

note

146 views

Project, Part 2. [extended +24hr] Wednesday, March 14 at 11:59pm

cs152coin, Part 2: Deanonymization

Due: (please note the extended deadline) Wednesday, March 14 at 11:59pm

Background

As discussed in Part 1, addresses are a key component of cs152coin, as they are in real cryptocurrencies. Like in Bitcoin et al., addresses are random-looking strings rather than real names, and each user can create as many of them as they want. These two properties have led many people to believe that Bitcoin is anonymous in the sense that it is seemingly impossible to (1) connect an address to a real person, and (2) link together all of the addresses that belong to a single person.

In this second part of the project, you will demonstrate that this belief is mistaken. There are a number of techniques that can be used to deanonymize cryptocurrency users, but the most common method is based on the observation that **if two addresses A and B are both inputs to the same transaction, then it is likely that they both belong to the same user**. Moreover, if address B is later used as an input to another transaction along with inputs C, D, and E, then it is reasonable to surmise that the *cluster* of addresses A,B, C, D, and E are all linked to a common user. The intuition behind this heuristic is that in order to cobble together sufficient cs152coin to pay for something, a user might have to draw on the balances of more than one of their addresses.

To see the power of this attack, consider the case where Alice buys a T-shirt from you and pays you in cs152coin from address A. That means that there will be a transaction on the blockchain with A as an input and one of your addresses as an output. Since you shipped the T-shirt to Alice, you know her identity, and you also know that the that address A belongs to her. Ordinarily, that wouldn't be a problem because, by paying you, Alice chose to reveal her connection to A. However, using the attack above and starting with A, you could discover other addresses that likely belong to Alice. This could reveal significant information such as other transactions that Alice has participated in as well as an estimate of Alice's total net worth — the sum of the balances of all of the addresses linked to her. All of this information could be inferred simply because Alice chose to conduct a single small transaction with you.

This heuristic isn't perfect, of course. The method can't discover an addresses when it only appears by itself as the input to a transaction or when it is never used as a transaction input at all (e.g., when it is only used to receive a mining reward). In addition, the fact that two addresses are inputs to the same transaction doesn't guarantee that they are controlled by the same person even though that's the most likely case. Nevertheless, the attack seriously undermines cryptocurrencies' claims to anonymity. It seems that a currency where all of the transactions ever performed are public for anyone to inspect isn't so privacy-preserving after all.

Deanonymizing cs152coin

For Part 2 of the project, your task will be to augment the blockchain analyzer from Part 1 so that it finds clusters of addresses that likely belong to the same user. Your program should be able to accept a list of addresses as command line arguments. For each supplied address, it should be able to print not only its balance as before, but also the size of the cluster it belongs to, a list of all of the addresses in the cluster, and the sum of the balances of all of the addresses in the cluster. Here's an example:

```
$ ./cs152coin_part2 <blockchain-deanon.txt f716bd6a36845f3275739e398e397828 55a6da20e3227ac43033062ebd4da8a6
Number of blocks: 1000
Number of transactions: 48269
Average transaction value: 5.742292

Double spends:

Address: f716bd6a36845f3275739e398e397828
Balance: 5.216846
Cluster size: 6
Cluster balance: 88.373015
Addresses in cluster:
f716bd6a36845f3275739e398e397828
9f48187da2f8a3a8009bab43db01cd81
d4ca2d004f4a404798635353ed702dd
ab2c1f6971ddeb06ea3cc1218b5b2d5
36d9cb70ede4194eacdda97aee62d1b0
1c46cf28bba42ef25f33ac139e4375ee

Address: 55a6da20e3227ac43033062ebd4da8a6
Balance: 4.110627
Cluster size: 4
Cluster balance: 9.477080
Addresses in cluster:
55a6da20e3227ac43033062ebd4da8a6
117ea287db6613026be6056f9073f450
7d000fd0e37161e8b4d77e18d38f3ec1
ad4b6efe567cdf7ed4f6d5ff90bf6e4b
```

The files you need to get started are here ([cs152coin_part2.zip](#)). The zip file contains a complete working reference implementation of Part 1 and a new blockchain, blockchain-deanon.txt, to parse. (Note that this blockchain does not have any double spends.) It also includes a complete implementation of a hash table-based map ADT in hmap.c and hmap.h that you can use. It maps string keys to pointers to arbitrary values (of type void*).

Your job will be to build on this implementation to enable deanonymization by filling in the skeleton deanon.h and deanon.c files as well as some new functions in cs152coin.c.

Part 2A: Preprocessing for Deanonymization

Your first task is to make another full pass over the blockchain to record enough information about how addresses are related to each other to enable deanonymization later on. To do this, you should implement the deanon_preprocess function in cs152coin.c. In addition, storing information about how addresses are related to each other will require you to design and implement a new set of data structures in deanon.c and deanon.h. Two notable functions that you'll need to implement in deanon.c are add_address, which adds a new address to the deanonymization data structure if it's not already present, and add_associations, which takes in a list of transaction inputs and records that all of the input addresses are associated with each other.

Part 2B: Identifying Address Clusters

Once you've preprocessed the blockchain, your second task is to implement the get_cluster which, given an address, actually finds and returns the cluster of associated addresses that probably belong to the same user.

The `get_cluster` function should meet the following efficiency constraints: if the supplied address belongs to a cluster that has never been returned before, it is allowed to take $O(a + b)$ time where a is the total number of distinct addresses in the blockchain and b is the number of pairs of addresses that have at some point appeared together as inputs to a transaction. On the other hand, if the supplied address belongs to a cluster that has been returned before, then `get_cluster` should take constant — i.e., $O(1)$ — time on average. For example, suppose a cluster consists of addresses X and Y . The first time `get_cluster` is called and given one of these addresses — say, X — it is allowed to take $O(a + b)$ time. But, if `get_cluster` is later called and given Y as input, it should have saved the cluster from the first call and should return it in $O(1)$ time on average.

Finally, you will need to implement `cluster_size` and `cluster_balance` in `cs152coin.c`, which return the number of addresses in a cluster and the total balance of all of the addresses in a cluster respectively.

Submission Instructions

Fill in all of the functions marked "FILL ME IN" in `cs152coin.c`, `deanon.c`. In order to complete the assignment, you will also have to add fields to the `deanon_t` struct and will probably have to define new structs in `deanon.h` as well. **You should not need to modify any of the non-empty functions or the Makefile.**

Submit all of the `.c` and `.h` files as well as the Makefile in a directory named `cs152coin_part2` using Subversion. Please do not submit `blockchain-deanon.txt` via Subversion.

#pin

project2


assignment_itself

Updated 11 months ago by Adam Shaw and Ariel Feldman

followup discussions *for lingering questions and comments*

Resolved

Unresolved




Jeff Qiu

11 months ago

You should not need to modify any of the non-empty functions or the Makefile

Does that mean we can modify arguments in empty functions or leave them blank if we don't use them?




Ariel Feldman

11 months ago

Please don't modify the arguments of functions that you have to fill in. Other code expects them to have a certain signature.

Resolved


Unresolved



Zayne Khouja

11 months ago

Is there a reason that all the given parse functions and free functions in `cs152coin.c` are marked with "FILL ME IN"? As far as I can tell, this part of the project doesn't involve parsing.



Adam Shaw

11 months ago

I think those are just leftover comments. Please delete and/or ignore them.