Cryptocurrency Cabal: Problem Set 1

Alec Grieser

September 15, 2015

1. a. The transaction I recieved has ID

   c4187d0faf72e23a85c468d69a3e8f5b0aa3130f42b5cb4e1fcea216126b9924.

   b. The fees for the transaction were 0.0001 BTC, which is about $0.02 today.

   c. The block containing my transaction (block #371914) had an estimated total volume of 1,755.15212027 BTC, which was about $401,455.94 at the time of processing and $407,300.60 today.

   d. The first confirmation for a transaction occurs when the transaction is mined into its first block (block #371914 in this case). The transaction receives an additional transaction each time an new block is mined on top of that original block, so the third confirmation will occur when two blocks have been mined on top of the block containing our transaction. So, block #371916 was mined at 15:33:05 on August 28, 2015, while the transaction was made at 15:10:05 that same day, so it took about 23 minutes to get confirmed. This makes sense, as it takes about 10 minutes to add a new block to the block chain once the last one has been added. On average, it would then take about 5 minutes for a transaction to be added to its first block and ten minutes to add each additional block.

2. a. Assuming the instructors of this class aren't showing favorites, I will assume that each student recieved about $1 or about 0.0045 BTC (which was the amount I received). By following transactions forward, three other student addresses would be

   1G8Dbnesf35V7gKRuc3Cv4EeGCwWqMdXkE,

   1KEz1rFhEoy2vQV4zQcUCFWPYy7EkJyRcM,

   and

   15X2qKbsXxPHbgfFTc6UgXmETVe7fpBUzX.

   b. The chain going backwards is pretty clear until we get to address

   19WmbY4nDcjAEv6wb5rcd5E6MutVMXBZzy

   where there is a purchase of 0.2 BTC from

   14J6ep326owXDpc1waViGJNB1onSFy9eou

to the above address. This amount of bitcoin is then distributed among the addresses of different students. It seems reasonable that this was purchased by one of the instructors for the express purpose of distributing, so this is the most important source for our purposes. However, the coin can be further tracked down to an address which at one point (before being split) contained over 7,000 bitcoin (address 15giyR7xcUTSw3NHx1XQEnzj1gkU1KZcQ3). From there, the transaction web seems to be mainly internal to some exchange (with outputs being combined and broken apart fairly frequently to prepare for them to be sold).

c. If one follows the list of transactions from the starting point we identified, the transaction is relayed by bitcoin nodes in the following cities: New York (twice), Chicago (thrice), Glasgow (twice), Buffalo (thrice), Amsterdam (four times), Malmö, Tampa (twice), London (twice), Mountain View, Nuremberg, Cincinnati, Dublin, Ashburn, Liverpool, San Francisco (twice), Denver (twice), Frankfurt (twice), Berlin, Zürich, Auckland, Orlando, Barcelona, and Los Angeles. If we place these cities on a map and try and piece together where this single sender who originated transactions at all of the places might have been, we get a definite bend towards the Atlantic region (excepting the one transaction from Auckland), though we have to be careful because if there are more bitcoin nodes in one location than another, it may be the case that we see that first location appear more in this list even if the source is closer to the second. Notable in this list are Mountain View (home to Google), San Francisco (home to many technically-literate cryptocurrency early adopters/hipsters), and Ashburn (home to a fair number of cloud servers, including those for Amazon in the eastern United States). I don't know as much about Amsterdam or Germany, which were hit a fair number of times during these transactions, but it wouldn't completely surprise me if these countries, with fair minorities distrustful of increased government, had relatively large bitcoin adoption rates. But even with that in mind, the placement of these cities, with cities in the eastern half of the United States being used 12 times and those in western Europe 16 times, it would seem most likely that the original spender came from one of those cites. Though western Europe has an edge of the eastern United States in terms of strict numbers, there are also four cities hit in the western United States, so that would perhaps give the edge back to the eastern United States if we are searching for a source.

3. a. As a developer, it would always be possible to swipe the private keys of a wallet from an unsuspecting user and send them back to the developer's servers, thus giving them control of all of their users' coin. However, if the developer was then to make use of these funds to any significant amount, the user wouldn't have to be notified (the developer could just lie about how much money was in the wallet), but the user could always go check on their output's balance on a block chain monitoring site (like `blockchain.info`) and see that their coin had been depleted. If they don't expect users will often check these sites, this could be fairly effective.

   An alternative tactic would be to, with every incoming and outgoing transaction, surrepstitiously reroute some percentage of the coin being transferred to an address controlled by the developer. People are pretty bad at addition, so as long as any individual transaction seems correct, discrepencies in the total amount of coin owned

before and after a transaction probably wouldn't be noticed. (E.g., if a user had a wallet containing 210.23 BTC, if they recieved 0.42 BTC from someone else, they may not think twice if their new total is 210.61 BTC.) Alternatively, the developer could lie about the exact amount in the wallet, and the user wouldn't know the difference as long as they never attempted to overdraw their account. In any case, the developer just has to sit back and watch microtransactions roll in (exactly like in *Office Space*).

These are assuming that the developer wants to, over a long time, accrue a lot of bitcoin from many smalller transactions. An alternative approach would be to just have all of the wallets running a specific wallet implementation, send all of their bitcoin to addresses owned by the developer, resulting in a windfall. However, this approach would be noticed pretty quickly and might expose the developer's identity in the real world, leading to legal or extralegal repurcussions, so it might not be the wisest course of action to take.

b. I am moderatly confident that my money is safe from a malicious developer (at least for now), as I have verified from `blockchain.info` that the outputs credited to my address on the block chain are the same as those that Multibit HD says I have, nor have I seen evidence that any of those have been spent in mysterious ways. However, there could still be a Trojan Horse lurking and ready to steal all my coin, so I still have that to worry about. But also, this client has been recommended by several semi-trusted horses, including the course instructor and the Antonopolous book, so while it is possible both of these sources have been hoodwinked as well, the more independent sources who verify that the wallet is legitimate, the more confident I can feel that my money is secure.

If I were going to store all of my income in it, I would probably institute a few different precautions. The first would be see if I could examine the source code and, if possible, compile the code myself so that I can verify the stated behavior is the same as the programmed behavior. If the source code is considered unavailable or if there is obfuscated code within the source, these would be major red-flags. However, I could very easily miss something given the size of Multibit's source (I assume), so it would be good to have other precautions in place as well. First, I would probably set up a proxy to monitor traffic in and out of the wallet to make sure there isn't any activity going in and out when I don't expect. Also, it wouldn't be that hard to set up an indpendent service that monitored activities on the blockchain involving outputs I control to see if any fradulent use of my coin is occurring. (This would be similar to the way central banks monitor the activity of their users.)

4. Note that as $2^{256} = (2^4)^{64} = 16^{64} = 10^{64}_{16}$, which is simply 1 followed by zeroes in its hex representation. Using this fact, we can compute that the hex value given as the modulus in `btcec.go` plus $1000003D1_{16}$ will be equal to 2^{256}. (This was computed by taking each hex digit and substracting it from F_{16} except for the one's place, where the value was subtracted from 10_{16}.) But then

$$1000003D1_{16} = 1 \cdot 16^8 + 3 \cdot 16^2 + 13 \cdot 16 + 1 = 2^{32} + 2^9 + 2^8 + 2^7 + 2^6 + 2^4 + 1.$$

Thus, the value in `btcec.go` is equal to $2^{256} - 2^{32} - 2^9 - 2^8 - 2^7 - 2^6 - 2^4 - 1$ as expected.

5. Our hypothetical multibillionaire has to first trust that the mathematicians are correct in assuming that it is hard to perform the discrete logarithm. If they're wrong, then one could pretty easily loose the whole fortune. One also has to trust that the functions have been implemented correctly (and not intentionlly malicious either) by the bitcoin go client, and that the values for the point G the modulus p have been chosen so they don't form some kind of degenerate case that can be exploited by attackers. The generated key pairs in `keypair.go` rely on the go client to create the randomness used to create the new keys, so one has to trust that these keys are given enough entropy so that they are not predictably able to be reproduced. One would also have to trust whatever mechanisms are set up around `keypair.go` to store the private and public keys are secure enough that the key won't be stolen but also durable enough that the value could be recovered later, because nothing in `keypair.go` even attempts to save the created value.

6. Here is my vanity address creator:

```
1  func generateVanityAddress(pattern string) (*btcec.PublicKey,
2                                               *btcec.PrivateKey) {
3      pub,priv := generateKeyPair()
4      addr := generateAddr(pub)
5
6      for matched, err := regexp.MatchString(pattern, addr.String()) ;
7              !matched && err == nil ;
8              matched,err = regexp.MatchString(pattern, addr.String()){
9          pub, priv = generateKeyPair()
10         addr = generateAddr(pub)
11     }
12
13     return pub, priv
14 }
```

7. Here is the vanity address I generated:

   1Ge6TStAcr1aoGDWoGwePAMvEWeGALecSF

   (As you can see, it has my name, "ALec", towards the end. As an aded bonus, it ends with SF, which is where I'll be living next year, which is little free vanity I got.) It is generated from the following public key:

   02585a6914ff5909602f03e006879f9ca95de4025bdc57d387e8da933a7224aa68

8. The key I generated is not, in some respects, any more or less secure than any other public key as it is just as hard to generate a private key that will match the generated key as it is to generate one that will match any other. However, there are some possible advantages to be gained from having a vanity key over a regular key when it comes to imposters attempting to provide a fake address as mine. If I distribute my address with instructions to everyone to check that it ends with "ALecSF", if someone else wishes to propose a transaction that sends bitcoin to address similar to mine, they will

have to match this closing string and also the first five or six characters, which people usually check anyway when comparing addresses. This means I am adding an extra 5 or 6 letters that have to be matched by the attacker, which could add a significant amount of time to an attacker generating vanity keys to match mine.

9. I sent 0.0006 BTC (about $0.15) to my own vanity account. This is the following transaction on the blockchain:

   6ff0691c6cd2aaf16353f7301a9a5806f385cbf9c225d56e66b2e6afaec8e2c9

10. I sent the bitcoin that had been transferred to my vanity account to another student with the following transaction:

    71cdd7f482a02338acc032b93cc427c50f20406189dd510b583c33fe3f43ddd3

11. The code has been included in `spend.go` also uploaded to the Dropbox.