA-256 hash of the block data concatenated with a nonce needs to be less than or equal to the target value, which is a 256-bit integer. The nonce is incremented after every failed attempt to generate a valid PoW. The lower the target value, the higher the mining difficulty of the block is. The mining difficulty needs to be high enough such that the probability of an individual node finding a valid PoW is low.

Since in Bitcoin, each block references the previous block by its hash, in order for a malicious actor to alter data in a block that has already been mined, the malicious actor also needs to alter all subsequent blocks. In the case of competing chains, the Bitcoin network recognises the chain with the largest cumulative PoW as the legitimate chain.

3 Monero

Monero is a privacy-focused cryptocurrency released on 18 April, 2014. Monero is based on CryptoNote, and uses a obfuscated public ledger, meaning that any node in the network can send or broadcast transactions, but no outside observer can tell the source, amount, or destination. Monero uses a PoW mechanism to issue new coins and incentivize miners to secure the network and validate transactions. [4] As of 2019, Monero has switched to the RandomX PoW algorithm to further increase ASIC-resistance of Monero mining. [5]

3.1 Privacy Features

Monero uses different privacy-enhancing technologies (PETs) to achieve anonymity and fungibility. Such technologies include stealth addresses, ring signatures, as well as RingCT, which will be discussed below. While some features, like stealth addresses and ring signatures are included from the first block and are part of Monero's core features, others, like RingCT and the standardization of ring sizes, were introduced as measures to improve privacy in response to research pointing out flaws in Monero's existing PETs.

3.1.1 Stealth Addresses

Stealth addresses are an important part of Monero's inherent privacy. They allow and require the sender to create random one-time addresses for every transaction on behalf of the recipient. One-time stealth addresses are generated using the Dual-Key Stealth Address Protocol (DKSAP) [6], based on 2 pieces of information: the first is a shared secret produced by the elliptic-curve Diffie–Hellman (ECDH) key agreement, and the second is the public key of the recipient. The recipient then actively scans the blockchain, detects if a transaction is intended for their address, and recovers the private key for this one-time public key to access the funds. [7]

3.1.2 Ring Signatures

In cryptography, a ring signature is a type of digital signature that can be performed by any member of a set of users that each have valid keys. Therefore, a message signed with a ring signature is guaranteed to be endorsed by someone in a particular set of people, but it should be computationally infeasible to tell exactly which member of the group signed the message. Ring signatures were invented by Ron Rivest, Adi Shamir, and Yael Tauman Kalai, and introduced at ASIACRYPT in 2001. [8]

In Monero, a ring signature is used to form the input of a transaction. The public spend key of the actual output being spent in the transaction, as well as the public keys of past outputs, make up the ring signature. [9] This reduces traceability in the Monero blockchain, since it will take substantial time and effort to determine the real input being used in each transaction. Double spending is prevented via the use of key images.

3.1.3 RingCT

Ring Confidential Transactions (RingCT) improves the confidentiality of Monero transactions by hiding the transaction amount. Ring CT was implemented in block 1,220,516 in January 2017. After September 2017, this feature became mandatory for all transactions on the network.

Prior to the implementation of RingCT, Monero required transaction amounts to be divided into denominations. For example, if Bob wanted to initiate a transaction of 12.5 Monero to Alice, this output would be divided into 3 separate rings of: 2, 0.5 and 10. This technique ensured that there was always an ample amount of ring members that could be found on the network, since a ring signature could only ring together outputs that were of the same value. However, the limitation is that, from the perspective of an outside party, they would be able to see the amounts that were being transacted. RingCT gets around this by using the Pedersen Commitment to hide the actual amount being transacted, while still ensuring that the sender has sufficient unspent outputs to cover the input and transaction fees. [10]

3.2 Limitations

Although Monero has been touted as having privacy-by-default, it still has limitation and past transactions are still vulnerable to de-anonymization analysis, usually due to privacy enhancing measures not being enforced in the past. For example, before block 1,009,827, there were no minimum number of mixins (decoy ring members of ring signatures) enforced. A large proportion of transactions from the period have 0 mixins and are thus deducible. This also has a cascade effect where transactions with 1 mixin can become deducible if the only mixin has been marked spent in pervious 0-mixin inputs, and so on. This is known as the 0-mixin attack.

Researchers have also conducted other forms of heuristic-based de-anonymization analysis, such as temporal analysis based on the Guess-Newest heuristic and Monte Carlo Simulation, deducing the real input among a ring signature based on their timestamp.

In Chapter 7, I will be using the 0-mixin attack to conduct anonymity analysis, as it is the most straight-forward to implement, and will reveal the percentage of inputs that are deducible as a result of several rounds of analysis. [11]

3.3 Flow of Monero Transactions

A typical block on the Monero blockchain consists of 2 types of transactions: a coinbase transaction, which contains the miner reward for the block as its output, as well as 0 or more non-coinbase transaction(s), with varying amounts of inputs and outputs. Each input has to originate either as an output of a coinbase transaction or a non-coinbase transaction.

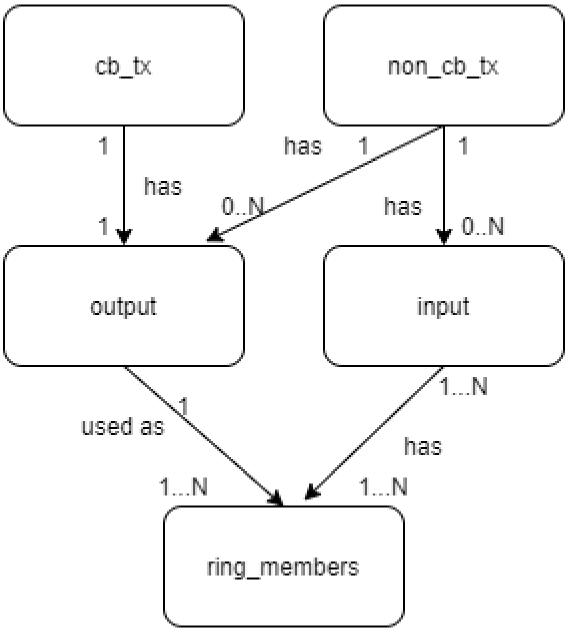


Figure 2: The Monero transaction flow, along with the degrees of relationship, illustrated.

Since each output can only be spent once, if the amount spent as an input is less than the output amount it originated from, the remainder is sent back as change to a new one-time address generated from the public key of the sender.

4 Data Collection

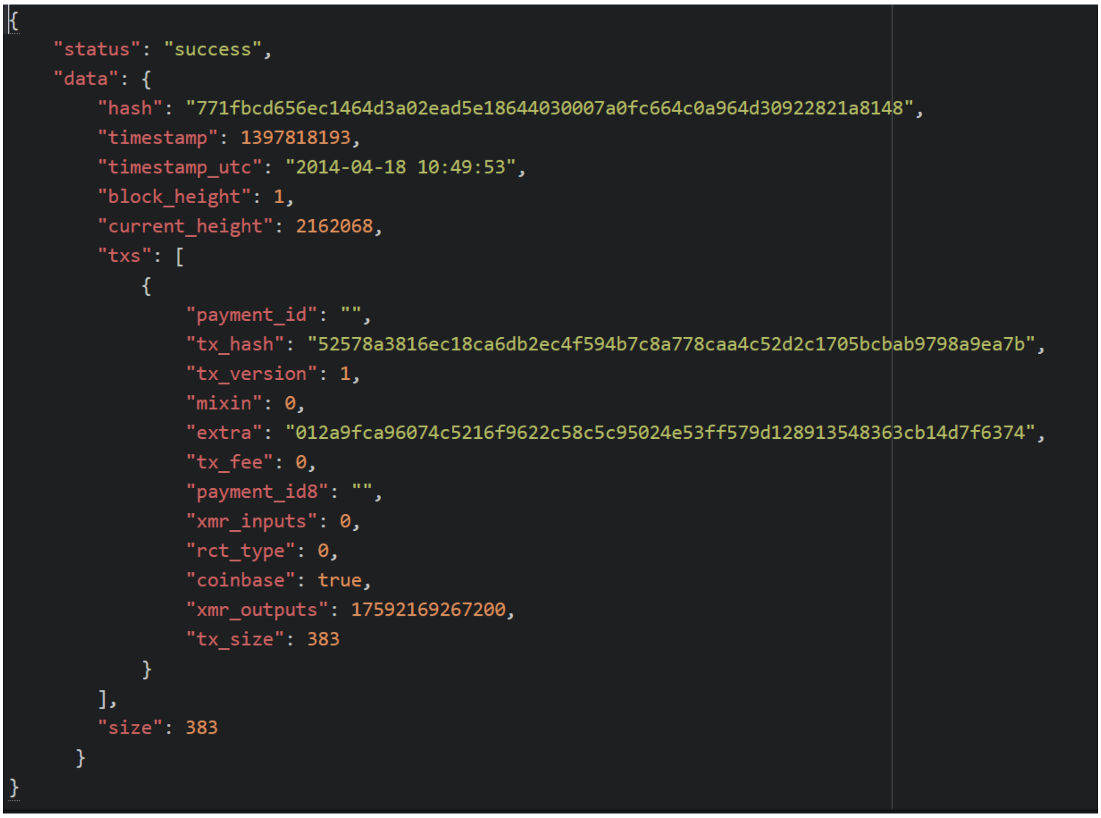
In order to perform network traceability analysis on the Monero blockchain, we must first collect data on Monero transactions, and store it in a format that is easy to retrieve and make amendments to. In this project, it was decided that we would scrape transaction data in the form of JSON objects from the xmrchain.net API endpoint, and store relevant data from the JSON objects in a graph-based database (such as Neo4j or ArangoDB).

4.1 Considerations

Initially, running a Monero node, and extracting data directly from the local blockchain of the Monero node was considered. However, due to the possible limitations and implications of running a Monero node on a NTU server, as well as possible difficulties interacting with a local blockchain, it was decided that scraping data from a public-facing block explorer would be preferable.

4.2 Scraping JSON objects from xmrchain.net

There were 2 types of JSON objects obtained from xmrchain.net: firstly, block JSON objects, which contains data such as block height, timestamp, transactions hashes, as well as miscellaneous data, and secondly, transaction JSON objects, which contains information such as inputs and outputs used in the transaction, mixins used in the inputs, as well as miscellaneous data. A sample JSON block object is shown as follows, and sample JSON block and transaction objects are included in the appendix:



```
{
    "status": "success",
    "data": {
        "hash": "771fbcd656ec1464d3a02ead5e18644030007a0fc664c0a964d30922821a8148",
        "timestamp": 1397818193,
        "timestamp_utc": "2014-04-18 10:49:53",
        "block_height": 1,
        "current_height": 2162068,
        "txs": [
            {
                "payment_id": "",
                "tx_hash": "52578a3816ec18ca6db2ec4f594b7c8a778caa4c52d2c1705bcbab9798a9ea7b",
                "tx_version": 1,
                "mixin": 0,
                "extra": "012a9fca96074c5216f9622c58c5c95024e53ff579d128913548363cb14d7f6374",
                "tx_fee": 0,
                "payment_id8": "",
                "xmr_inputs": 0,
                "rct_type": 0,
                "coinbase": true,
                "xmr_outputs": 17592169267200,
                "tx_size": 383
            }
        ],
        "size": 383
    }
}
```

Figure 3: Sample block JSON object - block 1 JSON data

Data scraping was done using a Python script, *get_json.py*, to iterate through block height in an ascending order, to send GET requests to the http://xmrchain.net/api/block/ endpoint, where block JSON objects are obtained. From the block JSON objects, the individual transaction hashes within the block are obtained. Iterating through the transaction hashes, GET requests are then sent to the API endpoint for transactions, http://xmrchain.net/api/transaction/, where the JSON transaction objects are then obtained. The source code for get_json.py is included in the appendix.

4.3 File structure and archiving txs Directories

The file structure for storing the JSON files are show below, using the example of blocks 0 and 1:

```
data

|-- blocks

|   |-- 0.json

|   `-- 1.json

`-- txs

    |-- 0

    |   `-- c88ce9783b4f11190d7b9c17a69c1c52200f9faaee8e98dd07e6811175177139.json
```

Figure 4: File structure for storing JSON files, pre-archival

Block JSON objects would be stored in the **blocks** directory, with the file name being the block height, while transaction JSON objects would be stored under a newly created directory, with the block height as directory name, and transaction hash as file name.

While in the process of scraping JSON objects from xmrchain.net, I ran into an unusual storage issue: the server that I was running the python script from still had significant storage space unused, but I was getting an error from the python script that indicated there was insufficient storage space. After a brief investigation, the problem was revealed to be that the server had ran out of inodes, which can be thought of as pointers to a file. Usually, the ratio of inodes to storage space is predetermined at time of portioning of storage, so if one needs to store a lot of small files, as in my case, the server would run out of inodes before it runs out of storage space.

The solution I undertook was to write a Python script to archive the directories in txs, as well as to update the *get_json.py* script to archive future txs directories. The updated file structure looked as follows:

```
data

|-- blocks

|   |-- 0.json

|   `-- 1.json

`-- txs
```

Figure 5: File structure for storing JSON files, post-archival

Before this solution, the server would need at least 2 inodes per block – 1 for the block JSON file, 1 for the coinbase transaction file, and 0 or more for the non-coinbase transaction files m(if they exist). This solution ensured that the maximum amount of inodes needed per block was 2 – 1 for the block JSON file, and the other for the archive file of the transaction JSON files. Essentially, the lower bound for the number of inodes needed per block became the upper bound, ensuring that the server would never run out of inodes, based on the highest block height of Monero at that point.

5 Building the Graph Database

After collecting the Monero transaction data in the form of JSON objects from xmrchain.net, we need to store the data in a format that is easier to retrieve and modify. A graph database is ideal because it allows us to explore the relationship between the different parts of Monero transactions, such as the inputs, outputs and mixins.

For this project, ArangoDB was selected as the graph database to be used due to its open-source nature and an abundance of documentation of its API. After a schema for the database was designed, a Python script, *ArangoDB_init.py* was written to implement the schema by initializing the appropriate vertex and edge collections. After which, another

Python script, *json_to_adb.py*, was used to parse information from the JSON objects and write them to the database.

5.1 Initial Schema Design

The initial schema design for the database was made based on certain assumptions about the nature of Monero transactions, based on the way that Monero was designed to be used, and based on how rational users would behave if they wanted to maximise their privacy. The schema is as shown below:

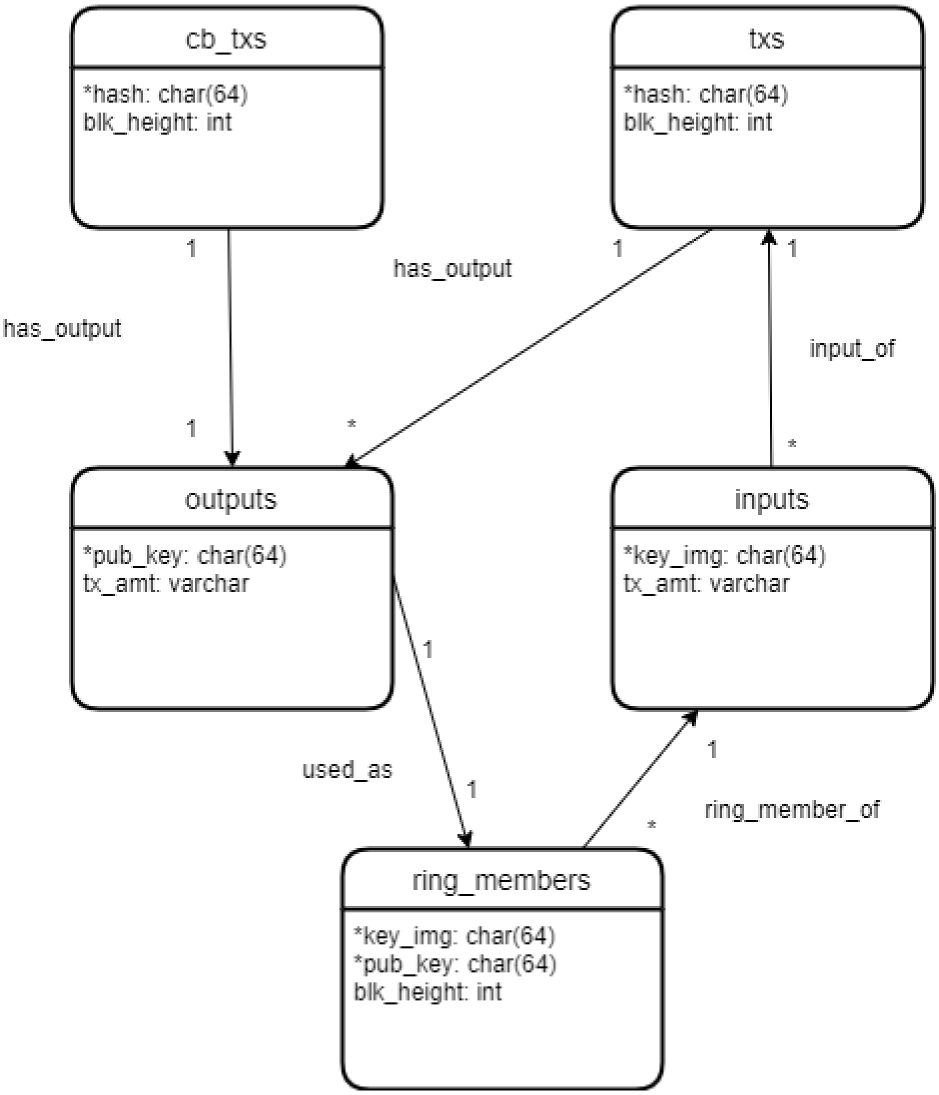


Figure 6: ArangoDB Schema V1

The following assumptions were used to make this schema:

1. Since the public keys generated for receiving a Monero transaction are essentially one-time addresses, they will not be re-used again to receive another transaction, and hence can be used to uniquely identify outputs of transactions. Hence, they are used as keys of the vertex collection *outputs*, and why each output can only belong to coinbase transaction or transacation.
2. Each block will only have 1 coinbase transaction, with each coinbase transaction having only 1 output.
3. Each output can be used as a real input in a future transaction at most once, and can be used as a decoy infinitely many times. However, it will not be used in the same transaction input more than once, since it will defeat the purpose of using mixins by essentially duplicate the candidate inputs. Hence, a combination of the output's public key, and the key image of the input it is used in, should be able to uniquely identify mixins.

However, as we shall see in the next section, there is empirical data to disprove assumptions 1 and 3, and we will have to make changes to the schema to reflect the reality of Monero transactions.

5.2 Anomalies in Data

This section shall explore anomalies encountered during the writing of data from JSON objects to the graph database. These anomalies resulted in the amendment of the graph database schema, as well as the code logic of the *json_to_adb.py* script. We shall explore what these anomalies are, the first instance where they are encountered, as well as possible explanations for these anomalies. These anomalies have been cross-checked with a second block explorer, localmonero.co, to ensure that they are actual anomalies and not just an error/bug with xmrchain.net.

5.2.1 Repeated Output Pubkey

The first anomaly encountered was a repeated output pubkey, or stealth address. This occurred at block 200401, transaction 86c73c157e6fe10aa98c78f25d5a42dbc531e697cd8a366b1292e71cf372b33b, where the pubkey 9b2e4c0281c0b02e7c53291a94d1d0cbff8883f8024f5142ee494ffbbd088071 was re-used from transaction 1a24eea7554d5d2e8f2d5c4fdbbb00bb5ec4ba4718681dd46f0718675d1a5efc from block 200382. This anomaly disproves assumption 1 used in section 6.1 for the initial schema design.

This anomaly affects the schema design of vertex collections cb_txs, txs and outputs, as well as edge collections has_output from cb_txs to outputs and txs to outputs. The changes are listed as follows:

1. An outputs vertex can belong to more than 1 txs or cb_txs vertices.
2. Outputs vertices have a new property, total_inputs which track the total amount of Monero going into the stealth address
3. Txs to outputs and cb_txs to outputs edges have a new property, tx_amt, which replaces the old outputs property.

The new revised schema is shown below:

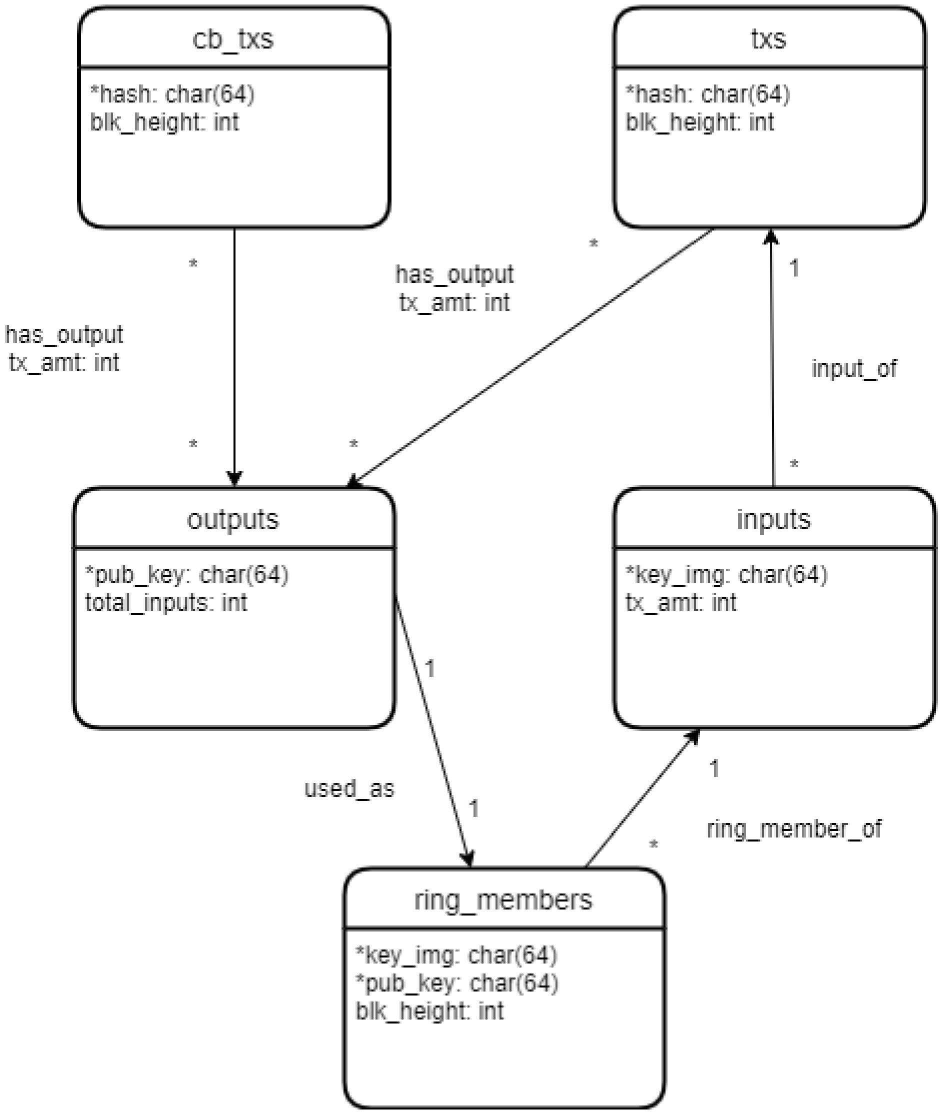


Figure 7: ArangoDB Schema V2

5.2.2 Zero Output Transactions

The next anomaly encountered was 0-output transactions, which were first encountered in multiple transactions in block 202612. The full list of transactions can be found under Table 1 in the Appendix.

5.2.3 Repeated Mixin in Same Input

The third anomaly that was observed was the re-use of a mixin in the same input. This was first observed in block 327 625, transaction

06dfb085e0f5de4cc10b5ab134f06b8c5285887a5cb53c155ec52330887576fd, where the mixin cf216608163fc795aba56d68ab52304a18c0cee62f3b7d0968fea98aede54f79 was used twice in the input 42969704d33bb76decb429235ea8778b4c92e0584afb8a8c680fd62702299c9f. Repeated mixins in the same input are detrimental to reducing the traceability of Monero transactions as they give the appearance of meeting the minimum amount of mixins, but the repeated mixin does not actually contribute to reducing traceability. A possible explanation for the repeated mixin is that there were less outputs of a particular denominations than the required number of mixins, and hence the sender had no choice but to re-use a mixin in order to meet the minimum number of mixins. For example, at block height 327 625, the minimum number of mixins seemed to be 5. It was possible that in the entire history of the blockchain at the time, there were only ever 4 outputs of denomination 0.011111111111 XMR, hence the need to repeat mixins in order to have 5 mixins.

As a result of the discovery of this anomaly, the key image of an input in which a mixin is used in, as well as the public key of the output in which it originated from, can no longer uniquely identify ring members together. As such, a new element must be added to the compound key in order to differentiate repeated mixins. The index of the mixin (since mixins to an input appears as an array in the JSON object) is chosen as this element. In addition, a new property is added to the ring_members collection: ‘duplicate’, a Boolean, which identifies whether the particular mixin is repeated in the same input. The new schema is reflected below:

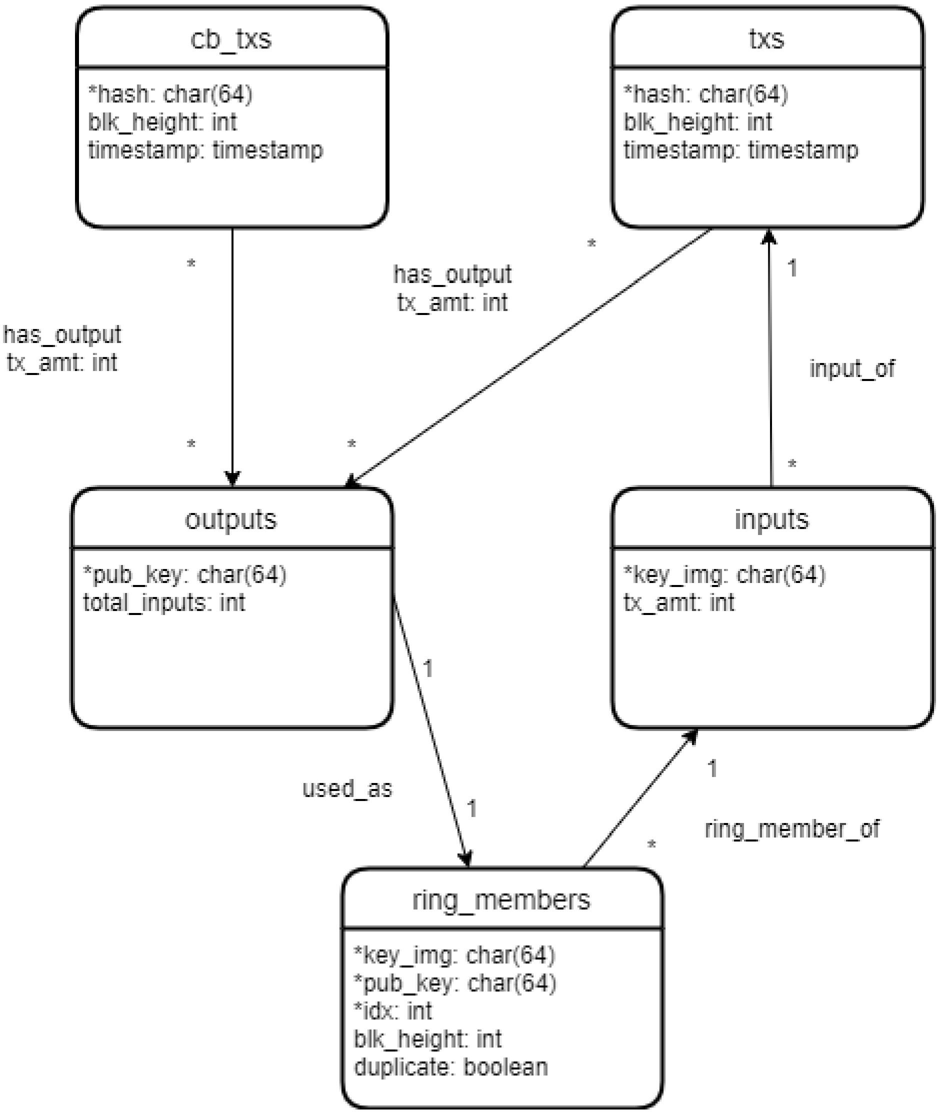
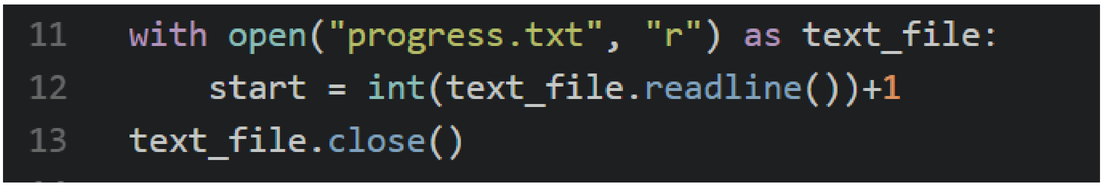


Figure 8: ArangoDB Schema V3

5.3 json_to_adb.py Design Justifications and Code Snippets

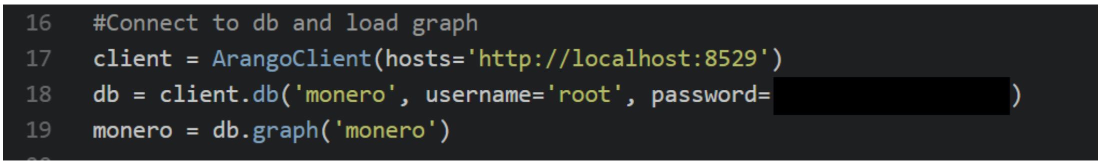
In this section, we will go through a few key snippets of the json_to_adb.py script. The full source code will be made available in the appendix.



```python
with open("progress.txt", "r") as text_file:
    start = int(text_file.readline())+1
text_file.close()
```

Figure 9: Reading start index from text file

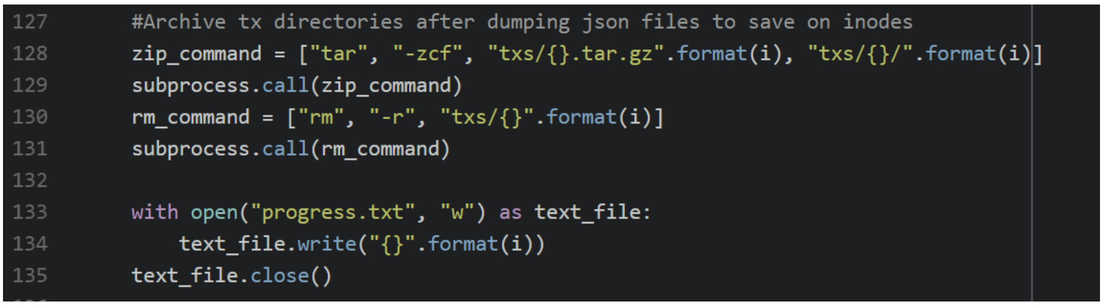
First, we will read in the start index for the for loop from a text file, progress.txt. The reason for doing so is that this script is expected to run for a long time, in the order of days of weeks, and in between, the script might stop running for reasons like unhandled exceptions, running out of memory, or unexpected power failure of the power source powering the server where it is run from.



```python
#Connect to db and load graph
client = ArangoClient(hosts='http://localhost:8529')
db = client.db('monero', username='root', password=[redacted])
monero = db.graph('monero')
```

Figure 10: Connecting to ArangoDB

We then connect to the ArangoDB server on localhost and load the graph. After which, we will enter the for loop, which will extract the archived JSON transaction files by block height, extract the relevant data, and create the appropriate vertex and edge documents and write them to the ArangoDB graph.



```python
    #Archive tx directories after dumping json files to save on inodes
    zip_command = ["tar", "-zcf", "txs/{}.tar.gz".format(i), "txs/{}/".format(i)]
    subprocess.call(zip_command)
    rm_command = ["rm", "-r", "txs/{}".format(i)]
    subprocess.call(rm_command)

    with open("progress.txt", "w") as text_file:
        text_file.write("{}".format(i))
    text_file.close()
```

Figure 11: Re-archiving JSON files, writing index to text file

Once the vertex and edge documents are created for each block, we will re-archive the JSON transaction files and store the current, completed for loop index in the same text file that we read from earlier.

As disruptions to the script can occur in the middle of the for loop, we have created a clean-up script, *cleanup.py*, which is similar to *json_to_adb.py*. In the event that a disruption occurs to *json_to_adb.py* in the middle of a for loop, cleanup.py will be called. For example, if a disruption happens in the middle of writing block 100 data to ArangoDB, the index stored in *progress.txt* will be 99, since the writing to progress.txt happens at the end of the for loop. *Cleanup.py* will then read in the index 99, start the for loop from index 100, and do the writing of block 100 data to ArangoDB, but with try-except block to handle key errors that ArangoDB will throw, due to data that have already been written prior to the disruption happening. After block 100 data has been written, *cleanup.py* will then write the index 100 to *progress.txt*, and exit. We can then call *json_to_adb.py* again to continue. If a disruption happens in the middle of running *cleanup.py*, we will simply call *cleanup.py* again. The full source code for cleanup.py is available in the appendix.

The overall workflow of the *json_to_adb.py* script and its interaction with the JSON files, *progress.txt* and ArangoDB is summarized in the figure below:

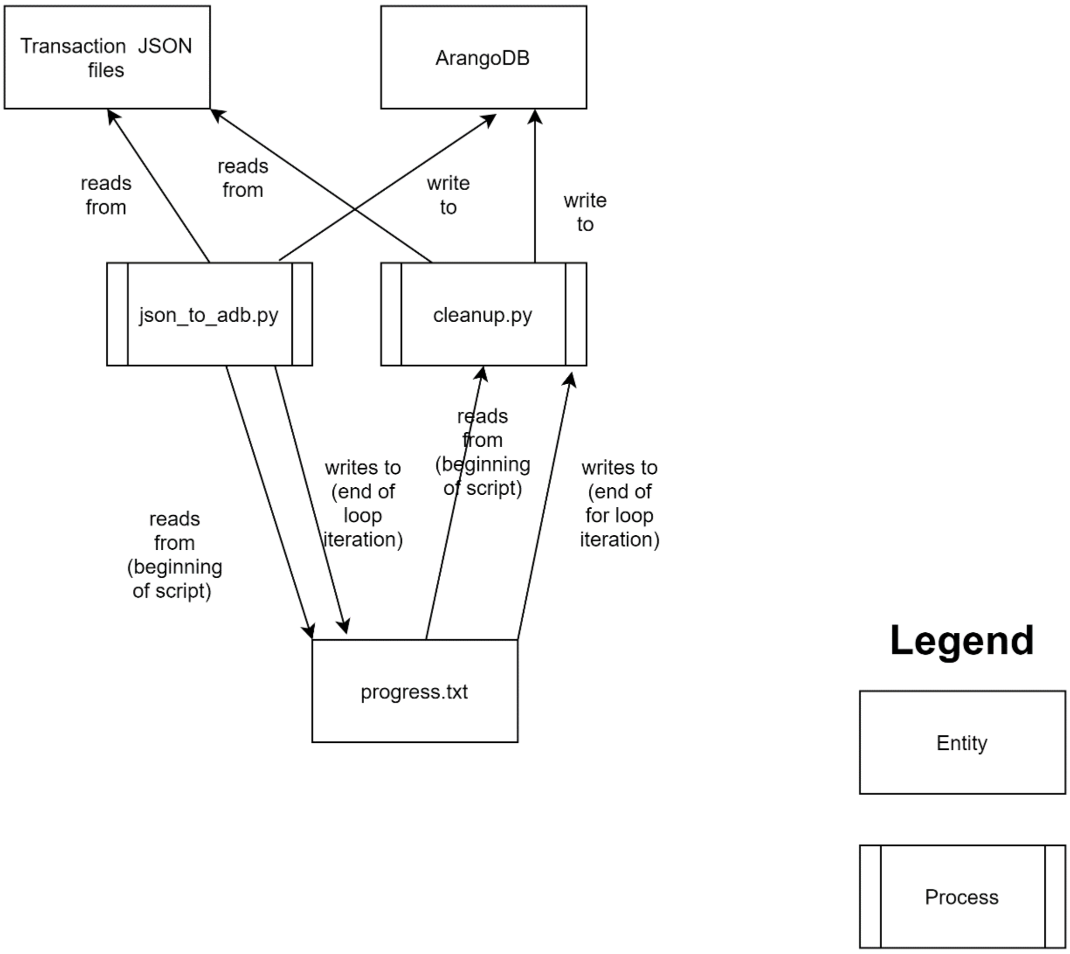

Transaction JSON files
ArangoDB
reads from
reads from
write to
write to
json_to_adb.py
cleanup.py
reads from (beginning of script)
writes to (end of loop iteration)
reads from (beginning of script)
writes to (end of for loop iteration)
progress.txt
Legend
Entity
Process

Figure 12: Workflow of json_to_adb.py

6 Anonymity Analysis

Anonymity analysis will be conducted on the ArangoDB graph of Monero transactions that we have built. The analysis will be conducted through the use of 0-mixin analysis, taking advantage of transaction inputs that have only 1 ring-member and do not use mixins to conceal the true input.

This analysis will be conducted in 2 phases: the first phase, the primary analysis, will mark the only member of the ring signature input of 0-mixin inputs as the real input, as well as the output from which that member originated from, as spent. In the second phase, the secondary analysis, we will filter out inputs where all but 1 member of the ring signature has been marked as spent. We can then conclude with certainty that the only unspent member is the true input, and mark it as such. Once again, we will mark the output that this member originates from as spent. We then repeated this for as long as the project continues, or until no further inputs can be de-anonymized, whichever is earlier.

6.1 Preparation

Before we can proceed with the anonymity analysis, we need to do some modifications to our ArangoDB schema. Looking at ArangoDB schema V3 in the previous section, one might realize that the only way to iterate through inputs to find 0-mixin inputs is to iterate through the *txs* vertex collection, which then points to the *inputs* vertex collection. However, due to the sheer size of the *txs* vertex collection, it is simply not practical to ask the database for the full list of *txs* keys, and them call the database to return the objects. We need a way to iterate through the transactions by using for loop of numbers, not keys, which are not in sequential order. As block height is in running order, and each block can reference transactions, iterating through block heights is a great way to indirectly iterate through transactions and eventually, inputs. Thus, a Python script, *create_blocks.py* is used to add the vertex collection blocks to the ArangoDB graph. This vertex collection will have the integer property, *blk_height*, and will have 2 edges, *has_cb_tx* and *has_tx* connecting it to *cb_txs* and *txs* respectively.

In addition, the following changes need to be made in order to mark true inputs and spent outputs: the addition of Boolean properties *true_input*, *spent* and *deanon* to vertex collections *ring_members*, *outputs* and *inputs* respectively. The *deanon* property simply stores whether an input has been successfully de-anonymized (either in the current or previous rounds of anonymity analysis). The updated, and final schema is reflected below:

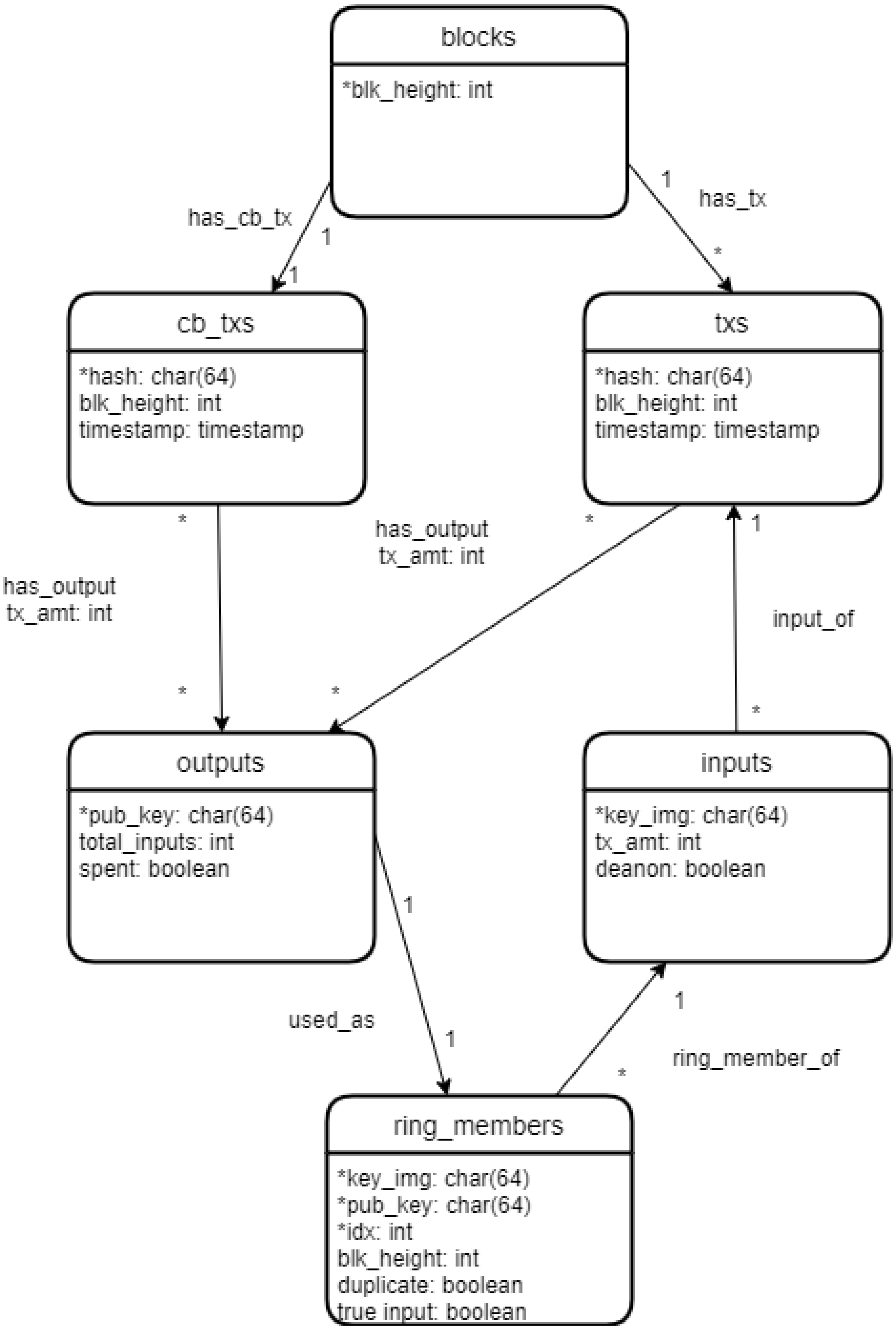
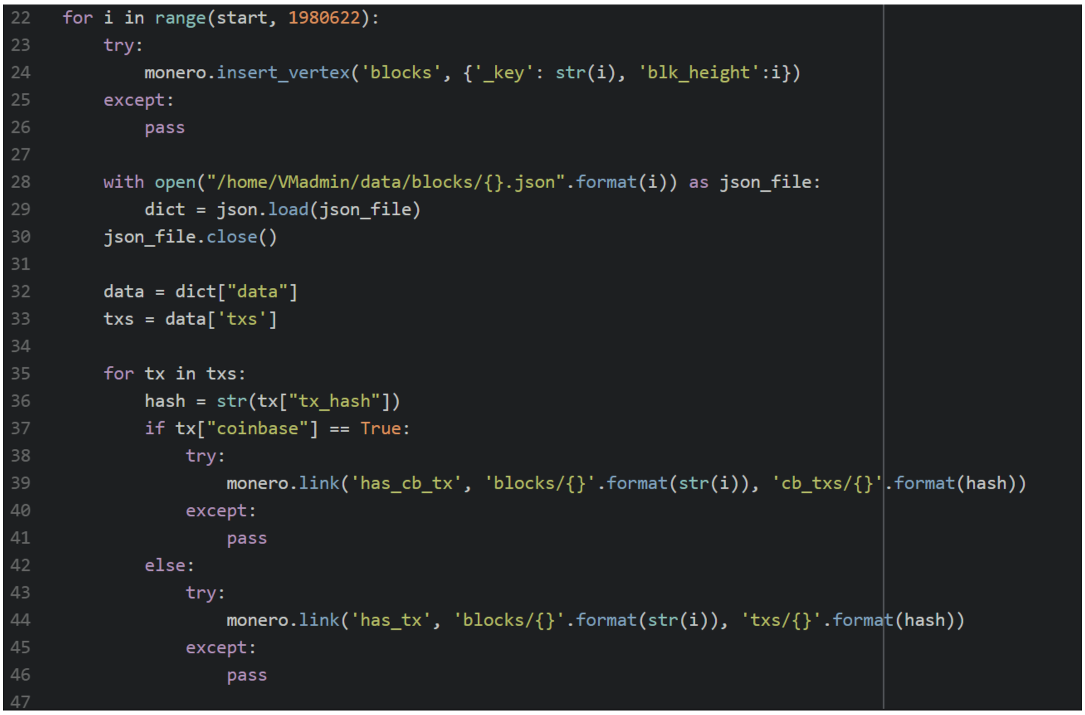


Figure 13: ArangoDB Schema V4

6.2 Primary Analysis

While the block vertex collection, and its corresponding edge collections are created in the earlier section, the creation of the actual vertex and edge documents will be done just prior to the first round of anonymity analysis, in the same Python script. This is done to save runtime. A code snippet of the script, update_adb.py, showing the creation of the appropriate documents is attached below, with the full version in the appendix.



```python
for i in range(start, 1980622):
    try:
        monero.insert_vertex('blocks', {'_key': str(i), 'blk_height':i})
    except:
        pass

    with open("/home/VMadmin/data/blocks/{}.json".format(i)) as json_file:
        dict = json.load(json_file)
    json_file.close()

    data = dict["data"]
    txs = data['txs']

    for tx in txs:
        hash = str(tx["tx_hash"])
        if tx["coinbase"] == True:
            try:
                monero.link('has_cb_tx', 'blocks/{}'.format(str(i)), 'cb_txs/{}'.format(hash))
            except:
                pass
        else:
            try:
                monero.link('has_tx', 'blocks/{}'.format(str(i)), 'txs/{}'.format(hash))
            except:
                pass

```

Figure 14: Creating the relevant ArangoDB documents

The code logic for the primary anonymity analysis is too long to be shown as a snippet, and the source code of update_adb.py should be referenced in order to gain an understanding of the implementation of the primary anonymity analysis.

6.2.1 Setup

Similar to the process of writing data from JSON files to ArangoDB, the script for primary analysis is expected to have a long runtime of days to weeks, and hence we will also need text files to store and retrieve variables, as well as a clean-up script. A Python script, *update_adb.py* will be used to update the ArangoDB schema from V3 to V4, as well as perform primary analysis. This time, however, there needs to be 3 variables that need to be stored and retrieved from text files – the index of the for loop, which will be stored and retrieved from *progress.txt*, the total number of inputs so far, which will be stored and retrieved from *inputs_count.txt*, as well as the total number of traceable inputs so far, which will be stored and retrieved from *deanon_inputs.txt*. There will also be a clean-up script, *update_adb_cleanup.py*.

We also managed to automate the process of cleaning up and restarting the *update_adb.py* script. This is achieved through another Python script, *check_status_restart.py*, which sends the command 'pgrep -f update_adb.py'. If there is no Process ID returned, it means that the *update_adb.py* script is not running. We then check *progress.txt* to see if the script has stopped running because the for loop index has reached the last value. If that is not the case, we then call the *update_adb_cleanup.py* script, before restarting the *update_adb.py* script. We set the *check_status_restart.py* to run every minute via crontab, thereby minimizing the downtime of the *update_adb.py script*.

In addition, we also implemented a Telegram bot which reports on the progress of the *update_adb.py* script. This is done by calling the *bot.py* script every 4 hours through crontab. The *bot.py* script then reads from *progress.txt*.

The overall setup is summarized in the figure below:

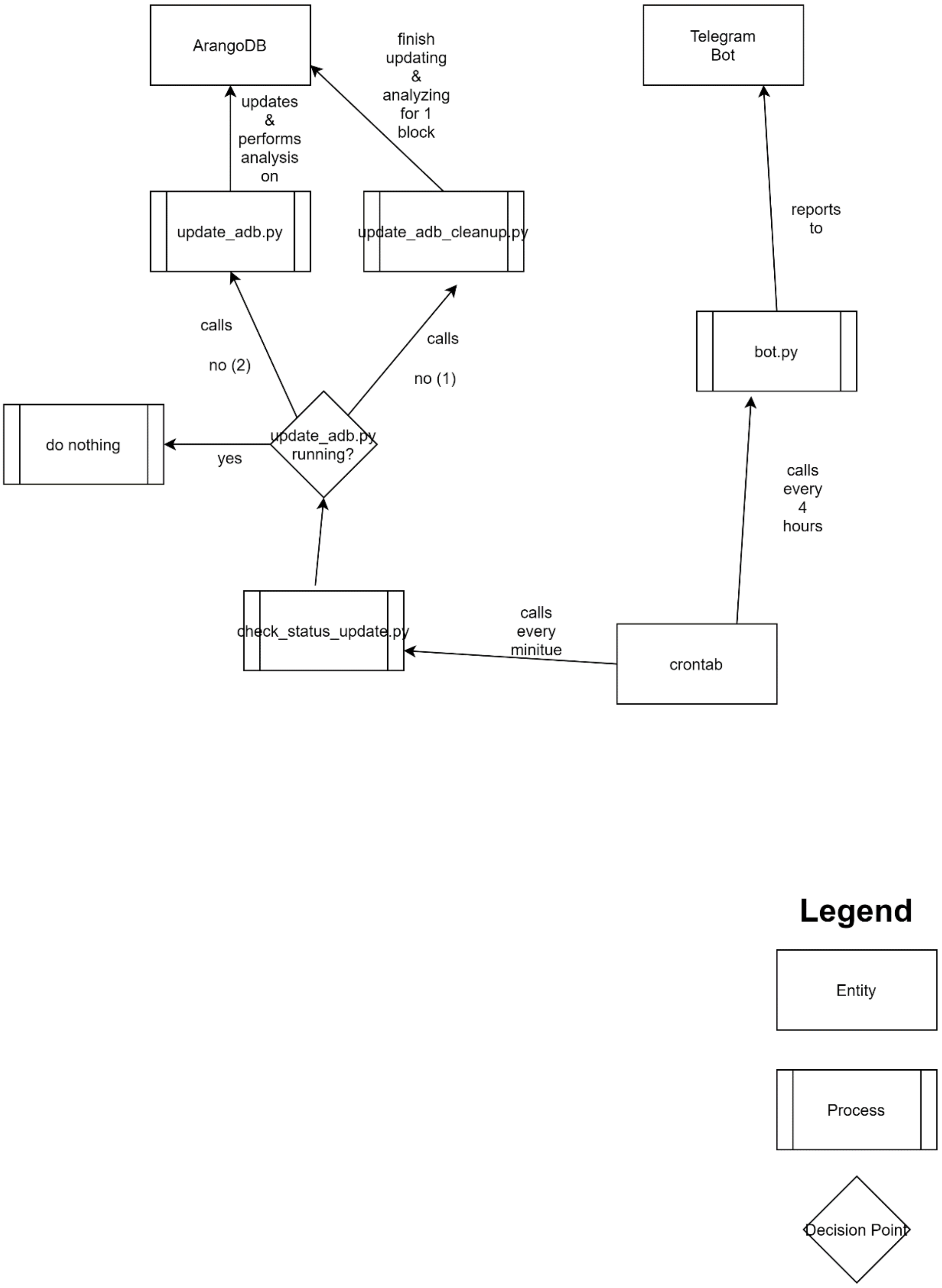


Figure 15: The overall setup of performing primary analysis,

6.2.2 Results

At the end of primary analysis of the first 1980622 blocks, the ratio of traceable inputs is 0.399738639. Unfortunately, the Python script for primary analysis did not include code logic to store the deanonymization ratio at regular block intervals, so we have to rely on data that is reported from the Telegram bot. The following is the plot of cumulative deanonymization ratio (CDR) across block heights.

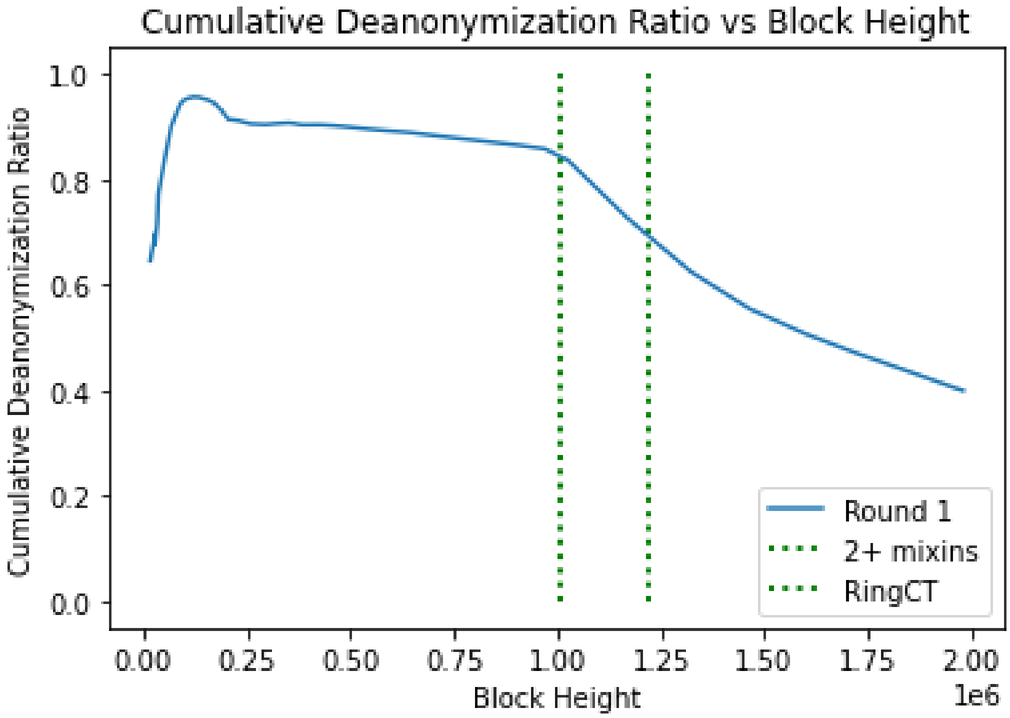


Figure 16: Cumulative Deanonymization Ratio vs Block Height

As we can see from the above figure, the CDR drops substantially after a minimum of 2 mixins is enforced, and even more so after RingCT is implemented. Since primary anonymity analysis relies solely on 0-mixin inputs, it should not come as a surprise that CDR drops after 2+ mixins is enforced, since there should be no more 0-mixin inputs after that.

6.3 Secondary Analysis

Using a similar setup to that in 6.1, we perform secondary analysis as discussed in section 6. The Python scripts involved, anon_analysis.py and anon_analysis_cleanup.py are available in full in the appendix.

Unfortunately at the time of writing of this report, we have only managed to perform secondary analysis till block 1768000. However, this time, we managed to collect CDR data at a regular interval of every 1000 blocks. The results will be discussed in the section below.

6.3.1 Results

Secondary analysis ended with a CDR of 0.551860605 at block 1768000. The following is a plot of CDR over block height, compared with CDR from primary analysis.

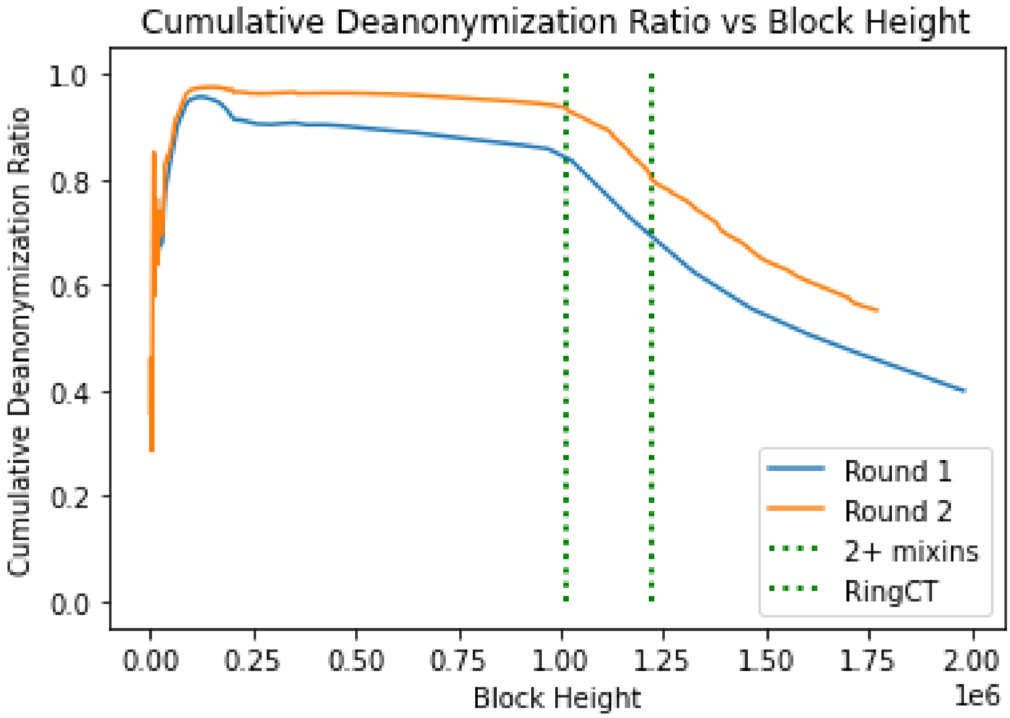


Figure 17: CDR vs Block height, Round 1 vs Round 2

As expected, the CDR obtained from secondary analysis is greater than or equal to that obtained from primary analysis across all intervals.

7 Conclusion

In conclusion, Monero is not as private and untraceable as we would like to think it is. However, as a minimum number of mixins has been enforced, and the number of non-0-mixin inputs increase and becomes a larger proportion of inputs over time, the traceability of Monero should decrease over time. Increasing the minimum number of mixins, while increasing transaction sizes, could help decrease the traceability of the Monero network.

References

[1] A. Narayanan, J. Bonneau, E. Felten, A. Miller and S. Goldfeder, Bitcoin and cryptocurrency technologies: a comprehensive introduction, Princeton: Princeton University Press, 2006.

[2] R. C. Merkle, “Method of providing digital signatures”. US Patent US4309569A, 01 05 1982.

[3] R. Kumar and R. Tripathi, “Implementation of Distributed File Storage and Access Framework using IPFS and Blockchain,” in *2019 Fifth International Conference on Image Information Processing (ICIIP)*, 2019.

[4] N. v. Saberhagen, “CryptoNote v 2.0,” October 17 2013. [Online]. Available: https://bytecoin.org/old/whitepaper.pdf. [Accessed 9 March 2021].

[5] “RandomX is a new Proof-of-Work (PoW) algorithm used where decentralisation matters,” 5 June 2019. [Online]. Available: www.moneroouttreach.org. [Accessed 9 March 2021].

[6] X. Fan, “Faster Dual-Key Stealth Addresses for Blockchain-Based Internet of Things Systems,” in *Blockchain – ICBC 2018.*, 2018.

[7] “Towards Preserving Privacy and Security in Blockchain,” in *Essentials of Blockchain Technology*, CRC Press, 2020, p. 111.

[8] R. Rivest, A. Shamir and Y. T. Kalai, “How to leak a secret,” *Lecture Notes in Computer Science,* vol. 2248, p. 552–565, 2001.

[9] "Ring Signature," [Online]. Available: https://www.getmonero.org/resources/moneropedia/ringsignatures. [Accessed 15 03 2021].

[10] S. Noether, "Ring Confidential Transactions," Monero Research Labs.

[11] 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," *Proceedings on Privacy Enhancing Technologies,* vol. 3, pp. 143-163, 2018.

Appendix

1.1 Sample Block JSON data

```
{
    "status": "success",
    "data": {
        "hash":
"771fbcd656ec1464d3a02ead5e18644030007a0fc664c0a964d30922821a8148",
        "timestamp": 1397818193,
        "timestamp_utc": "2014-04-18 10:49:53",
        "block_height": 1,
        "current_height": 2162068,
        "txs": [
            {
                "payment_id": "",
                "tx_hash":
"52578a3816ec18ca6db2ec4f594b7c8a778caa4c52d2c1705bcbab9798a9ea7b",
                "tx_version": 1,
                "mixin": 0,
                "extra":
"012a9fca96074c5216f9622c58c5c95024e53ff579d128913548363cb14d7f6374"
,
                "tx_fee": 0,
                "payment_id8": "",
                "xmr_inputs": 0,
                "rct_type": 0,
                "coinbase": true,
                "xmr_outputs": 17592169267200,
                "tx_size": 383
            }
        ],
        "size": 383
      }
}
```

1.2 Sample Transaction JSON data

```
{
    "status": "success",
    "data": {
        "payment_id": "",
        "inputs": [
            {
                "amount": 7000000000000,
                "mixins": [
                    {
                        "public_key":
"de00acad5a0df1c52ef51637cb89ae1c991c877acf6152252529009d6e51adbc",
                        "block_no": 2
                    },
                    {
                        "public_key":
"1b6367f72a1cdbc7a21aa37e0ab2155529e404c2efaadd72ca7702e42bc96640",
                        "block_no": 6
                    },
                    {
                        "public_key":
"1e4f2708aa04f52d4607d98ba18bf0f87b5045ff74df71f45649975093d19a12",
                        "block_no": 7
                    },
                    {
                        "public_key":
"d1468a64e2703489fcd7d759bb0ca2a93d4acbdda3aaa77c103f5eb4424ed6b9",
                        "block_no": 24
                    },
                    {
                        "public_key":
"feca0b1c0266f02eed4fb19f97bc077171de836d5dcca99280367f9c94ed05e8",
                        "block_no": 31
                    },
                    {
                        "public_key":
"3c65dd846c83fb48036cd978d4d40c35065de407d20df34234332e5db49c6fde",
                        "block_no": 32
                    }
                ],
                "key_image":
"f254220bb50d901a5523eaed438af5d43f8c6d0e54ba0632eb539884f6b7c020"
            }
        ],
        "tx_hash":
"beb76a82ea17400cd6d7f595f70e1667d2018ed8f5a78d1ce07484222618c3cd",
        "tx_version": 1,
        "rct_type": 0,
        "extra":
"01d34f90ac861d0ee9fe3891656a234ea86a8a93bf51a237db65baa00d3f4aa196"
,
        "outputs": [
            {
                "public_key":
"f9c7cf807ae74e56f4ec84db2bd93cfb02c2249b38e306f5b54b6e05d00d543b",
                "amount": 9000000
            },
            {
                "public_key":
"b6abb84e00f47f0a72e37b6b29392d906a38468404c57db3dbc5e8dd306a27a8",
                "amount": 90000000
            },
            {
                "public_key":
"cfc40a86723e7d459e90e45d47818dc0e81a1f451ace5137a4af8110a89a35ea",
                "amount": 900000000
            },
            {
                "public_key":
"6b19c796338607d5a2c1ba240a167134142d72d1640ef07902da64fed0b10cfc",
                "amount": 9000000000
            },
            {
                "public_key":
"1f6f655254fee84161118b32e7b6f8c31de5eb88aa00c29a8f57c0d1f95a24dd",
                "amount": 90000000000
            },
            {
                "public_key":
"3321af593163cea2ae37168ab926efd87f195756e3b723e886bdb7e618f751c4",
                "amount": 900000000000
            },
            {
                "public_key":
"95ed2b08d1cf44482ae0060a5dcc4b7d810a85dea8c62e274f73862f3d59f8ed",
                "amount": 1000000000000
            },
            {
                "public_key":
"dc50f2f28d7ceecd9a1147f7106c8d5b4e08b2ec77150f52dd7130ee4f5f50d4",
                "amount": 5000000000000
            }
        ],
        "timestamp_utc": "2014-04-18 12:13:09",
        "tx_fee": 1000000,
        "payment_id8": "",
        "block_height": 110,
        "xmr_inputs": 7000000000000,
        "confirmations": 2144114,
        "mixin": 6,
        "timestamp": 1397823189,
        "coinbase": false,
        "current_height": 2144224,
        "xmr_outputs": 6999999000000,
        "tx_size": 776
    }
}
```

2 Source Code for get_json.py

```python
'''Script to get json objects from xmrchain.net'''
import requests
import json
import os
import datetime
from time import sleep
from requests.adapters import HTTPAdapter
from requests.packages.urllib3.util.retry import Retry
```

```
import subprocess

#parsed at block 2112472, approx 03-06-2020 1952hrs

session = requests.Session()
retry = Retry(connect=3, backoff_factor=0.5)
adapter = HTTPAdapter(max_retries=retry)
session.mount('http://', adapter)
session.mount('https://', adapter)

print("Starting script at {}".format(datetime.datetime.now()))

for i in range(2112473):    #monero block height starts at 0
    response =
session.get("https://xmrchain.net/api/block/{}".format(str(i)))
    block = json.loads(response.content)

    with open('blocks/{}.json'.format(i), 'w') as f:    #dump block
data as json
        json.dump(block, f, indent=4)

    data = block["data"]
    txs=[]

    if not os.path.exists("txs/{}".format(i)):
        os.makedirs("txs/{}".format(i))

    for tx in data["txs"]:
        txs.append(tx["tx_hash"])

    for hash in txs:
        response =
session.get("https://xmrchain.net/api/transaction/{}".format(hash))
        tx = json.loads(response.content)
        with open('txs/{}/{}.json'.format(i, hash), 'w') as f:
#dump tx data as json
            json.dump(tx, f, indent=4)

    #Archive tx directories after writing json files to save on
inodes
    zip_command = ["tar", "-zcf", "txs/{}.tar.gz".format(i),
"txs/{}/".format(i)]
    subprocess.call(zip_command)
    rm_command = ["rm", "-r", "txs/{}".format(i)]
    subprocess.call(rm_command)
```

3.1 Source Code for json_to_adb.py

```
'''
Script to dump json data into ArangoDB.
To be run from directory ~/data
'''
import subprocess
import os
import json
import csv
from arango import ArangoClient

with open("progress.txt", "r") as text_file:
    start = int(text_file.readline())+1
text_file.close()


#Connect to db and load graph
client = ArangoClient(hosts='http://localhost:8529')
db = client.db('monero', username='root', password=[redacted])
monero = db.graph('monero')

for i in range(start, 2112473):   #Final value: 2112473
    #Extract json files from archive
    extract_cmd = ["tar", "-zxf", "txs/{}.tar.gz".format(i)]
    subprocess.call(extract_cmd)

    #Handle missing json files

    if not os.path.isdir("txs/{}".format(i)) or
len(os.listdir("txs/{}".format(i)))==0:

        with open('missing_data_log.txt', 'a') as logfile:
            logfile.write('Missing files at block{}\n'.format(i))
        logfile.close()
        with open("missing_data.txt", "w") as file:
            file.write(str(i))
        file.close()
        re_get_cmd = ["python", "re_get_json.py"]
        subprocess.call(re_get_cmd)
        extract_cmd = ["tar", "-zxf", "txs/{}.tar.gz".format(i)]
        subprocess.call(extract_cmd)
        txs = os.listdir("txs/{}".format(i))
        if len(txs)>0:
            with open('missing_data_log.txt', 'a') as logfile:
                logfile.write('Successfully re-got json tx files\n')
            logfile.close()

    txs = os.listdir("txs/{}".format(i))

    for tx in txs:
        with open("txs/{}/{}".format(i,tx)) as json_file:
            dict = json.load(json_file)
        data = dict["data"]

        hash = data["tx_hash"]
        blk_height = data["block_height"]
        timestamp = data["timestamp"]

        if data["coinbase"] == True:
            monero.insert_vertex('cb_txs', {'_key': str(hash),
'hash': hash, 'blk_height': blk_height, 'timestamp': timestamp})
        else:
            monero.insert_vertex('txs', {'_key': hash, 'hash':hash,
'blk_height': blk_height, 'timestamp': timestamp})

        if data["outputs"]:
            for output in data["outputs"]:

                pub_key = output["public_key"]
                amt = output["amount"]
                outputs = monero.vertex_collection('outputs')

                #If pubkey already exists, add new edge with tx
amount, and update total inputs.
                if outputs.has(str(pub_key)):

                    existing_total_inputs =
outputs.get(str(pub_key))['total_inputs']
                    new_total_input = existing_total_inputs + amt

                    if data["coinbase"] == True:
                        monero.link('has_output',
'cb_txs/{}'.format(str(hash)), 'outputs/{}'.format(str(pub_key)),
data = {'total_inputs': amt})
                    else:
                        monero.link('has_output',
'txs/{}'.format(str(hash)), 'outputs/{}'.format(str(pub_key)), data
= {'total_inputs': amt})

                    outputs.update({'_key': str(pub_key), 'pub_key':
pub_key, 'total_inputs': new_total_input})

                else:
                    outputs.insert({'_key': str(pub_key), 'pub_key':
pub_key, 'total_inputs': amt})
                    if data["coinbase"] == True:
                        monero.link('has_output',
'cb_txs/{}'.format(str(hash)), 'outputs/{}'.format(str(pub_key)),
data = {'total_inputs': amt})
                    else:
                        monero.link('has_output',
'txs/{}'.format(str(hash)), 'outputs/{}'.format(str(pub_key)), data
= {'total_inputs': amt})

        else:
            with open('zero_output_txs.csv', 'a') as zo_file:
                csv_writer = csv.writer(zo_file, delimiter=',',
quotechar='"', quoting=csv.QUOTE_MINIMAL)
                csv_writer.writerow([blk_height, hash])
            zo_file.close()



        if data["inputs"]:
            for input in data["inputs"]:
                ki = input["key_image"]
                amt = input["amount"]
                monero.insert_vertex('inputs', {'_key': str(ki),
'key_img': ki, 'tx_amt': amt})

                monero.link('input_of', 'inputs/{}'.format(str(ki)),
'txs/{}'.format(str(hash)))

                mixins = input["mixins"]
                pks = [mixin['public_key'] for mixin in mixins]

                for idx, mixin in enumerate(mixins):

                    pk = mixin["public_key"]
                    blk_height = mixin["block_no"]

                    if pks.count(pk)>1:
                        duplicate = True
                    else:
                        duplicate = False

                    key = str(ki)+','+str(pk)+','+str(idx)
                    monero.insert_vertex('ring_members', {'_key':
key, 'key_img': ki, 'pub_key': pk, 'idx': idx, 'blk_height':
blk_height, 'duplicate': duplicate, 'true_input': False})

                    monero.link('ring_member_of',
'ring_members/{}'.format(key), 'inputs/{}'.format(ki))
                    monero.link('used_as',
'outputs/{}'.format(str(pk)), 'ring_members/{}'.format(key))

        json_file.close()


    #Archive tx directories after dumping json files to save on
inodes
    zip_command = ["tar", "-zcf", "txs/{}.tar.gz".format(i),
"txs/{}/".format(i)]
    subprocess.call(zip_command)
    rm_command = ["rm", "-r", "txs/{}".format(i)]
    subprocess.call(rm_command)

    with open("progress.txt", "w") as text_file:
        text_file.write("{}".format(i))
    text_file.close()
```

3.2 Source Code for cleanup.py

```
import subprocess
import os
import json
from arango import ArangoClient
import arango
import csv

#Connect to db and load graph
client = ArangoClient(hosts='http://localhost:8529')
db = client.db('monero', username='root', password=[redacted])
monero = db.graph('monero')

with open("progress.txt", 'r') as text_file:
    i = int(text_file.readline())+1
text_file.close()


#Extract json files from archive
extract_cmd = ["tar", "-zxf", "txs/{}.tar.gz".format(i)]
subprocess.call(extract_cmd)

#Handle missing json files
if not os.path.isdir("txs/{}".format(i)) or
len(os.listdir("txs/{}".format(i)))==0:

    with open('missing_data_log.txt', 'a') as logfile:
        logfile.write('Missing files at block{}\n'.format(i))
    logfile.close()
    with open("missing_data.txt", "w") as file:
        file.write(str(i))
    file.close()
    re_get_cmd = ["python", "re_get_json.py"]
    subprocess.call(re_get_cmd)
    extract_cmd = ["tar", "-zxf", "txs/{}.tar.gz".format(i)]
    subprocess.call(extract_cmd)
    txs = os.listdir("txs/{}".format(i))
    if len(txs)>0:
        with open('missing_data_log.txt', 'a') as logfile:
            logfile.write('Successfully re-got json tx files\n')
        logfile.close()

txs = os.listdir("txs/{}".format(i))

for tx in txs:
    with open("txs/{}/{}".format(i,tx)) as json_file:
        dict = json.load(json_file)
    data = dict["data"]

    hash = data["tx_hash"]
    blk_height = data["block_height"]
    timestamp = data["timestamp"]

    if data["coinbase"] == True:
        try:
            monero.insert_vertex('cb_txs', {'_key': str(hash),
'hash': hash, 'blk_height': blk_height, 'timestamp': timestamp})
        except Exception as e:
            print(e)
    else:
        try:
            monero.insert_vertex('txs', {'_key': hash, 'hash':hash,
'blk_height': blk_height, 'timestamp': timestamp})
        except Exception as e:
            print(e)

    if data["outputs"]:
        for output in data["outputs"]:

            pub_key = output["public_key"]
            amt = output["amount"]
            outputs = monero.vertex_collection('outputs')

            #If pubkey already exists, add new edge with tx amount,
and update total inputs.
            if outputs.has(str(pub_key)):

                existing_total_inputs =
outputs.get(str(pub_key))['total_inputs']
                new_total_input = existing_total_inputs + amt

                if data["coinbase"] == True:
                    try:
                        monero.link('has_output',
'cb_txs/{}'.format(str(hash)), 'outputs/{}'.format(str(pub_key)),
data = {'total_inputs': amt})
                    except Exception as e:
                        print(e)
                else:
                    try:
                        monero.link('has_output',
'txs/{}'.format(str(hash)), 'outputs/{}'.format(str(pub_key)), data
= {'total_inputs': amt})
                    except Exception as e:
                        print(e)
                try:
                    outputs.update({'_key': str(pub_key), 'pub_key':
pub_key, 'total_inputs': new_total_input})
                except Exception as e:
                    print(e)

            else:
                try:
                    outputs.insert({'_key': str(pub_key), 'pub_key':
pub_key, 'total_inputs': amt})
                except Exception as e:
                    print(e)
                if data["coinbase"] == True:
                    try:
                        monero.link('has_output',
'cb_txs/{}'.format(str(hash)), 'outputs/{}'.format(str(pub_key)),
data = {'total_inputs': amt})
                    except Exception as e:
                        print(e)
                else:
                    try:
                        monero.link('has_output',
'txs/{}'.format(str(hash)), 'outputs/{}'.format(str(pub_key)), data
= {'total_inputs': amt})
                    except Exception as e:
                        print(e)

    else:
        with open('zero_output_txs.csv', 'a') as zo_file:
            csv_writer = csv.writer(zo_file, delimiter=',',
quotechar='"', quoting=csv.QUOTE_MINIMAL)
            csv_writer.writerow([blk_height, hash])
        zo_file.close()

    if data["inputs"]:
        for input in data["inputs"]:
            ki = input["key_image"]
            amt = input["amount"]
            try:
                monero.insert_vertex('inputs', {'_key': str(ki),
'key_img': ki, 'tx_amt': amt})
            except Exception as e:
                print(e)
            try:
                monero.link('input_of', 'inputs/{}'.format(str(ki)),
'txs/{}'.format(str(hash)))
            except Exception as e:
                print(e)

            mixins = input["mixins"]
            pks = [mixin['public_key'] for mixin in mixins]

            for idx, mixin in enumerate(mixins):

                pk = mixin["public_key"]
                blk_height = mixin["block_no"]

                if pks.count(pk)>1:
                    duplicate = True
                else:
                    duplicate = False

                key = str(ki)+','+str(pk)+','+str(idx)
                try:
                    monero.insert_vertex('ring_members', {'_key':
key, 'key_img': ki, 'pub_key': pk, 'idx': idx, 'blk_height':
blk_height, 'duplicate': duplicate})
                except Exception as e:
                    print(e)
                try:
                    monero.link('ring_member_of',
'ring_members/{}'.format(key), 'inputs/{}'.format(ki))
                except Exception as e:
                    print(e)
                try:
                    monero.link('used_as',
'outputs/{}'.format(str(pk)), 'ring_members/{}'.format(key))
                except Exception as e:
                    print(e)

    json_file.close()


#Archive tx directories after dumping json files to save on inodes
zip_command = ["tar", "-zcf", "txs/{}.tar.gz".format(i),
"txs/{}/".format(i)]
subprocess.call(zip_command)
rm_command = ["rm", "-r", "txs/{}".format(i)]
subprocess.call(rm_command)

with open("progress.txt", "w") as text_file:
    text_file.write("{}".format(i))
text_file.close()
```

4 Zero Output Transactions

Table 1: Complete List of Zero Output Transactions

Block Height	Transaction Hash
202612	fe4b08727d96d282184ff6667991b377c1c744e62d281068e8791f011dec4c57
202612	ef42a0ebfb06e029b0bea886920ca35b2673982ac6c8e137f55368923d17884e
202612	2ca13518b91e0526e1b0ed4c0e1e78a10f6ffea0cdaffdf6de62d8a57302dfb4
202612	577ed79b3aed71d4ded13145ce554a75abe6a11a1d1f95eb85f1a15c9c4b539c
202612	54c5a623fd32a3a97b0bdc5303e6d0354bdded5c82a6b0b863b811a1c2627a38
202612	71f927d6235a5ec86598d3ec285d31f8897b97550db8a3347d20235c9cc3e6d8
202612	6cbe2e5a048dc7088e6cce38ab1d4774da7dc6c85be3d1178d41aa75a1180824
202612	84cfd8f79b260109021f2c85373173ab8b59bad3f55ddea4bb1514c0c1448536
202612	cb65e3dbc36a4c756d742b30bc1877b4df8132ecc899608a4d96f1fae0a34b96
202612	5322b5a07660d7bc266f38d66a9b485123e69467b7abd10c59e9afa29b5fccf7
202612	290a70f599936b97d9682c490d0f74469295912e53de9ba36c3311449df36f80
202612	b5b742b3639e8677e197d29f5e0e45c887994d2710d1614201b68b255180f521
202612	1db8e164de5f4bcb2c0ffd1601bcda6fdd7f8457c51f06f4af2ea2a65844292c
202612	2690de1beac7e3d99c224b8395874f5c18f8a0911317f43584f209eb59e33431
202612	073b66aea877496d27dd61f9231ae835d66a0c8beddb21fb953caef8b1c6868a
202612	7ca27510637674c373e463b45170e603e5e38171a3b022f499ac44ffeea4ede9
202612	2d1b3752f130bb9a3d7ac04e0526a2db6f24664250b93fd8799e737bf62813b7
202612	9e29b0a51f61cc2bf162d9f82ae22b429670a8a6ebf1a909628c53ae2798b656
202612	96b05f723620a7a72e7da9b0cb43e1ab842dcbc17959247d944c74b9ef380c51
202612	54b7d694e18ae2c04bf24cabd236ab9d1504b0061d48477e8581aecf4f855eec
202612	cb27ff9bda1998b608311e6a7d1edbf1c83aff291257f03b4e33bbea150a370f

202612	76f21afb5f402a4366c65311b8134d8ff41a7566d1628fa8c937c0186dd7f03d
202612	57ca150f68d99eb813b85202a6c09f5884aacee58729afbfc3074ce07517683f
202612	fa142203f6c2f153a3448b18b20314a79c3e64425b9c69cc80b899c2d3c143ec
202612	b78c267ded6c7fccd9cfaa849a7feec0775401c2c8cf5849757aa1de923efa05
202612	02fbc8463ae3f6514a88a53cdcc1bec4aa439df57e084c5ca5cf507fd720301b
202612	4221dac24449d9d67d3ca5dccc443a744a5c3b272fa74a42d348eb7809964dd8
202612	2948e30bf789c414d06bed34a917cfc6a62dcd335a2b983acc999942f4594148
202612	f3c25408e55e53ff7a6aa81551eb295a939f72f9f2272120513df7f3942c0761
202612	73c73e26ab2d4bbe7f749298cd05a54afd814cc7389c358c92e4ca5528498395
202612	dfbdc61da434133aa78f7b38590c0cc64a989b609435444ca125de4e3208359c
202612	c531bccca8d555aa3e5d438a86971846f2d07f21192c604eaf21b25f8b5c695e
202612	33207e284ba346dc3f7f6e7aaa984a085cdb36e3030ff830b26d79cce48d4720
202612	e44f6749da932b62ea700c6eb5e1b32735aa0095e0da1a0db4505ef8e094b408
202612	34757b4181d6a77f8bb97e8d5d2e9db4b781c3f3515b179cee562721b574e51f
202612	0ca8eb6a90c5b90ba05723c18da60d20725d54502529f2486761b4afa257c67e
202612	06f27a6f3435147911981dae7d2b806ececa0e572927ce1451682a6bcd8ffa3d
202612	e7998d5ed9f1bc9e1a4338ff7a8d1a92b6b664a8b162308de29a13e2f7ea83d4
202612	ba6076371e1920af1e6e14fba0b690b5bd4d5f950dc9f828a55e321243a43b8b
202612	e9b57dc41159825dff888371ad4cf8a023676fff9468b3b664f78190167293d6
202612	dbb1333debc933640e6ba5abe580a636bd4508e5bf00e68c85faf5efb9cd5867
202612	aa717003a756f2d917372c5135cd33e924a32044e12a1440f85ad7590140cd51
202612	b59f2814d2bdd41a5f6156c14da5eb0c06be76f6afdbea8496217be019444737
202612	af2216a4df96f9fb629d0aea30ac9ba40c3e6ad8f81ab36c2d004a4ff4d8fe9c
202612	c56a9052d92373e8acc1c7f52a4c9bba4eddef16928da803fe69fd95cf66b041
202612	ef3d8eab3084f17c19dad187b65670dd1a0f04800819f0afcd7cb733b8f65e29
202612	e2cb95fc2f63cdbcb7eed3502b7acc07b09714e6220f421c66389d7dd289d738
202612	862f2141095027802553d235acaadebbc3ee595f53a51df939beadad9e46fc6d
202612	a6166d7657cf1c44ee5474d8d3b3866e571d1ac298194b80ac92fd16dd5fc365
202612	c2f9bc724c7ce9b56947f12390162dae5ec571ca782d9f59aa56817e8d242abd
202612	d5b05f64c030fe06ad5497b115efddfbcd11debd08401c8218c118d60b2d66d1
202612	b077b5008168528a4265da050822d37f39ee37d920a89a04b5820cad71f5125d
202612	5ba89b51b5b41565046420c2215f8b4d6aae1a5ac3d76d5e6531677fcc1dedb7
202612	871d003793be4a129ccdcd6a55c1c5f5f9269923efc8aeae9fef6691bac92987
202612	30d2028790eb706c1d7f2ca439efddaccbcc8dfff38f8ae1bcb08d40a6391019
202612	c7aab9f217fdedce123d1d59aec4348bc1082ee1155bfc7e1d855f9a0618fb6b
202612	79e2b2431a62ed77f069e81351b3ddae218352aec1df772823d67f35b99a8fee
202612	ee79d808f3f95a39e11ed93a92393617d7bf89c78cb8a87d1731da13d5347f9a
202612	2470929d49730ea7795360a54a8046f637ea8f956352fe1a7ead454bd05fe9a3
202612	e8aadb6617f3c37dda37c22791c1687bfe07fa1d1d2c03302c3ecbca0fb65c72
202612	952f0893913181df09f976fab16a1881aafdf22d5c6edb9b630f49ff284ee84d
202612	a3604d1af063189becce88cbacb8cc56a9592a20c11ca0b512566c8c9927e86b
202612	c9d6c869b8141fb3b29644a1b6f09d7a46c401429e45dfec38401b20f3ba42be
202612	97cc3e85bcb7c0ac1b27ea009de1a02e3da71c13dda31b559e7068e93fd17a49
202612	333dd623cd245186be98edde0a3687381231faaa4012eaba0fa49cd72df73d32
202612	ff676a7aa6847a61cc97ce4856fcf2bd3ece00dd0dcaa2cf6bbbe82acd6ddb3e
202612	1afff53b9e0aa6fc2bc912d64fb15ba9574b626ed50c66e7de171d09dbb9b69a
202612	871089a5edbbe5515ce8ad0ca49790e2e59a7585c67b60cffff543e45c8e3a04

202612	8301074787649e7e905ab135a172ab5d0ecac0383f064364b9c7c84b1b621d7c
202612	846e64dbb06769cd43774ecc7bbfc08510ec2feeb03984732a79088e6c1e45d1
202612	99f34952fa352e23dbb95931df1123e32749f4bd50700b99b352b5820c4af578
202612	488d0a8dd835ed1143c750f14b179e8fb5f0b451d00fafe05723dbbf7e5b931f
202612	4b400a63e35d410f175b25e865a225ae7a85d1c5fe1d9467689361efcd49a955
202612	7480ec78eaec6075dbbc1f9dba9ed6e7719cdd508ae00171fd8bd7cbeb922432
202612	2742c25bcaa49d4930e454a49bb15b6a6be04d57dfa50b920dca5c7bd0a70970
202612	d226266378415f181a37ef2833118304cd0e505c229d8e6d6aecba8c35859659
202612	c0dfd58b8bde7e23b5fddccb508843ee1be53b8b28c783ad2c3b302f0335cbc0
202612	64f96c08c21da0fa4048f91f118c89e10707538c89bde3ce48f7bd0e691e6cea
202612	b98a8c8984b3311d0c294845acf13b39fb164c725f938cd52ba3fe0ad64021bd
202612	ba7a52b19a1a834ae037ddb688f1b368b4c945ea50c88941bfe707f75f917691
202612	83a054f48698eab23cdd88ea309fe6608d3a94dc5345e7b196723ef8c16e7b86
202612	3c6ae81848d6779f1556d4aa365805c719f4cc44860d8733633d251c210af65a
202612	01215c58138820199faaa56b540ec0274d2f02080cba82be5c8c00ca088b19e9
202612	7912413b7f8ebe362a4b1485e8566740fc030764164db64ecc6d970f1f51978c
202612	95386e2cdeeaa6eb0d36a13ee67e390c0417432a89a614bdda122680930401f4
202612	8653ca847121d3eaf8a04221f4ad227ab430fae7b7ab73925c1043a59bf7af3f
202612	0e543108a3f901d53234bb3d5b0b3c0c33998dd580968bd47caf245a30a15e7a
202612	e8a860d9e96e8222cd5f794269852359babc557b2079f379d2ea08ea557a0418
202612	07c01f8d0fa885a16b8a719cde738da238b4b7f4c0622a6bb6ce88b09b3204c4
202612	43ade7774efd22c71c92eb34700bd72993383ba264d6caddd49c678ff859ab2e
202612	3f3c7e9cdfb1ec719e943a58db3f9a3748f6b574c9d362851ee9e93d6b8666ae
202612	d0c629b37a1bf258135d7a8e8ddbb72da9122e0c56e506b0da2ea547ad286fbc
202612	c0e53e5a81c13c716b5600c265619b9670e44f6e879b8800ba18000fea4cfebb
202612	90d7ebe6c7cee16d24c9349f0c3b4ccdadae48967266d0001f5115fbe44dc5be
202612	5cc2af701a10bb06075f880cd024d69b3d30dfb80fa6dfebab3b765ff912d421
202612	63ce9d40b49564a840a6286e524195c44c7638d5ab60c808f9fd49ede97aa3c0
202612	ec0032f33f7ff5de39b8a76797a6b9142f2a9e11d8b78dd7a8ac5f0ca089d58f
202612	eb56ec440616fcad9d72ffef8c3092c2d93082181397c2c0fe516ef3b32339fc
202612	16ba10e985bf5aa566964722808119fdbcbf0dd12675c2a1bcbf83ee75e54a46
202612	0b931afbac2a7145f7e74e7bfba306da065f92bae91660652a69f46950625ee8
202612	f4471c9d4108c96f4db7bdeb68b27bc2f30293d027a7ffbbb9206a86461b9ef7
202612	efeac81ddc7f3b2b7cf4a7baca33b8f3f0f38da3abda076fed36bd6ad6f2c917
202612	34bf07dc525d8e416cc82e85809a884f87b819693a720aaf6ee1cbaba6eb0e1f
202612	69912107b38ca3870c0f15eb81595286a3d0e8d5ce54b0c5a9eb92d9b378a9f8
202612	5c5a4a93bc1a20c77c872aec71af8ac11c563d5809a3addf8a6766b5f97b6bdd
202612	6e6e4404bd830e8dc39a90287163c1f594abe1933aef3bb059921667bd8d4dec
202612	eeb76421e3690ec2e48ffc6817ef65a32079f5a30e81992a8df74cb8c708f17b
202612	a14378375cc95a8293e3659493b26f5008818662ecdec7302e4e4d74a0455621
202612	d9487bd9c5f665b1e9584382b9531c8680aeb059103193a94f0dfc2908230bc3
202612	e11f86fcf7efbbfe6218567e269f86f7aa84e21e7442ef1d1198743430a53267
202612	fdf42f2078d8fd1127905cd94ad034b4407383f54301838da88fd7b198dcb59d
202612	16bf4002e5a5c1d1d21cbb3ec879454108b2710030b898d9f139e09a24785363
202612	d4a09ea799b980f7464f6f13ed095b08e9aecaf471085fc2157510ceb9863522
202612	365751dc015a7303d7c5af762793a62f6eb0e373337aff03347b552e8f97ee66

202612	6e04157e5953d6502b5244481a039cc0785018c7f296306bc64aa778ffa98d77
202612	fd1d1f7ea3c395e106f31e409044f731d769bc0d9ae3dca2600ab62c261b5e7b
202612	3d50102352822a7959a6461761482e0789a97675dad23b01ee2612eb4ca47767
202612	a022a5867af095300286bbb3e9c01f8e7a286be7dcdf2ffecf87125c042e8757
202612	235b50431a0a0c23a054137f364a4de18ce701677abeb0033e785d9ea73d7e6c
202612	48e65dd5117e9face756201429c08700dd0e77a63dfbd4f7c42bea07671f2665
202612	0d4d48195a8359b3de164093d0db2df51a65b379aba2cdf170e5a05e6cade3d1
202612	00dbc84e96d00a547132d6268a71f15f0b387a8553ca3cb1e7bc60b33fee1285
202612	d2959930d9a44f955455f0400cecd809fb99fb9884d0103c1918f2ac9b288021
202612	68684c5971ead0aed7220d0e7bbeb55159161de3f37a888a9bbd27c00764159e
202612	ec94570ca062599746c028b7a3bebbf722de2c5933370f99da0e59934bbef6c8
202612	5a11fa0f01ab37a04b91ab3d8e7f05fac7a894ee33296a5ee2b5f74a2502a39a
202612	4a403e1d2b28f9bd2204a24cd0a12ed99d3e171edd89afe54c2ab1a4bd9ec00d
202612	1f73d7ac3e617dbc8135bb610139a2bd53f0b3e4ab722c93ec13e6b5274c089c
202612	2366dfa041e9ce0be812e1c84b17bc613f4d6a89b242990826d30b7b8012d10e
202612	c29f627877f11cab3825881ccf74bebdc49a23ebf6759322660332ac8e3b8a29
202612	268af973df48c1f5c80a9b95c241a6ed27779dd58b0c4e9b8cd50de7ae5d161d
202612	c35e23f40c3f0f424daa94df58788328192a6ad75f8c7d0d4d6388cf560acb17
202612	e4b5832eb35692f132f82ee408ea14f4475ef4d703e9b670aadd45e5681f3328
202612	d9a8313eecfe692b3e935a3b1432d69757a0c8f43783d598fc772d7538d614ff
202612	e041cd6b7f232257001a391ee342d5bed8d2e641e284739d98781527c507eb34
202612	faf1b465426f987f000e75ef43892f31e024e21145c442e40586776bc52980b4
202612	0b0e7eb4f3a9b04b04d34cee48dd91aaef830e90a041869d4526041ad3f35d20
202612	f8f1e24aca97db18f0230b92bc581ac454d4c1e2956f27f7875ec40b0b797fbd
202612	73583e678a352aa965d5978ca0f5385dbf83255bc52401c1859fed2295b8c2a2
202612	b2e48c210596fa90ec1bd1e235cf84edcc90b29b6b6ae3834649fc8701a72e18
202612	29b6b5044a181bf0d645087fea19b5e2f7337d7a1dfb3fd5e2db332ed03f023b
202612	fb9e2263981448f2595575724c9e76356c7e27907ab35d5e69733e45ab1cb4ae
202612	dbd132a9d082d539d39a71cac2903a3ec5c0ba1fe65f7b26309e7aca3f63a58e
202612	b82db1fa094fef7cd453576998f7d39edce77ea31ad3ba846a30a1244d67e26e
202612	697c3da49f2b79ca6ffdd7a349af87331bc80ba0ab1a5e271c7a7aad77132e6c
202612	c4d1b60a67bd552a748572dedf45895ecb46fdf370f7ec5fb3f99a46394c2e6a
202612	1f522a7196749f75f765544cc83c686c5d1647382af337c547a4f6087f6972fc
202612	cefa783d58201eafde42a5819759b41dd3787ddb96a6f468ebf9ebac1e9fa997
202612	7139dc1416cdb106116d5eb712117e1da488487520fb12c0aa3d4ecb736da96b
202612	3ccaac060a6264d5bb2b6a3a06a721fe155e48c23a0047ad187632dc8107dac2
202612	f874b24abfac04981192f6d545505159ca57cd3a3364dbb028fc668cf6b25571
202612	5f6fa10ff935ebd54e7095bbebb4a5714158ed74ec42b8bc46c6b16981459c6b
202612	b378cbad5c329ad6b9203111db87dccda28c521f747b0ddc1464811b7d9a0159
202612	14fb7a678ac92b06e828590d21c578d23d30ce9ee94c15dfaea916cde9e41fae
202612	518c028137de922dc80c9ff037839ebe29638b6fbdd26ed00c29e97f6abf6b9a
202612	306473e9031e0bde6a26cac15eb6d0ede8a238853daca575a1bf1f13b28ebd96
202612	a2376cf635e83a434f076823d8d92d1658a3ab25fad0cb8feed1aee52e65ea64

202612	fc31fe828b80bf12a3e7a5a724cc69ce5504e37dc3218bbd9404c3e4dbb210bc
202612	4c460eaa6a9005786ca511d368ca32e9d0a8bed94961ec77ee57f49982c048cc
202612	7537a166800fb2ddfbf78be078ba9114a2ef5cf8d01cb0d35b0b95a3fe329286
202612	7ff1094881dcaa9fd2f32ef6927e5c2a52c920222fd98c4470a3cf2341bf9c77
202612	da650fa48f7b24f687018a78a540173d69906277bd2a8d2a6647b89f008bf3f6
202612	2cc3490b677c55ffcefbb9ab40a9b93eb0792f05941b9c87cfb76c3c55c456a5
202612	7790ff70cd3038b55cf358d38ae2427f93178a7657d8545f004960e1d9376c37
202612	249b08e5ad12973662e6b3cf71e372b433a9015dbd9b09c8da695a9e5ec581e4
202612	bb888b0f442f51e2f49f0b94d32e0571d4b7556b62e5c0d1a91066c94ff7da85
202612	c0ecc4e6a4a241028df6b95b3cd8778d6b33b54fffc8da025acee6f934dc9cb6
202612	d6a4af1f30e967180e8685ca34773373d345e70f7f91104ff148eb89cb083fa3
202612	6c7f6b9994832df39d860bcf85f3e772f530fab26a5b6bf79e0ce3a53df445b5
202612	d17b893563fda29408e224649fa060188130076acffbcb390cae5e8a1c1aa459
202612	b5c04549512177d8989a3175dd503d5d7cda8da1b68cc4624991a4bd083fdaa6
202612	dd8b7d6c745b6ac16726d3fac28aa58d1613ca87f6d3ef052bb9a231a6a70843
202612	32280d9aaceb72845340ff93fcdd315ac51055473d3a361773b2c508fd116033
202612	a2c5f4f90ef9f8101072e93736f48582a589f3b83c5c55c55492172904ed7bc8
202612	60ab34a1eaf45c9a46b397c1cb3900d5738338081a7fe613d39504072e273068
202612	57e60ae653131976f8736d502b288377ce5dc3adfccb94d0a21e799934afc699
202612	57cb2397b018818a51d63945389b2cc229a164c977ef794886e4887e544de3fc
202612	c561f649f52e395f6f45f7ffd973d696c631c5880c9d800d102f92f5ba079d16
202612	8faac88847153bb9931ceeb6f2864bbd63e8af0b422eafd97c700bd35de7366e
202612	bf193058026ac97521be54e2803cc0b2c7bea00604638d4dcce057a48bcb0239
202612	e176789e2d0e8fd5a36eec628e66731d3665e7db559dcae0464a02962ec969d8
202612	3dd4020a18e5a17f1b5c276066e3f9db04a85948da4009f42a1fb40f8c2cfe3b
202612	69ba1c7e877148e18e7fbf0c900b4987284b0702cc9906066ec350f6b51455b8
202612	1f66f0a8b97add4e8edf8f266d2d90dadad4120f5910992dd5fa49afd346e679
202612	407aee205cd65bc6533bc132c4c31d277a5de2229cbddd841e642db39dead415
202612	e27c85bb3c033ac8866050ddabfa464f2c16254b946fddfd1349cb2de3dbc3d0
202612	4efe9520675c0c2d8ec2ecdfed299839b62869d1907053a2e191c9812c439999
202612	6e9da3eb771ca30256fcb6749875fee537fd83a12f6bdfb09001ec68dc7c85fe
202612	676c0fad1a3ca71c75e2812ebd470d29ed7031c3e7b2c4280315fa1a453760e8
202612	1ad732796e7f90f928499060b0512ffedf27cf7d7e142134488685bcd566f5df
202612	f6dd8418318c7af1001b96edacda0675df665040bf04f701d4fd14531dc1f779
202612	65f3ca95d1bae8e43fbd7d6db327310fa7fe784195d384690154af6bbc3acc2e
202612	db4a8b112eb28fcec3ca590c46031eb41f5a247319c24df0e579550a52839e99
202612	b6b20a995b8a387b143265c90fe839940ca9907bec4d3bf0cc96ae21adcc8aca
202612	5c5a2990718b03b59d28d3db5d2b58e63258d5b69eeed290a26ae8916693b11f
202612	e508ecd9e62cb639263522b6fbb5eca94d4b9cbb0de8826f6b9f2c04a9d085b4
202612	b2ea474241aa35225a2d73f7313ba09ac4dfb64f76f1bab8547a3aa9f9614836
202612	1cbd66b0ad6ede1124311744750f7ea0a1adf0e4b2ca81ba233c72ddb4d1d5af
202612	b021f8cad659bbb87695eb51b0c1bd80d41adfcba93f4a617af29197700bdff4
202612	f8b9481bfa0b806bfc5d16dc6b25199f911ca8b62f9d89a86dbbddd1297b3ef8

202612	fb66be7b5c5390c5eff7538049f789f910e7b2aa92d7273b845f6ba2ec250d0e
202612	86bd9322702c7875e57a212027fd453fedeac75495a719afdf5bc0c23e790818
202612	9e61890a9ec1e0d6bfc51641814e770d6c168f1c56d117fce5fd1d8154bfcc2a
202612	c0029782a704228ac92373b675479305960eb04ac081a28bf413fb92f3bc8a58
202612	61b99ac665fe5f9e12b5f9e346ed69584e44b102fe657a996ba49d3479238d39
202612	0aaae1372ea7f2a12d52e12b9c431f524d1aed43a466a56516ed8e936a904a0f
202612	db84a3aea5fe763e4ea19c2d3d874156001e89eb3bb57e69be71276996fa49d8
202612	23ab1ddf853f51f01b404c42a9f602a5013558ce942630b9d8ea9f6109475547
202612	74f3ba47495f5e3b4c2acd70565f3d6f246c727db5d6b247d61027c3b63d8ad1
202612	05e0bebaf2087bcbf11e1accb179178aac1a01eb8888f219c2b145ef6c4a672e
202612	f5c6513a929b210a3353b4047567a596618da2c3da64ba66600e4f6b53a8b4ff
202612	5bfc97147661116d0cd2f251417132f09694c63ab029dafb28e7d75d9c85412f
202612	b4480fd1b7042eb2820e33d263b2346a3fb613a1b6e23f254ee738e8207523ba
202612	67d123ed72d4333ec3c7f8ce4ad3ee8ec5e0c532a08ff387be336379863a9bd2
202612	853b536d3854c1626889704e9c05022f7ede4aa442b7831896fc50a2f370f682
202612	6f6623da309d802c4fb801db02be7d486635c7b9a2577a60b4fe64f749809896
202612	094f1633b137b040e3912395bbdf664ee790467fa6243c9e068b27e879d431a0
202612	00c721804af7e39dc6e84ef2c69b975f11c5431a1372e231c5a8baa4f238c849
202612	efaca8c29555a849601534cb25e6f146802523801e0ae4183ab1d3be60b74386
202612	a99ebeace3f1e3056ad58e7964beefce19aa80135780d6b8a17ced8a223c4e7a
202612	2f4c5129d6930775132cf24cab421f36bb18e21e9e6b2512723687cede5a92d1
202612	92d0feff43b3552f8d16f102b930f9eedd87511a619376b8af27b48bf4cb3604
202612	a4f136807fb14c4145b4c63ba4ea3914da3f7d66455b51f4e19e983cafd3f646
202612	e0ec734124327833471c88720ba749d7f7a87ef8822b4fd5718670b58e014985
202612	41baa70ed5e253342e6ccc45a8387fb23afde6d7dcb218a584f6408b6ff0c5ef
202612	5bc1f8329df41022c0b32c319d90c5a25a8d235e71bb755bdfa189ee7c12eecd
202612	e85e2d728916fa73af14e6e24a4c28ae93da89d9d227323e4da31c66a9fa4631
202612	5e86d6f79ed1007b9ae52e35006ec19bdd8ac5f6d911ecb484b1e9911bac8592
202612	9390a76a84a76ad19ebb77f244acd4f5c3dff8abfd2e6ad308bd181a155d4f1f
202612	61c48437efdbdc2a0248e33ddd07da5634f944c43d214cabc6df0287fef3adcb
202612	5054ddba88b29ea30833edb7a6d30dfedfd09da1d9d2cdccf31a4faf8d6145a2
202612	e2401a59222d4cb80fb1cade735c40cd6d45afc35d369421edb7228c417c5920
202612	bcbaf43b8421071a37d7b0a195159f5ca8604e3d6d52fccd3f564866de915bc3
202612	168dc17d57a786c76890f0dded9d0ac1c54de94fd29183b403855ddf1604d072
202612	73998a85dd8eceec30e0f9e25286a38fbd866190477d341eddcc2dcc04e1410c
202612	813cede5ff9183af53fb3b8528617a2e609c2142543e2042ea3e8162e86e7045
202612	ba17a8354b1b4521b6cf2033dd56b2594b5ec3f547b97bc7ab27a4c3ee2d7eb8
202612	263f2e9373e12c51d0c979c5da1c70f761b8343c82876f1a667e8b5ade37bfbc
202612	c3ac956f6334ab59226e8dbc3ddbd8ed563b7da0ce3f77775739d39d4ff90599
202612	df9a7e72e0a01d5f59dd24cd0f73d91c8023fd3d534abe1f50082c900346d38e
202612	c086fe305f3d2b604f0ed5dcc3d0fbd80806fc026277ccffd3d4ca79f06f72fd
202612	4d10769784947bedb6f3b551e1563492d706a5179808deb348d7a698cc920c1c
202612	a442df2a3898e8936000906133a18b5ae192a5f1e423523d345103e6b60c0349
202612	47ff0bd14537645dac16d0ff066650f8059fc4bb2c6ffdf98afc437b95c657d2
202612	1edb949f946366842ce7b2917453bc9ef97f5bc6c761542b5a642447f5e29a8d
202612	2a588f0fce4200a54e36add594cfae9e42d1590e96055ff4ce7d054d1ac8c981

202612	210cc14eb552f477850fbe45f3a8580f180b9e157c62c7afbff42166bd5c04f4
202612	50978c35d2220424e6792144b935efae3611fbfac5a9c58e34437991a9ca1fcc
202612	febed11981c8ebd721d43b2e9d4e3a9826e58cebf1d9bcf51d1f013f489f9cf1
202612	2ba663dfe1a1434b6233f5d95db95342be5e368d6ee1d9af821f84dbf672dcfa
202612	6f3810816b9ea45a9a0cde1d14c99f48ff92f3e7384e9f480aeb3bb248a4d7f6
202612	e1a5d6c83235ffe2738424a407e79eefae90d0d13c463f6c6dbeca771f236225
202612	0c05da5aefa823896a014f10cb2c1a5e70a7d8341e8cec540f3ff9fef55af1dd
202612	0458a0a8ec37fd483bf04657469e92885c06973a141aa03ec1e907a90affdc3f
202612	2cdc63a55417d89bb07e92e79304232e3536b981e1b8c109d83d3087c99d1195
202612	e397f35ae6a5e2f71bb50b28c1520be29c9868ed88d7eb3537cbe14a92938ada
202612	6b145e804da95223566eb4409e45ba25ff6f814f2feb1dc79cdc86fc1068f300
202612	cc5a63700bbbd3e45cc167f67159dded4b805b09805a83176181efae5f8a91f0
202612	578511e3b711bbf65f6c475ac0c03a63094d6650a348db3020c2ce644f568ffc
202612	5fda63273f420c7acb3e3f7150d8a3426f78a9a200fcd8f96431dbd2a946cba8
202612	6a01f8dfbc2211959cdc2bce9fa65e465b429b02503e37a6ed066a4e639b13f1
202612	a5daa479a83d1741997bb71bf0c7c66f35cdc84f40b9b34e0bc1728f1686973e
202612	f7552c48fef75fe7c355915423e19521133454001eb2a95e8aef4d5a7220d4f0
202612	66c15eb4816cbb924135cf9258d5e4354dd584019b00ba654aaa378ea5d592d7
202612	aa563b329c37f6009e41c5594b9e9fafdd67aeaee2c6de2a16fbba9a3c0c3358
202612	733f078b23a389b01f3f9f84ff0f86298803a39e62687a851be6369644c38646
202612	1e94c5712e92e2c6540dfd880eef0b4380f6e74903927f700646401fcfe6f2a1
202612	35b4402f2b8ec0ffe207c6c3c0d02f8f0b73feeb27e84a9c0a6aa7fc571d72bb
202612	d7f5ecdbf16082b0e5464a001e848d89eb7bcbd49a2a7e9c3e7c9f2900cc38b9
202612	b03ab5d72baff6952a607e0b0148bc42fa97e1a316119581fec6dc597f5b8d47
202612	ab755dd59d45c21902f98f4ef6a9e597d4444e7f53bc030a96cb8d3cc736a602
202612	f2df7e6a542fdcb67c2fd590842f8404fb2a9ce032acab67a3dcd7afe034e0a5
202612	649f7cd47c379862164001b338724f1596e537a6b6c4c5446a1c29897dfbc7d8
202612	4788712e4d9d873960b9ddd1349f04356eda71f0aab37dca520db9778a4a4866
202612	cc5b22cccd5a355f623dfa976b5c8b01e1b683520914d0065a384c5c693f10a4
202612	8050b4bb588c1e0382e000f062a7d316135606b34768c93818b36b3b8f0d77e8
202612	8b6f31740153db9aefd2a61ba25fdd574e624474474a9593c321393c6bdf11dd
202612	6e933fbb0283d65ec676075a60f8b9a1000144bc60a0a105545dbec201485c8e
202612	b8570e3f41bec6a959f700cb57136b9f86e468d20fee20b1ae2dc0b408fad342
202612	dde396de5c982d8d2030685cc51f01de75c34091dfa41adeb32bd3783501deb6
202612	0a0932925b1c25c009e0037f15c7a2b861c33a1300307cc22a995a5349eb380f
202612	afe13ef4a9274c47f6e0b6330231ecf4caae072ce83b4622bb37cd886f0d6154
202612	4c6e65dffcb9fb17135ca0aa28e30d53b7123a8e57054ef173d7c85dacc59d2e
202612	55036bedc25dfd04f3dc056b9ce6f2d87692240c294e049d608929be850bf89b
202612	34ed49001b64a981751a3c967ddfd8f3a36f15f5354e840a2a899682e0e47af9
202612	1fc5102d8ed8e3aee1d172b98f5d310940509c88e497263712f20fbd827eb5ef
202612	18c38c123a5e200658021e3dd3e0cc8b00f866ab62b889cdb3e511cd145710c6
202612	3a885dd15bdb4dc40ceaebdadd0408cdf455a9cc17358b17fde893157ad51d75
202612	1fdc7ea5deffe526e805c7fc55ba0972ac0fdb8e3c28b408dfcaeaa147128df5
202612	86e4179f4a8ce18ca992737550f0dceb02b57fb3049036eda2ad8159cac5fa0a

202612	3e1ecfeaf17151699d26b7d80984362281102f30b263521fd7a434794ca9c927
202612	df5f87fc8186d35af26983576318602dfe253655e3672e81559edd128f3aeb9b
202612	f8ec78a97b95ef8bc5be7d9933509b5f47c944d627ea5eabaf91906c725a088e
202612	eeb703d1d8a3b13097d9f5c2ccd89e7dd89977bd8ad8139b84f57646fb8bc129
202612	6fb10df20a1ff3d742f4f29c06e2dc7b3517cbc9fba2758fc72b3f59ff974072
202612	a1026c4a9e7c031e1f32c3399fece4100ed37c519cff7924b08ec48b6eb50652
202612	c21c43f1fc94ca840ee219f8e9131e3c1075a9b10d3c16e3f514e88d64c808fb
202612	6993cde84ff1696474e1437279e3665f4e69598ec7e1100fbf00c2fb0536228b
202612	2114c09aa3e859ce281a9738e1a4ec1c1c22abb539b2637d226f33a590b6dcca
202612	d7f5eeb93aa72c6e8665168f4a38dd4e8d8fe30b373241666a516753f97f74c7
202612	cda8d2a47e1153bc5ca0402ab1d274363b2e08f1b322be59d0083b03f21c43db
202612	8c724f47b624ce6b2a4825738f3ce23fa9535255f911ab8706d7b4868c1b1319
202612	d951f9dbbf698e14cca664191609b62277346150122c12d6e12cbe334d91882e
202612	4668dc1a37e3bf6d53412502fdd929157113816ca9c41f6c9b5dd1562eb4aedf
202612	3ebd9e8b954665421149c3cbab118eed74c2b29d7ea3e79c59b37c2c295219e2
202612	f83c1b58360e0a1129c6378e516f615ba0b4f3ac3ddf8baedb38174fffbfea5c
202612	a3854e639dfce55146e33abdec99918811858d1027a2fa100ade9977e236b0a2
202612	0d56891379f3eb346578a6b805fbfd5b90d1e41d324e7d666b50de53299f2ff7
202612	aef286783b744ffa547959125bae87bbe89d74d9c7e82a9ce21f2e9bd1aba31e
202612	360728853e76d8e230694bcae16264d02898bd130c44ae27e110bc2fdb329069
202612	9b2db6e2c57df3896825d973bb6118b1055397a5d7a32ef834d6f02472f195bb
202612	15c86bcd540ef33410937dbf8e6b0f4714a0bf437b296b213bc42a41d249265e
202612	0f7ad7ea8d88bbda31e68d5422c0886857ddcff404baabbc50aa33838a7f7730
202612	15c02f12eaeea0e7e23c17b6cc4cc8d03a26fa46cfa66131f2aa721e53acc0c0
202612	f3ff5fc0a18838ff666987788f034bb958fc27637aa30603f12a28e64a87f2bc
202612	ca21427b29bb999e5a2203c8204f133ffb66faa280ced90afd9aac2d2be759c8
202612	49b9a1700b9b7bbf96810725969a8bb391c8dcc6b37e225b5a2cb8ee9839c1ff
202612	17ce4c8feeb82a6d6adaa8a89724b32bf4456f6909c7f84c8ce3ee9ebba19163
202612	e50dad4876ea7314de6bea40fa6eaf63d6ae45b2cac6ef5b821603780435591e
202612	cb6c87380c3606905547ec0e769b7e578ba1340f22358b2bcdf5f60e8309cf52
202612	c92d01c388cc4e48ff4255214aff6d5e00fb3207788bf7aa8e757268924969d1
202612	e282886a263176c04b9f58b28c11d326ed54abddb841c29cd428e2d3afe24d01
202612	47e085dc6f61f0ad9384a615407d39e909fde96ce5b5164fb31f67b17259d150
202612	4dbdffabf99ba22d1eb488dbe27480c34309f43a1fcce160e2c5a923b89485aa
202612	41dd972b4e84247a243754ddbf0030cfb8549db9c2e03feb2fb70d7b051ae784
202612	9414a5cb5dd7bef16833f26af2bba0ca53338f5940dce4cf9772dcaeb48abb26
202612	bae7a8f6419b850a38fd8110a8940937cab256ac8d3a1723f278df52c5856afb
202612	22c8edea4711558c4338919f255d8ce38b7e8704478216bc30fe117e7a8432ec
202612	ad83b868d1cf27b5e0da32914c2207c49a51d12fe14905d86436ed604c6a009a
202612	2c9d8d5ac27737cb381f86772265f7073a9f05e6c36ee81f3c4d57a531d330f4
202612	b543464c4c8cd7c902a45b8da79d6257a885399992198a4f6019373feee60538
202612	d0e4e8226a2f722768e4ad491f66aab7dc87ef15e17dc47a6b450f9ad8400d4d
202612	d3d31c389a5f74b21810894b8bb05a35564d14cdeeb4198a2a3912f7e22552b5
202612	b95cf65782f4042f73a2cebb7c8e25b155d0115702dd33a7e10099c0541423f0
202612	d0b92d5257c51698be359e6c3cfe916c220e9eb741d49f6d27cc3be47df1f9db

202612	6cf665b8ca5c442a8e22a23620cdb8f3f97b4f21268ae7a135277a579ad65c78
202612	79ec03666367f7bc1d6d9198f9b1c7621c3dbb4adde509e3d022cc7e322ef6b0
202612	2d2792c89114f6c40490abe47e51bd3591b796865b84719033b042fdb2964961
202612	f0c0fc652d308baf362603b3650021bbfc108ff8e8aa6352f7431e18d20e8fb9
202612	428a189562311be8352378395405b161ba22d85849507e6e9ea23cb251e879ca
202612	bd99fc37cc390bb6cc15ea8a81852c77b412189e3716da93b326e6971070e64d
202612	8ffe8cff7791d31ccfb27f942afbbbac65d47ab2e78d12d7e9fafa706c16121f
202612	051eccb6dea2f3caaa0a0acbe7bb7e6d0b26e66ad587f3e36180afe101d3961e
202612	aa86b0676590262666c078b1ec62258bb2951180debe35f3c7bf04bc0f0511b3
202612	54996b125edd2a1b4f91eb97e156b6fd4b23ce2c1db4d9851a5218b26991cff9
202612	29af9d9bc9047b6c4fef5600adab302213e111533041becb4e9465e951e336b4
202612	53cfe99e0a7a72c045351682f3c4bebd5b482e261582f08c1acd197ab8b65131
202612	cf4afffca4fe3aa34b1b812d7ce6e03f9410ad299aadda5115a74ca9c96e2aa7
202612	49ed866f36c9d9ae641e859d76f42512f1eee12e03d2a5e670f0438c3b4eaf94
202612	8837c547b38c640c8b2a81de8a36d7284ac81a771c6d4ac0e1901dc574c5c0ec
202612	b2463e26d046dcd0a1efdb689a756da8d7c28d5ac64193646e591039c8e1ae89
202612	95bbe122e78a302b2ed13604c84522ef792ac155ce88e4bea934b5c78f7be302
202612	09744c976b1ccdae5333e1a97cb1c9a356fc495bdb4f59855d12492a5784b50e
202612	655180dcc153ee26173938676fd540dd748921408da110d1962dfc8f9b693540
202612	a0bb1652ac3eb5e5a2749a5c7383a7b30f604f928d08edc336d44748bcffdaf0
202612	2179720ff40fcbc8e2f2a1d8af7cf0d6d0a676cd2e420aa80f0ac6ad2b84ce39
202612	158420c4ba80409ce88a9e61a186b06485186809736ad469d2acdffe2d2c2a4a
202612	5a82e5cfc1b5ec1b5459035e73613d8bbefc35bb20f3ec337a278400132852e0
202612	6a3f681e37c3f567f0e43c9af0a786d8ada3e526de39e3e1b5ad759d8173a7df
202612	d2b2d28a2e3b33e3179e96fc0e792d574490c62aab1c0b690cca9f9c65a411f2
202612	bfa6eca354a1a83e1acb3ad2607d0cd443afba8b56619549f94b1f9574526f2a
202612	af4e0c2007cde745fced03a45d618b7f1522245f4efcacebee8fadca9eb9060e
202612	d63f8bb8d7b7d53af0dfe91de6f479a54fdfdd2502864bd3d896bfbb3ee8c73a
202612	527024a5186d587275e8c55df758ea43e4c4deb8582a12a6975b405c85566e46
202612	b681cb64f58084d7c360114cc4b7e0b4f7c75bd95a3ec893242223d570c65a73
202612	7847566f1417d085b5648385ea7af7dc3e6e0016d6612ee56a6f6d7dd33154f1
202612	d251413c9ff7d61c381df478b60a13aaf1c7b3652dae3a4ad71772806c080cfc
202612	f76e304b06e99cda9409792b8be55b0458f358bf00e24cedcf8b2805ff86848a
202612	e79cb93179bdb219f133297e89ddfd570312421bbeefd3eae4123d60c5863a7c
202612	0121c50960db03ec9cf1fa27c9c79b135eb42f88d97b90878d3c0646b4a1a1ac
202612	057f38432459d10174f4c5d22dff800df376d12fe44c46bc0e5d78c7b6baea70
202612	a0c35787255d844734a0f14b0cf888a208480ab4097930c61fc0f0ea10210111
202612	a65197a693491d001c540e7d0d4fc873e01d489a3e55ec7afbf07574b6a80bd0
202612	bac1d7d389d67fad90c87031597f4d2f2315c4591961ab88d6bd21f92bf17e84
202612	9546e02aaf3f5bbc1a97623e4467514135609c5fb051ca0762382ef00ef0202f
202612	74f638ab45b92752bf1a1d6e1cdd67fd9b636d8d45b4fcf152285b7c123cb606
202612	9d95e1506d3db36bac94e71e2279f416f5f174c4e0479031c91a2e2d57e8dc27
202612	bd4a0e630f8659ac95ca4fc3a57d9b2d35786a2aa269a6079f32a1f3c80bbc59
202612	64671d0d0f20e422738dab820ba562ca46cbb0df2e3b725edd029da185c7cf57
202612	d7eeaa49ad45ceb1eccfbfd534378fc7e9f2c293a6aee429c0c2240ce6f0e034

202612	39f49f8ec5c75c4a65c7b68adec29aa5f8c3e188e57049b040dcc4f61d076251
202612	fcb2dc4dd43f54fcafaed31b6a1443c5dff8c20540b86edffb615b42f5c7484c
202612	ff182e6e951903b68ece50a747e767907efa2d6163d4e9489af05b9df8668157
202612	1a842bc7f12e03653d70339ee703fec97603b95c17d84be175bb7c77850222ef
202612	3910fe0919c2bfc4cd11ed6a2d74e8ebe6508287a9b116cdb33fb945c92ac2c5
202612	027c068175f56863ed89f74ac1cc8d0179ac0a6a07127486be9a4d4830f69fad
202612	0141e59263b35195d27c9dff9e6021e7a60b5d57ccc700d266e5149254a6b394
202612	98acfd30b3f90089d5620b8f8c3e4e87c51a6d43ec16139f251bfc5c3d85f12a
202612	87792a58f0e9e139c103eb009aaa7baa4c8913305abc6d436384a6a6722bc0da
202612	9929b253a32ba9b97f6f2b74b545fe8d118f1b42234d4428270cf0ecd193442a
202612	3c7a947f3e17c8266a5f37217551ae4f31326c16ddaea0913838bc604b45acc2
202612	848523d7ff671f11f7e91ea84924779722b8cdbf567e74e0e2304afc0ff8fee6
202612	cd07eba8a6a47f260084da727ca8e3ee60aca73d84a0d11b9947a16a071aa769
202612	7c3e65e642fec6d4e623f39afcea7e21997901d52ec340fa9514934d2f0d5b8c
202612	14f5f36277bde23897bd24f1a92afe42c117ec9d9113be370950e04b050a66a5
202612	e462f2328a662060ae7f051284e3b989a8f686039c9221ce7492d252e8be03b2
202612	58ac4ddccfb488895c93fc1f0079579c1b0e7af5b27bf84c6cea52f340505993
202612	e470e6ea166381ba2c38b21f25af3e349c8be32311820e21cc9fd741d7fb93ae
202612	6643076df153042fe83854950578208cc41d37355cee2d8464998e8ce8d9c877
202612	eacd94f74b75b8cf6e4256cb790a8e7188580388b2a9d7a9f38f2ab3e7ba7c43
202612	a674e1456107cb4d2b189029da51434067a9c7985728462e2a2319b1ac7f029a
202612	5f56bb52a57da48146efb92055ccad13450effe19f7cea60fc086a44d0094e5d
202612	f9d95c0b2539e3de2d12026763b88c321ff88c0622b8aa0392b4e6da98579cd2
202612	f06f35591e399a640249433c306d8e185f07a3e9a7858b1a4dd566a292cc6bad
202612	84393e6ff67eb2698da125ef920b4d7707f6c3bc78665fda43814e41339b7c5f
202612	ee6042199ebff10aa497ba9b86221e948b4a8680ac44684054080e5e1d6c3642
202612	a08f1334387d269ff647157e818eef40e77e037fa32577fd20cce5876fedc680
202612	509d3254177e3e1d353a4472a358c8980b52bf0c5e937173ad4627155027b6ee
202612	522c03527dde6024d379f4cef3090e94007f4c966d84900cf86c540b2ae69708
202612	e0cdc46dbbf449252b0b252bc543e71a04fe387aa96c512a8b0dfd8c3114acd1
202612	9723356f7940345cacdf2ce0de28ef58824d82651676846ef21309551f070d27
202612	8fbee0d45a8373291b9c6fda29cc8440ff5c45f183727c75d390b8149abea607
202612	9b7bbc6e5d8857b3c6fcf5d509ea84e806166ecfd7da542029c5f7aa23f59944
202612	f55800d539059de53b57385a5e11efcff3da27b61e2eff1e1ba712d5eeb1d783
202612	123756367c7a80b605e8965a11c40d6c3943830a583a4235076a8bbb39feb499
202612	8d908771d9a6f40e4821ce85e5487072f8ddd1db883a8df565f6b19e52b2b6c1
202612	34188a3b0456bb9f189c9d35497f4a783686202695aa53498a43ecebd383a2e9
202612	5ba12b0f166bd3191dfa55da0093277bc8939803f66fa3cc9df3ec01924b81cb
202612	28f01f99d0a37a7a1c7273f8277257417b1c4053668e283446aa62bf7f5320a1
202612	5c0b2547d4420bb2e0c9269dd90b58755478fe6434711a640080f114ac96993a
202612	4a10a14d341838b77a3e18cca63810c950e4983613d349e14ce35d2df55c66b8
202612	50634c7ba8b6e1fe7b504b99bb69e39f5f940f3c0870e992b03ac3f2523a2a09
202612	9872c0eb538ee3d2698bfa5d0c305c8d7035f182bc795cae078c176e1cd03f1f
202612	9a2ed482573fcf7a96a1949e9ca2553c9c650e04cb5338c98766919381b1b7cf
202612	af63640dea679e30e3421cf46598d5bc18b0e2eebb5a326716c02987c9ad4037

202612	1d231a745ff90ff72bb9ae2eed18ab5746d78aefebe814a780d306dbb7901d47
202612	b6ff392141f231c10455608c1b51730f90ae46c42936347dd2b4a4f301a08c45
202612	d801a9d18a2eb2d0e8c1af378884da6d8bc6ee4b1def4b082e881b7d9cfd6757
202612	effd95d01814e3c91b4f7b5221fb46bb7d2ceae13be9d38ff714830aa6d3e457
202612	d8e68904d9e9533affb4d948dd96ab85ede03f2dd2e5e4ce4a185aceb68b846a
202612	0ff7ecbe8732b71d34ca4634695077dd27ccdaae2bfc682f0e89610543ecb3df
202612	1d0f80a7059fd4a82815e6ea68ed108f0e54bbd5339a20f3a2eec88f44ab9b9b
202612	1db98b2a37c16ca7fdf841545099d2fda4ce869c2c3fbad7def3b9536e427978
202612	eba9479e96b713028b7af097b83f34a0edc4b7ffec75e7b7448bc2017815ebce
202612	a54a2a0a873b9ff9e837d8755e569be510c6985fae46233ede308ae3f6c2efa5
202612	26c11c6c1dff8d739d19e91b330691c0a7a482b0261a04d6b6fd37dffd3334a9
202612	5dba4a6f3d507a54d9a52265950514c2dad83ec0c9712018fb0102ffbad20c1d
202612	0e8aaac3b0dfaf23c3dfbc3f91f65aa325d5b6d514f0d680ecbad032e25bc309
202612	890309453ad51d4e9687b9c7c644ec51cd15fd1b6cc66723f7dd4e3827614c97
202612	f1b73491d579b2447f48680ca05c72f2fe398e5c9ef064520f6b906b07c08b71
202612	c2b6215d7c85c6460946981f21904a38d95b0cb2be73aba05991b643067ad949
202612	9247eeaa04be8483f795bee5c44a2239d21a67071292d3c9a04750343bc23f7c
202612	0660dac30b9ac6969d36c7868114def473eebbac7f79a1dd510ae3e41fc51c7b
202612	b48f76e33764031737668fd24c1067e4d85577b46b9f2c53bb66828707fcebba
202612	ef389eb379db286f82b66a6db40b552bfdc0039fcfc0cb350dc0c5497e735fae
202612	f7367858895ce27310187ce1c9ff3a587e63a01395aa55f49979580bdbfd3e51
202612	40d9ad419e00e206035661b079454f4b2f857c4a322d4a084fb269ce496624fc
202612	16bc7c405977b4e19b6ffba8450c8ce21d31c77834cd5622a48dcef2d48710a5
202612	c8835c1b63117ebcb4645be40761ef025a0cce2e106ba353da6a15840cba73f2
202612	1bda68070438bc818770a1c1b8a54ed06705434c66474afa7f5aadf762184b27
202612	cf5140bf901a9a7a5a44c05f3fd55cf025c0edfcac1dea962443b1507c7d5e81
202612	7a576a35966134534361dedfe4b5f1083240c49524bb5e82da3e43052d199e69
202612	450c562056438893029974f87c742538e234942ecaaad365f177c1952fe47422
202612	e869fe9fcaaf39ecd0502f7a8906131b8e3cafd17f5588a2490cdf339c04328f
202612	3f7997930eeda64ac6db91f2a34873a2248e345c2069c5ff0ccebc1bcca0d838
202612	ef8dcfd68330815fc6a89971ce060f5046d69ba27afac95a615b01b3f0109439
202612	2faa565d0d913ed75d0d2a248d4d754ba0aa836d0acd01acab1b8d083d7022b8
202612	a8ebfa78c86eb166a37eaff4a9729681788e6b3ecfcdac6f67dcc248aa7fa295
202612	3fe1a8d8d45dcaceceb07dc8977118f2dad352ed7d8a57d7fe02ac3454e3afcf
202612	f50d299333564d56af8762bd7620a314e297ec14ebed4f7cbf6e9c460fac05fd
202612	840cdc057463a0d8fef20ff7ff9d1fd07adef1dcf6be0a03e1e25958c5021405
202612	6dbb18c980a4f6b32dd07f8590128411537fff181820d19ef1cb68dd041eed02
202612	896f10acbcb2272f50c54cdc82296bf95712161826ce5184730c52c74ce31093
202612	c8d7bf696d6ea722a2592d2e132e1c766343d29ec1b7fcfc25a41138b8030130
202612	971475142fe3d22f8998a40e5ea12860d0d86173a6a219c019afa548bfe4b9ce
202612	9e58bcbc26bde71ae78f5458e9bbdea0316777250bb993383502833cf679559e
202612	f34cb751428e96c2ddb786731d99054fab4e7a3bf042a82c801d6cfda08f8263
202612	751f16bbf590130eba70263fa4c1bd431aed75d5b0be9ff87c26ce0818fa211d
202612	dda7471101268d4ac032f6546fa1fbf5931061bd7087efa69caca3f75df522f6
202612	34e1e672c38e881c1919d0035b8ee0620ca22e43586e18223e51f55a93938c60

202612	719dec682e740af7b0d8a05019b80efb7d40f2a083a81974d6494d8d7cf0b992
202612	fe998f4fdda5e360dd6ed8e8df498aeae53176372c07c019bc834c204a86e9a1
202612	a0c1a47ca99959f87ea403241d4aa6e145b94230cfdc7a91628bb1420d377c55
202612	522b7dd7038659b2fd5ecaa2ca592918f7e967e7444ff49c4fc5426a4b630115
202612	a17c3668365d91caf18a30d648ba3e54151209eda85675adec175a29635c6d87
202612	56e93a508a91e50f3b7a3cc2f90e690ddf1ad4a117e12fc2fb8d3b7b4373e4d4
202612	60970c41e8963aad0087527753471ee08662b84935a26ecfb114f7511ced61e0
202612	763831fbed0a7299ee0b9c6bf42909d18fd1ce820456be20957dced02b5ccabd
202612	bcbbcd2b62a00c15c3724535b4a3d97738a1017fff4d3e2b6609b37396dc9d29
202612	67c865626609a576f91a9536de01a78c5c7c8de21dca8703e6a8c82cb15a3a6b
202612	dd86ee1b94884ce95ef736548cc944f9d4a634fb165bf512d7609b624ad77bfc
202612	1353a5e71139012c0f9765bddce4735491b919640f049243c239dad9aab9edba
202612	295421119b253763fd5c31e312102acba946770125b49b1443cdf2d1f67bec8f
202612	373c2a876f2699206b5a59c0681d3c0a35f28d1821efbcf08661b5bec25e62d9
202612	7ab5008efd0ee0d869ace47413a08aeaa91015b8297f6deb24b1fbd60238216d
202612	5748615b7b56f3b9d368bcb861e22e92787053dc7e31f158020edd4012f2ba45
202612	ce7079c99e56c4e04a866989c7f3a2b47de9ec0e34eaefa44f1369bc1bd8703f
202612	65c6b43f715ae8bd1d108673bbca7642285555ba057093526b507c3ad9c4e30c
202612	3ba879515893f2fefd916ed8bbce571abff0ce53e7e25b82ef2a9628a69af4b3
202612	8c4f477e0dee388013243b8ba5599c00f60231e3f4f86c6921698f5a7a02de86
202612	ec441beb68c0a97c8dc005c1135f884039c1110ba7be696a435aaad1c12d2224
202612	9219b6768fe878f24460832c6e4bcef1feefbb88c4435a57bc06b8fda5e321be
202612	46e953d011e3644e10bff0ec461c6816e8df0af56f1425d4cf6765698f0c5133
202612	528877cb0cc9401e0a561a0c3d1fb65cd40647dacc2bde15fee821414cd02812
202612	81b91de9ed7c313518334217ebe6d8f7b0dbc696f1e3b7a38d768a12cffb17b2
202612	3f1561426710a7cb3165359b842745ecf7c0b1691330e58f9513008ddb87dc81
202612	d4d21b6b1be0bde83ac16b3aee2dfb222a8c6e347f3f035a1d6e9535243ea7fc
202612	90dea5acef11bf404bd55f0f90e674c242a27b80a6d524313c180bb017365e89
202612	f6e5d455d6f3dcd055310b63964824c5c7170d313ef8e42bb5cad63b8e393a8c
202612	9542bf4f3e1d9728b1150d0a528ba54acf9e35d83a9a7579ab4e80494e5273a0
202612	9927b37f79cd9d9c71c8377519bc7a3e89564ec3c12f2afdc8a2af6ef221b383
202612	5c5c13fdd1be84e25bc3284728a948fdc0c607244b977a147565831095af4abf
202612	3661c4416f21328100ef4c1cb2702d76079d364d4b71129c18c2c413b44bb532
202612	0a30b78542bf3f91043983d6e8a13f4c44de48eb3e3d9e4d32f5a8aa8a3bdae1
202612	ecacb63e63097590b476e9f290b0a7c96ebbe0309ce919c138024e5b613e5b20
202612	660f07b185fb634d5a67ee084edf58c9417c1051af7a9b37abebe06034db6c30
202612	0693e27da04f95eab3c105fc5f9076243d43aec3bedcd83150a5105430d1a8d5
202612	0d06228c9cd7fc0d51b685f3ac880220fbcb6f20763b206beed5c570be319363
202612	13516d0d901e6098a302599c60bbaca7d494c9deb90f1c102a3c8ef0787e6630
202612	74d740655a45c14d535564541f611741dd24ba9186d780dd5488728f382b2423
202612	c83f0868bc451f1dbc2237ca216fd7621bd83dc60fac80fc485931654f03b154
685498	c34a3b726bed1c13cd66accb184d4a0f3f51238b2d2c1733dd99e9f5d74ae7d1
761389	01b257e003e8686d3b153331480a558f19eb323379e7eb6484efb9c089b4b9e8
770439	ee55915c178b9961c89d81f0059a9a3b54026e2be069d0d2e6dae12119bc0f69
775584	3a64c998b7abd2144d2bfde04fdeb631d0034db04ea30bf5b4400425d02d8742

782657	c83a2107e32b366704b9b747de2dfb1e008bf5cfcb0668f60132d5e2f8843848
844742	2c81d050f135e387d5d5f72fac9da88960926dc8af5b5fd7be92a48c57b27d6a
894416	4d282799872b46712f6cc6387d8286a76174cc3f195b645408a5dd7a0f9530d4
914263	5c58a204bf8b559118cefc4d65ede74fc45b009ae255fad29cd92aa65670e059
923755	8e5dac538c0efb357828a6a739605e8da4499a734a3ade177ba306d11f2604d6
950311	b1681099ffb5157d6f7057daf9f6f6090463d6e33716430a6141a0a0456de851
954707	a1e7aac4f34901c9394048eec9621e805b0d16b9d817255fe5a11863ac99d89d
954851	c1912d0d3dbdf7132a36f0abb4883b7873f9404a880efd24a374179b8a216981
960504	2ac32c2a3cdecec9d9e46ef3a606f334a607b3e41847f4805df7d6f2f2ef0fc1
960583	712037dc6cca4bef5eabfc4ab53a0316cd5066155030146a3d0f26f43889fa83
960591	36600922fa908d1cc32d5912e89a676932b93a944ebb086e0e785eba04d4e9aa
964973	2ba329cf242ce689ee0969a05fe859c0ed3365686b0f7619a3bda20edd3bf3b9
979604	c6e7cfc1b2d8d95bfc6a88b941071bd7f2cd02be51d039951410e194ec7dc796
982072	30bbc9f4bf951f011f823d048a91d29be544f636aa00a6b78d72a660e6efc0cd
990283	695d3fa1b7411f877f8ac8574a160c5c24da1f0f4505fc39e936c798d020098b
995534	32d8d17f8ecc6c8f7556039793c50f7def0f5eb9573321f31d482d6ca00df13d
1006680	785d320a94a1a515300a820e73b5ca713358aa2a3f45ccd3a6aed584669cb864
1006680	30f97c54470b57dfe8aa94a4097d645fbb864cb50ea756b436bb5c0bc69c8f45
1164055	f6b1637c4d4c95db5f16a872dc678849346ca7324aef2711406fff25ad213108

5.1 Source Code for update_adb.py

```
'''
Script to update ArangoDB graph to schema V8.
1st round of de-anonymization- get % of 0-mixin inputs; mark true
inputs.
To be run from directory ~/data
'''
import subprocess
import os
import json
import csv
from arango import ArangoClient

with open("/home/VMadmin/data/progress.txt", "r") as text_file:
    start = int(text_file.readline())+1
text_file.close()


#Connect to db and load graph
client = ArangoClient(hosts='http://localhost:8529')
db = client.db('monero', username='root', password=[redacted])
monero = db.graph('monero')

for i in range(start, 1980622):
    try:
        monero.insert_vertex('blocks', {'_key': str(i),
'blk_height':i})
    except:
        pass

    with open("/home/VMadmin/data/blocks/{}.json".format(i)) as
json_file:
        dict = json.load(json_file)
    json_file.close()

    data = dict["data"]
    txs = data['txs']

    for tx in txs:
        hash = str(tx["tx_hash"])
        if tx["coinbase"] == True:
            try:
                monero.link('has_cb_tx', 'blocks/{}'.format(str(i)),
'cb_txs/{}'.format(hash))
            except:
                pass
        else:
            try:
                monero.link('has_tx', 'blocks/{}'.format(str(i)),
'txs/{}'.format(hash))
            except:
                pass

            ec = monero.edge_collection("input_of")
            ec1 = monero.edge_collection("ring_member_of")
            vc = monero.vertex_collection("ring_members")
            vc1 = monero.vertex_collection("inputs")
            vc2 = monero.vertex_collection("outputs")

            with open("/home/VMadmin/data/inputs_count.txt", "r") as
text_file:
                inputs_count = int(text_file.readline())
            text_file.close()

            with open("/home/VMadmin/data/deanon_inputs.txt", "r")
as text_file:
                deanon_inputs = int(text_file.readline())
            text_file.close()

            inputs = ec.edges("txs/{}".format(hash),"in")['edges']
            inputs = [str(input['_from']) for input in inputs]

            for input in inputs:

                inputs_count+=1

                ring_members = ec1.edges(input, "in")['edges']
                ring_members = [str(ring_member['_from']) for
ring_member in ring_members]
```

```
                mixin_count = len(ring_members)

                if mixin_count == 1:

                    deanon_inputs+=1
                    rm = vc.get(ring_members[0])
                    rm['true_input'] = True
                    vc.update(rm)

                    ip = vc1.get(input)
                    ip['deanon'] = True
                    vc1.update(ip)

                    pk = rm['pub_key']
                    op = vc2.get(pk)
                    op['spent'] = True
                    vc2.update(op)

            with open("/home/VMadmin/data/inputs_count.txt", "w") as
text_file:
                text_file.write("{}".format(inputs_count))
            text_file.close()

            with open("/home/VMadmin/data/deanon_inputs.txt", "w")
as text_file:
                text_file.write("{}".format(deanon_inputs))
            text_file.close()

    with open("/home/VMadmin/data/progress.txt", "w") as text_file:
        text_file.write("{}".format(i))
    text_file.close()
```

5.2 Source Code for update_adb_cleanup.py

```
'''
Script to update ArangoDB graph to schema V8.
1st round of de-anonymization- get % of 0-mixin inputs; mark true
inputs.
To be run from directory ~/data
'''
import subprocess
import os
import json
import csv
from arango import ArangoClient

with open("/home/VMadmin/data/progress.txt", "r") as text_file:
    start = int(text_file.readline())+1
text_file.close()


#Connect to db and load graph
```

```
client = ArangoClient(hosts='http://localhost:8529')
db = client.db('monero', username='root', password=[redacted])
monero = db.graph('monero')

for i in range(start, start+1):
    try:
        monero.insert_vertex('blocks', {'_key': str(i),
'blk_height':i})
    except:
        pass

    with open("/home/VMadmin/data/blocks/{}.json".format(i)) as
json_file:
        dict = json.load(json_file)
    json_file.close()

    data = dict["data"]
    txs = data['txs']

    for tx in txs:
        hash = str(tx["tx_hash"])
        if tx["coinbase"] == True:
            try:
                monero.link('has_cb_tx', 'blocks/{}'.format(str(i)),
'cb_txs/{}'.format(hash))
            except:
                pass
        else:
            try:
                monero.link('has_tx', 'blocks/{}'.format(str(i)),
'txs/{}'.format(hash))
            except:
                pass

            ec = monero.edge_collection("input_of")
            ec1 = monero.edge_collection("ring_member_of")
            vc = monero.vertex_collection("ring_members")
            vc1 = monero.vertex_collection("inputs")
            vc2 = monero.vertex_collection("outputs")

            with open("/home/VMadmin/data/inputs_count.txt", "r") as
text_file:
                inputs_count = int(text_file.readline())
            text_file.close()

            with open("/home/VMadmin/data/deanon_inputs.txt", "r")
as text_file:
                deanon_inputs = int(text_file.readline())
            text_file.close()

            inputs = ec.edges("txs/{}".format(hash),"in")['edges']
            inputs = [str(input['_from']) for input in inputs]

            for input in inputs:
                inputs_count+=1

                ring_members = ec1.edges(input, "in")['edges']
                ring_members = [str(ring_member['_from']) for
ring_member in ring_members]

                mixin_count = len(ring_members)

                if mixin_count == 1:

                    deanon_inputs+=1
                    rm = vc.get(ring_members[0])
                    rm['true_input'] = True
                    vc.update(rm)

                    ip = vc1.get(input)
                    ip['deanon'] = True
                    vc1.update(ip)

                    pk = rm['pub_key']
                    op = vc2.get(pk)
                    op['spent'] = True
                    vc2.update(op)

            with open("/home/VMadmin/data/inputs_count.txt", "w") as
text_file:
                text_file.write("{}".format(inputs_count))
            text_file.close()

            with open("/home/VMadmin/data/deanon_inputs.txt", "w")
as text_file:
                text_file.write("{}".format(deanon_inputs))
            text_file.close()


    with open("/home/VMadmin/data/progress.txt", "w") as text_file:
        text_file.write("{}".format(i))
    text_file.close()
```

5.3 Source Code for check_status_restart.py

```
import requests
import os
import subprocess
import locale
import datetime

cmd1 = ['pgrep', '-f', 'anon_analysis.py']

#Function to check if process is running
def check_proc(cmd):
    proc = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    o, e = proc.communicate()
    return o.decode('ascii')!=''

if not check_proc(cmd1):
    with open('/home/VMadmin/data/progress.txt') as file:
        n = int(file.readline())
    file.close()
    if n!=1980622:
        os.system('python
/home/VMadmin/data/anon_analysis_cleanup.py')
        os.system('python /home/VMadmin/data/anon_analysis.py &>
anon_analysis.out &')

        with open("/home/VMadmin/data/logs/restart_log2", 'a') as
file:
            file.write("Restart at
{}\n".format(datetime.datetime.now()))
        file.close()
    else:
        pass
```

5.4 Source Code for bot.py

```
import requests
import os
import subprocess
import locale
import datetime

locale.setlocale( locale.LC_ALL, '' )

def telegram_bot_sendtext(message):

    bot_token = [redacted]
    chatID = [redacted]
    send_text = 'https://api.telegram.org/bot' + bot_token +
'/sendMessage?chat_id=' + chatID + '&text=' + message
    requests.get(send_text)

    return

cmd1 = ['pgrep', '-f', 'anon_analysis.py']
cmd2 = ['df', '-h']
cmd3 = ['df', '-i']

#Function to check if process is running
def check_proc(cmd):
    proc = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    o, e = proc.communicate()
    return o.decode('ascii')!=''
```

```
def check_mem(cmd):
    proc = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE)
    o, e = proc.communicate()
    return o.decode().split()[23]

def get_status():

    with open("data/progress.txt", "r") as text_file:
        count = int(text_file.readline())
    text_file.close()
    count_str = locale.format("%d", count, grouping=True)
    dumped_percent = float(count)/2112472*100

    script_status = ('Running' if check_proc(cmd1) else 'Down')
    mem_usage = check_mem(cmd2)
    inode_usage = check_mem(cmd3)

    with open("data/inputs_count.txt", "r") as text_file:
        inputs_count = int(text_file.readline())
    text_file.close()

    with open("data/deanon_inputs.txt", "r") as text_file:
        deanon_inputs = int(text_file.readline())
    text_file.close()

    deanon_ratio = float(deanon_inputs)/float(inputs_count)

    if script_status == 'Down':
        with open("data/anon_analysis.out",'r') as file:
            try:
              error_msg = file.readlines()[-1]
            except: error_msg = "No error msg"
        file.close()
        timestamp =
datetime.datetime.fromtimestamp(os.path.getmtime("data/anon_analysis
.out"))
    else:
        error_msg = 'NIL'
        timestamp = 'NIL'

    return "================================\nAnonymity analysis
untill block {} ({}%).\n\nAnonymity analysis script status:
{}.\n\nDeanon ratio: {}\n\nError message: {}\n\nTimestamp:
{}\n\nMemory usage:
{}\n================================".format(count_str,
dumped_percent, script_status, deanon_ratio, error_msg, timestamp,
mem_usage)

telegram_bot_sendtext(get_status())
```

6.1 Source Code for anon_analysis.py

```
'''
Subsequent rounds of de-anonymization
'''
import subprocess
import os
import json
import csv
from arango import ArangoClient

with open("/home/VMadmin/data/progress.txt", "r") as text_file:
    start = int(text_file.readline())+1
text_file.close()


#Connect to db and load graph
client = ArangoClient(hosts='http://localhost:8529')
db = client.db('monero', username='root', password=[redacted])
monero = db.graph('monero')

for i in range(start, 1980622):

    with open("/home/VMadmin/data/blocks/{}.json".format(i)) as
json_file:
        dict = json.load(json_file)
    json_file.close()

    data = dict["data"]
    txs = data['txs']

    for tx in txs:
        hash = str(tx["tx_hash"])
        if tx["coinbase"] == True:
                pass
        else:

            ec = monero.edge_collection("input_of")
            ec1 = monero.edge_collection("ring_member_of")

            vc = monero.vertex_collection("ring_members")
            vc1 = monero.vertex_collection("inputs")
            vc2 = monero.vertex_collection("outputs")

            with open("/home/VMadmin/data/inputs_count.txt", "r") as
text_file:
                inputs_count = int(text_file.readline())
            text_file.close()

            with open("/home/VMadmin/data/deanon_inputs.txt", "r")
as text_file:
                deanon_inputs = int(text_file.readline())
            text_file.close()
            inputs = ec.edges("txs/{}".format(hash),"in")['edges']
            inputs = [str(input['_from']) for input in inputs]

            for input in inputs:

                input_json = vc1.get(input)
                inputs_count+=1

                if 'deanon' in input_json.keys() and 
input_json['deanon']==True:
                    deanon_inputs+=1
                else:

                    ring_members = ec1.edges(input, "in")['edges']
                    ring_members = [str(ring_member['_from']) for 
ring_member in ring_members]
                    unspent = []

                    for ring_member in ring_members:
                        pk_str = ring_member.split(',')[1]
                        pk_json = vc2.get(pk_str)
                        if not 'spent' in pk_json.keys() or 
pk_json['spent'] ==False:
                            unspent.append(ring_member)

                    unspent_count = len(unspent)

                    if unspent_count == 1:

                        deanon_inputs+=1
                        rm = vc.get(ring_members[0])
                        rm['true_input'] = True
                        vc.update(rm)

                        ip = vc1.get(input)
                        ip['deanon'] = True
                        vc1.update(ip)

                        pk = rm['pub_key']
                        op = vc2.get(pk)
                        op['spent'] = True
                        vc2.update(op)

            with open("/home/VMadmin/data/inputs_count.txt", "w") as 
text_file:
                text_file.write("{}".format(inputs_count))
            text_file.close()

            with open("/home/VMadmin/data/deanon_inputs.txt", "w") 
as text_file:
                text_file.write("{}".format(deanon_inputs))
            text_file.close()

    if i%1000==0 or i==1980622:
        with open("/home/VMadmin/data/inputs_count.txt", "r") as
text_file:
            inputs_count = int(text_file.readline())
        text_file.close()

        with open("/home/VMadmin/data/deanon_inputs.txt", "r") as
text_file:
            deanon_inputs = int(text_file.readline())
        text_file.close()

        deanon_ratio = float(deanon_inputs)/float(inputs_count)

        with open('/home/VMadmin/data/round2.csv', 'a') as file:
            csv_writer = csv.writer(file, delimiter=',',
quotechar='"', quoting=csv.QUOTE_MINIMAL)
            csv_writer.writerow([i, deanon_ratio])
        file.close()

    with open("/home/VMadmin/data/progress.txt", "w") as text_file:
        text_file.write("{}".format(i))
    text_file.close()
```

6.2 Source Code for anon_analysis_cleanup.py

```
'''
Subsequent rounds of de-anonymization
'''
import subprocess
import os
import json
import csv
from arango import ArangoClient

with open("/home/VMadmin/data/progress.txt", "r") as text_file:
    start = int(text_file.readline())+1
text_file.close()


#Connect to db and load graph
client = ArangoClient(hosts='http://localhost:8529')
db = client.db('monero', username='root', password=[redacted])
monero = db.graph('monero')

for i in range(start, start+1):

    with open("/home/VMadmin/data/blocks/{}.json".format(i)) as
json_file:
        dict = json.load(json_file)
    json_file.close()

    data = dict["data"]
    txs = data['txs']

    for tx in txs:
        hash = str(tx["tx_hash"])
        if tx["coinbase"] == True:
                pass
        else:

            ec = monero.edge_collection("input_of")
            ec1 = monero.edge_collection("ring_member_of")

            vc = monero.vertex_collection("ring_members")
            vc1 = monero.vertex_collection("inputs")
            vc2 = monero.vertex_collection("outputs")

            with open("/home/VMadmin/data/inputs_count.txt", "r") as
text_file:
                inputs_count = int(text_file.readline())
            text_file.close()

            with open("/home/VMadmin/data/deanon_inputs.txt", "r")
as text_file:
                deanon_inputs = int(text_file.readline())
            text_file.close()

            inputs = ec.edges("txs/{}".format(hash),"in")['edges']
            inputs = [str(input['_from']) for input in inputs]

            for input in inputs:

                input_json = vc1.get(input)
                inputs_count+=1

                if 'deanon' in input_json.keys() and
input_json['deanon']==True:
                    deanon_inputs+=1
                else:

                    ring_members = ec1.edges(input, "in")['edges']
                    ring_members = [str(ring_member['_from']) for
ring_member in ring_members]
                    unspent = []

                    for ring_member in ring_members:
                        pk_str = ring_member.split(',')[1]
                        pk_json = vc2.get(pk_str)
                        if not 'spent' in pk_json.keys() or
pk_json['spent'] ==False:
                            unspent.append(ring_member)

                    unspent_count = len(unspent)

                    if unspent_count == 1:

                        deanon_inputs+=1
                        rm = vc.get(ring_members[0])
                        rm['true_input'] = True
                        vc.update(rm)

                        ip = vc1.get(input)
                        ip['deanon'] = True
                        vc1.update(ip)

                        pk = rm['pub_key']
                        op = vc2.get(pk)
                        op['spent'] = True
                        vc2.update(op)

            with open("/home/VMadmin/data/inputs_count.txt", "w") as
text_file:
                text_file.write("{}".format(inputs_count))
            text_file.close()

            with open("/home/VMadmin/data/deanon_inputs.txt", "w")
as text_file:
                text_file.write("{}".format(deanon_inputs))
            text_file.close()

    if i%1000==0 or i==19806220:
        with open("/home/VMadmin/data/inputs_count.txt", "r") as
text_file:
            inputs_count = int(text_file.readline())
        text_file.close()

        with open("/home/VMadmin/data/deanon_inputs.txt", "r") as
text_file:
            deanon_inputs = int(text_file.readline())
        text_file.close()

        deanon_ratio = float(deanon_inputs)/float(inputs_count)

        with open('/home/VMadmin/data/round2.csv', 'a') as file:
            csv_writer = csv.writer(file, delimiter=',',
quotechar='"', quoting=csv.QUOTE_MINIMAL)
            csv_writer.writerow([i, deanon_ratio])
        file.close()

    with open("/home/VMadmin/data/progress.txt", "w") as text_file:
        text_file.write("{}".format(i))
    text_file.close()
```