# A study to build a tool for retrieving information from Bitcoin transactions

Niklas Larsson Hampus Runesson
*Email: nikla691@student.liu.se, hamru119@student.liu.se*
Supervisor: Niklas Carlsson, niklas.carlsson@liu.se
Project Report for Information Security Course
*Linköpings Universitetet, Sweden*

## Abstract

Because of its pseudonymous and decentralized system, Bitcoin is commonly used by scammers to lure people to send them money. Bitcoin can be sent between users without revealing their identity, which means that they stay anonymous until they post their identity tied to their Bitcoin address.

In this project, we have created a tool that given a file of Bitcoin addresses, can fetch information about the addresses and their corresponding transactions. The tool also includes properties to filter the data in order to make it more manageable in future work. Using the Blockchain API to fetch data, our tool can retrieve public information about an address and create a data set that can be used to analyze transactions regarding an address. Besides filtering, the tool is also capable of converting the filtered data from JSON to CSV format in order to make it easier to use in other tools.

## I. Introduction

### A. Motivation

Bitcoin is a pseudonymous decentralized currency where no central authority is used and funds are tied to Bitcoin addresses instead of a physical person. Bitcoins can be sent directly from user to user but in practice intermediaries are often used to simplify the process. Bitcoin utilizes a ledger, also called block-chain, to track all transactions using Bitcoins. This ledger is available to anyone.

Bitcoin transactions was commonly used by criminals because of the anonymity, but since authorities found out that a lot of information about the transactions could be found in the ledger, they have started to use it for information. Since the ledger is available to everyone and contains every transaction a huge map of all transactions could be made to correlate a Bitcoin address to a wallet or person [1]. Criminals did however adapt to this "problem" by using different strategies to ensure anonymity of their activity when using Bitcoins [2].

In this project we intend developed a tool, that given a Bitcoin address, can fetch more information about the address and each corresponding transaction from the ledger. Furthermore, we intend to build a data set of the fetched data that is easy to manage and analyze. We want to point out that we are not going create some kind of map over the data and try to find names that correspond to an address. We are only going to fetch the information and create a data set.

### B. Aim

The aim of the study is to develop a tool that will help gather useful information about a given Bitcoin address. The tool will also include some filter properties in order to get a cleaner data set that also is easier to manage and analyze in future work.

### C. Research questions

To be able to fulfill our goals with the project we choose the following research questions:

- How to collect data from Bitcoin addresses?
- What raw data can be retrieved from an address?
- How can we filter and process the raw data in order to make it more manageable?

## II. Theory

This section will display background information about APIs and tools used in the project.

### A. Bitcoin

Bitcoin is a decentralized digital currency that were created in 2009 by a person or group in 2008 that goes under the pseudo name Satoshi Nakamoto. The main purpose of the currency is to make peer-to-peer transaction viable without any involvement of a third party, such as a bank. The lack of a central administration unit makes it impossible to manipulate the value of Bitcoin and e.g induce inflation of the currency. Instead, Bitcoin is created as a reward for mining. Every transaction that is made is stored in the distributed ledger, which is called a blockchain. One property of Bitcoin is that it is pseudo anonymous, which means that you are anonymous as long as you do not connect your name to your Bitcoin addresses [3].

A non empty Bitcoin wallet contains unspent transaction outputs (UTXOs). UTXOs are immutable, meaning that you cannot split it to cover a specific value. Instead UTXOs are collected from the wallet to, at a minimum, cover the cost of the transaction. This is called the input of a transaction. All input UTXOs are spent and used to create new ones for the output of the transaction. If there is change left in the transaction, i.e. the input had a higher value than what the

recipient(s) are sent, a change output is also created. This output transaction is then sent back to the sender. [4]

The following tables describes the structure of the different parts of a transaction. Note that some information has been excluded because it is not inside the scope of this study.

| Header | Data type | Description |
|---|---|---|
| ver | int32_t | Data format version for transaction |
| tx_in_count | var_int | Number of transaction inputs |
| tx_out_count | var_int | Number of transaction outputs |
| time | uint32_t | Time when transaction was made |
| tx_in | tx_in[] | Array containing transaction inputs |
| tx_out | tx_out[] | Array containing transaction outputs |

TABLE I: Transaction (tx) headers

| Header | Data type | Description |
|---|---|---|
| signature script | uchar[] | Script for transaction authorization |
| previous_output | outpoint | Previous output transaction reference |

TABLE II: Transaction input (tx_in) headers

| Header | Data type | Description |
|---|---|---|
| hash | char[32] | Hash reference to the transaction |
| index | uint32_t | Index of specific output transaction |

TABLE III: Outpoint (previous_output) headers

| Header | Data type | Description |
|---|---|---|
| value | int64_t | Transaction value (in satoshis) |
| pk_script | uchar[] | Script for claiming output |

TABLE IV: Transaction output (tx_out) headers

It is recommended to generate a new public/private key pair for each new transaction. To keep track of funds a user would need to backup each new key-pair which could become very cumbersome. To combat this HD-wallets (hierarchical deterministic wallet) were introduced in BIP-0032. HD-wallets constructs a tree of public/private key starting from a root node. The root node is a 12 word seed chosen by the user and to this root node a counter is added. After each transaction the counter is incremented and a new public/private key pair is generated. This means that the user only need to know the seed to be able to retrieve all of their funds. It also simplifies the process of generating new key pair for each transaction which in turn makes it really easy to maintain the policy of always using new key pairs for new transactions. [4]

The general structure of a HD-wallet key pair generation is shown in the following diagram

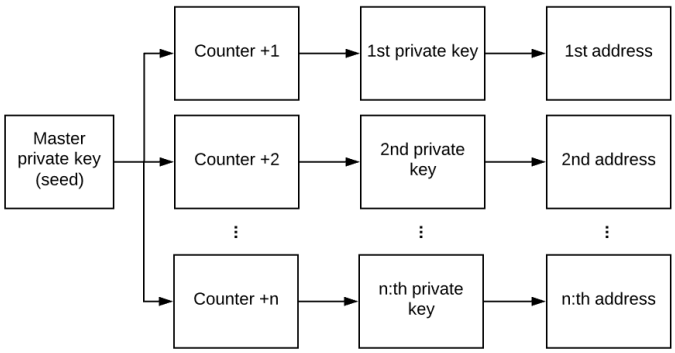


Fig. 1: General structure of HD-wallet key pair generation

### B. JSON

JSON (JavaScript Object Notation) is a popular text based data format used in most web applications [5]. A JSON object contains name/value pairs where each pair is separated by a comma. A value can be multiple data types:

- Bool
- String
- Number
- Object
- Array
- Null

This gives a JSON object nestling capabillities. JSON is a raw data format which means that a parser is needed to handle the data. The structure of JSON has an impact on parsing speeds, which is around 100MB/s per core. Some newer tools can reach speeds up to 2GB/s per core. [6]

### C. CSV

A CSV (comma separated value) file is a text file that separates values using a delimiter, often a comma. Each row or line in the file is a record of data that consists of fields, separated by the delimiter. The CSV file type is often used to exchange and move data between sources but also sees use in areas such as data analysis. Because of the simplistic approach of CSV files it does not offer nested records or fields (like JSON) but this does instead increase parsing performance. [7] Below an example of the CSV structure is shown.

```
City,Longitude,Latitude
Linköping,58°23'53.3,15°34'38.1
```

## III. Method

In this section we will describe how we built our tool and how we used it to fetch data about a given Bitcoin address.

To be able to build a tool that could fetch information about a Bitcoin address we first researched how Bitcoin transactions works. We then had to do some research of different APIs that we could use to fetch information. After considering different alternatives we decided to use the Blockchain Data API. We also decided that we would write our tool in Python since it is a simple language with a lot of use full libraries that we could use if needed. Also, both of us felt comfortable with writing Python code.

### A. APIs

| API | Blockchain | Blockcypher | Sochain |
|---|---|---|---|
| Address | × | × | × |
| Total received | × | × | × |
| Total sent | × | × | × |
| Number of txs | × | × | × |
| Txs hash | × | × | × |
| Txs result | × | × | × |
| Txs input | × | × | × |
| Txs output | × | × | × |
| Txs time | × | × | × |

TABLE V: Information comparison between three different apis.

We looked into 3 different APIs for fetching information, Blockchain Data API [8], Blockciphers Blockchain API [9] and Sochains API [10]. We looked into what information they could give and picked the most relevant information. As seen in Figure V, they all could supply us with the same information that we wanted to look at in JSON format, which were easy for us to handle. While both the Blockchain API and Blockcypher API could give us all of the listed information in Figure V in one API call, the Sochain API needed more calls. We also wanted to fetch as many transactions as possible and the Blockchain API could give us a maximum of 100 transaction at a time while the limit of the other two APIs were 50. This meant that in order to get all transactions corresponding to an Bitcoin address, the Blockcypher and Sochain API needed a higher number of API calls in order to give the same information as the Blockchain API.

A disadvantage of all three of these is that the number of API calls are limited. The websites do not specify exactly how the limitations looks like which makes it hard do know which one is to be preferred in this case.

We found all three of them easy to use and that they could be implemented in python which we perfered to write our code in. The implementation for the Blockchain and Sochain APIs were to perform simple HTTP GET requests while the Blockcyphers API had a library that were imported into the project. This means that Blockcypher is ready for use right away, while you have to import some other request library to use the other two. For us, this was not a problem since we have worked with the Requests library before, which is really easy to implement and use [11].

Since the Blockchain API needed the fewest API calls, which needed to be considerd because of the rate limits, we choose to use that API.

### B. Building the tool

Figure 2 shows the structure of our tool. It's built with two main components in mind. Collect raw data and filter the data. As the diagram shows we will continue to make API calls until we run out of transactions or addresses to fetch. After that the data is filtered to make it easier to handle, i.e. converted to a smaller JSON object or a CSV file.

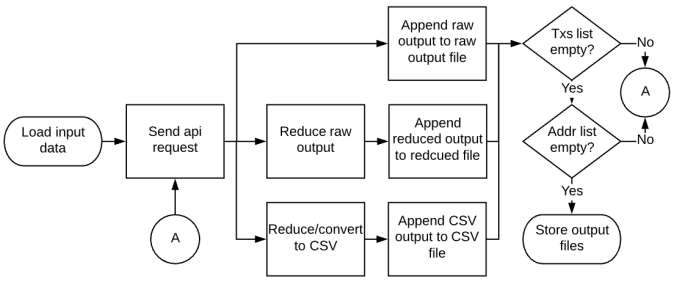


Fig. 2: General structure of our script

*1) Making API calls:* Since the Blockchain Data API used simple HTTPS GET requests to fetch data we needed a library to make requests. For this we used the library requests which is a simple HTTP library for python [11]. To use it, we installed it using pip3:

```
$ pip3 install requests
```

and imported it into our program. To make an api call to Blockchains service we used the following URL:

```
blockchain.info/multiaddr?active=$address&n=n&offset=m
```

where $address corresponds to the given Bitcoin address, $n corresponds to the number of transactions requested and &offest corresponds to the starting point for fetching transactions. We choose n=100 to maximize the amount of transactions received per api call. If there were more than 100 transactions available for an address, the offset was adjusted to ensure that we could capture all of the transactions. E.g offset=0 for transactions 1-100, offset=100 for transactions 101-200 and so on. In the code, we sat the parameters as an JSON object, and made the request to the URL with those parameters:

```
params = {
”active”: address,
”n”: n,
”offset”: offset
}
response = requests.get(url=URL, params=params)
```

When we received the response, we only needed to check the status code of it. If it was 200, which meant that the request was successful, we extracted the JSON object and started to filter the incoming data.

*2) Filtering data:* There are two parts to our filtering process. The first step is to remove unnecessary values from the retrieved raw data. This is done to save on space and make processing/reading the data easier. The raw JSON object will be reduced to the headers shown in figure V. The second filter is a to convert the data to a CSV file. The reason we added a CSV conversion step is to enable easier data processing in the future. For this study we have implemented a simple conversation tool that converts the JSON file into two separate CSV files, one containing the information about the

addresses (address.csv) and one containing information about the corresponding transactions (transaction.csv), as represented in below:

address.csv

```
address,n_tx,total_received,total_sent
example_address,100,1000000,1000000
```

transaction.csv

```
address,hash,fee,result,time
example_address,example_hash,100,1000,yy-mm-dd-
hh:mm:ss
```

Since the JSON object that we extracted had nestled JSON object we needed to split it into two separate files when we converted the data to CSV.

*3) Benchmarking the tool:* To evaluate the performance of our tool we implemented two different tests. The first test calculates the average response time from the API and the second test calculates the average processing time (converting to JSON and CSV) for 100 transactions

To calculate the average response time we made 100 API calls requesting 100 transactions (maximum amount per request). We used a random offset for each request to ensure a better spread of data size per request. For each request we recorded the time it took the get a response. After all requests had been made we used the following function to calculate the average response time:

$$\sum_{n=1}^{n\_req} \frac{req\_time_n}{n\_req}$$

where $n\_req$ is the number of requests made and $req\_time_n$ is the response time for the n:th request

After every request, we process the data that we collected which usually contains 100 transactions. We are measuring the time to process these 100 transactions, i.e filter the JSON data, by calculating the time it takes to process every request and then sum these times together and dividing by the number of requests to get the average amount of time. The time to convert the JSON file to CSV is also measured by calculating the average time it takes to convert 1000 transactions from JSON to CSV.

## IV. Results

This section will show and explain the output given by our script. We will show four different results: Raw output, Reduced JSON output, Reduced CSV output and our small analytical results. The results shown will mainly focus on the size of the different data sets and information gathered.

### A. Raw output

The raw output gathered by our script is the biggest data set containing every header given by the API. The size varies a lot because the number of input and output transactions can vary for each single transaction. The table below shows an example of the variation in lines written for multiple addresses with the same number of transactions requested.

| addr# | n_tx | lines written |
|---|---|---|
| 1 | 50 | 7528 |
| 2 | 50 | 7908 |
| 3 | 50 | 9232 |
| 4 | 50 | 8712 |
| 5 | 50 | 8064 |
| 6 | 50 | 7854 |
| 7 | 50 | 8194 |
| 8 | 50 | 7298 |

TABLE VI: Number of rows in raw JSON file

### B. Reduced output

The reduced output is filtered and only contains the fields described in Table V. Therefore a significant decrease in lines written is seen in the table bellow:

| addr# | n_tx | lines written |
|---|---|---|
| 1 | 50 | 2971 |
| 2 | 50 | 3104 |
| 3 | 50 | 3576 |
| 4 | 50 | 3298 |
| 5 | 50 | 3186 |
| 6 | 50 | 3156 |
| 7 | 50 | 3194 |
| 8 | 50 | 2930 |

TABLE VII: Number of rows in filtered JSON file

When comparing the raw output to the reduced JSON output the number of lines written is approximately reduced by 60% while also maintaining the same amount of usable information. In Listing 1 an address with one transaction is visualized in order to show a snippet of our result (alot of transactions have been removed which is why balance etc. does not add up).

```
{
    "addresses": {
            "address": "16hnZniy6xRZAKrp8
               gVzaxDqR5DMJEfCc2",
            "final_balance": 100000000,
            "n_tx": 419,
            "total_received": 92721986704
               ,
            "total_sent": 92621986704
    }
    "txs": {
        "hash": "52bfdcc8adc6b36815096461
           fcca19577ffc573e53686a5555226
           de3aeb8ad56",
        "fee": 4176,
        "result": 100000000,
        "time": 1588590750,
        "inputs": {
            "prev_out": {
                "value": 11533000,
                "addr": "15p4
                   iUfhnsszLNAbx4Hnayz5
```

```
                    zZQZVQ8vNF"
            }
        }
        "out": {
            "value": 100000000,
            "addr": "16hnZniy6xRZAKrp8
                gVzaxDqR5DMJEfCc2"
        }
    }
}
```

Listing 1: Reduced JSON data

### C. CSV output

As described in section III-B2 the CSV output consists of two different files. The *address.csv* file writes one line per address from the input file.

When converting the nestled JSON objects into CSV, the raw data was reduced even more. The table bellow shows how many lines are written to *transaction.csv* for every address:

| addr# | n_tx | lines written |
|---|---|---|
| 1 | 50 | 50 |
| 2 | 50 | 50 |
| 3 | 50 | 50 |
| 4 | 50 | 50 |
| 5 | 50 | 50 |
| 6 | 50 | 50 |
| 7 | 50 | 50 |
| 8 | 50 | 50 |

TABLE VIII: Number of rows in CSV transactions file

By comparing the CSV output and the other two JSON data sets, the number of lines are reduced significantly but some information was also lost. We no longer have access to detailed information about the input and output transactions that went into a transaction. I.e. we cant track who money was sent too or whom money was sent from.

### D. Benchmark results

In our first test we saw and average of 1.32 seconds per request. Some interesting results are that most requests take around 0.7 seconds while some outliers take up to 16 seconds.

To filter the JSON data from the raw data, it takes approximately 0.009 seconds/100 transactions. Furthermore, to convert the data JSON to CSV format takes approximately 0.0038 seconds/100 transactions.

## V. Discussion

In the discussion section we will review and evaluate our results. We will also discuss our method of work, improvements that could have been done and further development of the project.

### A. Results

From our results we can see that filtering the raw output has a significant affect on the file size of the data set generated. This in turn means that performance should increase greatly when doing analytical work on the data set. As the results show the CSV files are really small compared to both the raw and reduced JSON data. Even if some information is lost in this conversion we would argue that the best approach is to only convert to CSV files because they don't require and handling of nestled objects. Also the information that is lost could be added to its own CSV file. This would however mean that multiple conversions would be made on the raw data which could slow down the script. But an advantage would be that we would have smaller subsets of the complete data that is much easier to do analytical work on.

The benchmarking results show that the bottleneck of our tool is making the API calls, however this bottleneck is highly dependent on the servers that we are requesting data from and is in many ways not a problem we can solve.

### B. Method

Our method is based on fetching data and reducing the raw data as much as possible while also retaining information. That is why we have a reduction to a smaller JSON object since it allowed us to do exactly what we set out to do. We were however also instructed to look into converting the data into CSV since it could provide easier data analysis in the future and thus we added a CSV converter into our script. Based on the results we would now probably completely scrap the JSON reduction step and instead simply create multiple CSV files containing all of the necessary information. As mentioned previously this would most likely improve performance for analysis work in the future. It would also mean that all the data is available in only one format which we think is preferred. However, the nestled JSON objects makes it harder to convert to CSV right away. We did not manage to find a library which could handle nestled JSON objects and arrays of these objects. Therefore, in order to convert the raw data to CSV right away, would require to somehow looping through every object and extract information.

Our method and our tool are built to fetch data as fast as possible, but regarding the rate limit, that is discussed more in the next section, we need to do some changes. Since the data sets that we are going to use as input later on is very large and will require a lot of request, we need to slow down the tool. However, we do not know exactly what the rate limits are which requires us to do further testing in that area when running data sets with a large amount of addresses.

### C. Limitations

During our testing period we ran into a few problems that affected our ability to collect data. The API we used has request limits that are not stated in their documentation which meant that we quickly ran into problems when running our script with a bigger data set given to us. Luckily we had access to VPN services that allowed us to circumvent the

limit and avoid losing 24h of making requests. We are not sure what measurements are used to issue a request ban. To further investigate this we made some simple scripts to try and find the rate limit. We made one script that sent requests as fast as possible and another that implemented a sleep timer (500ms) between each request. The first script showed that they measure requests per second because we were blocked after only 80-100 requests. The other script was blocked after 3387 requests. We now know that there is a lower limit to how fast we can send requests and that there is a upp limit to how many requests we can send. Our small tests do however not show exactly how to optimize our API calls and there could also be many more restrictions that we have not encountered. This means that the tools must atleast be slowed down to work properly. However, by slowing down the tool we would increase the time to collect every transaction from every address drastically, especially when there are a large number of address to fetch information about. When researching this problem further, we found that you can request API keys, however these keys seemed to be aimed towards their wallet and transaction API that is used when building websites.

The other big problem we faced was that some transaction information is lost when there are input and output transactions with multiple addresses. This is because each input is collected into a single sum of Bitcoins that is then distributed to the outputs. So from our perspective we are only able to track the total amount sent or received but not the exact amount sent to a specific address. Note that this is only when there are multiple addresses in the input and output. For us this problem meant that our reduced CSV data set became much smaller than anticipated. We had hoped to have a data set tracking exact transaction history from addresses but sadly that is not possible.

### *D. Future work*

We think that further development of our tool is needed and that functionality like only converting to CSV is a good first step. We would also like to see analytical tools that can read and process our data sets. That could prove to be a valuable combination of tools to track the monetary activity of scammers. By collecting a ton of data and searching the internet for addresses tied to identities it could even be possible to create a tool that in some way could tie certain addresses to real identities.

## VI. Conclusion

We have now built a first iteration of a tool that creates data sets of Bitcoin transactions. All that is needed is an input file of addresses and the tool will collect every transaction ever made by those addresses. The tool filters out unnecessary data to greatly reduce computation overhead for further data analysis. The tool creates two different data sets, one in the JSON format and the other in a CSV format. The CSV data set is made up of two different files and does not contain as detailed information compared to the JSON data set but allows for better performance when processing the data set.

## References


[1] "Is bitcoin anonymous?." https://bitcoinmagazine.com/guides/bitcoin-anonymous. Accessed: 2020-04-01.

[2] "Mapping the bitcoin economy could reveal users' identities." https://www.technologyreview.com/s/518816/mapping-the-bitcoin-economy-could-reveal-users-identities/. Accessed: 2020-04-01.

[3] "Vad är bitcoin?." https://www.bitcoin.se/vad-ar-bitcoin. Accessed: 2020-04-16.

[4] "Transaction documentation." https://en.bitcoin.it/wiki/Transaction. Accessed: 2020-04-16.

[5] D. Xie, B. Chandramouli, Y. Li, and D. Kossmann, "Fishstore: Faster ingestion with subset hashing," in *Proceedings of the 2019 International Conference on Management of Data*, SIGMOD '19, (New York, NY, USA), p. 1711–1728, Association for Computing Machinery, 2019.

[6] Y. Li, N. Katsipoulakis, B. Chandramouli, J. Goldstein, and D. Kossman, "Mison: A fast json parser for data analytics," in *Proceedings of the VLDB Endowment*, vol. 10, pp. 1118 – 1129, Association for Computing Machinery, 2017-06. 43rd International Conference on Very Large Data Bases (VLDB 2017); Conference Location: Munich, Germany; Conference Date: August 28 - September 1, 2017.

[7] T. Döhmen, H. Mühleisen, and P. Boncz, "Multi-hypothesis csv parsing," in *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, SSDBM '17, (New York, NY, USA), Association for Computing Machinery, 2017.

[8] "Blockchain data api." https://www.blockchain.com/sv/api/blockchain_api. Accessed: 2020-04-16.

[9] "Blockchain data api." https://www.blockcypher.com/dev/bitcoin/blockchain-api. Accessed: 2020-04-16.

[10] "Sochain api." https://sochain.com/api. Accessed: 2020-04-16.

[11] "Requests: Http for humans." https://requests.readthedocs.io/en/master/. Accessed: 2020-04-16.