Michael Parisi (mjp9zn)
CS 4501 – Cryptocurrency Cabal
Fall 2015

Problem 1. Answer the following questions about the transaction where you received the bitcoin. If you received more than one transfer, include all the transaction IDs).

a. What is the transaction ID?

b. What was the transaction fee for the transaction? (Give your answer in BTC, as well as current approximate US dollar value.)

c. What was the total value of all the transactions in the block containing your transfer? (Note: https://blockchain.info provides this info conveniently, although you could compute it yourself)

d. How long did it take from when the transaction was received until it had 3 confirmations? (Include an explanation of how you estimated this in your answer.)

a. 050c90d69f039f61f619e8821a7b4491fdbd8d780b50bc97695a227e04d2b956

b. 0.1mBTC = $0.02

c. Estimated Transaction Volume = 1,440,126.59688mBTC = $326,390.29

d. Received time: 20:48:48
Included in Blocks: 20:57:27 (1st confirmation; Block 371658)
Block 371659: 21:03:27 (2nd confirmation)
Block 371660: 21:03:40 (3rd confirmation; very fast)
Total time: 00:14:52

Problem 2. See how much can you figure out about the way bitcoin was transferred to students in the class, starting from your transactions.

a. Identify the bitcoin addresses of what are likely to be other students in the class (you could potentially find all of them, but it is enough to find 3).

b. Trace back the source of the bitcoin as far as you can. Bonus points if you can figure out from which exchange the bitcoin was purchased and when.

c. (Bonus) Can you learn anything about where the send of the bitcoin is located geographically? (In this case, you have external information to know I'm in Charlottesville, but what could you learn about the sender's probable location just from the information in the blockchain?)

a. 112Kh8RHajxsj2yqdvrdcq5L4bLm8xymY (4.501 mBTC)
   1FM45Varjz955Sh9SrEp41XH3gRTcFM4i6 (4.5 mBTC)
   1G8Dbnesf35V7gKRuc3Cv4EeGCwWqMdXkE (4.501 mBTC)

b. 1AxtTbSeLopt9LkacZ9w8kmzqxLaFobhyj (4,000,000 mBTC)

c. This isn't technically in the blockchain, but blockchain.info stores the IP from which the transaction was first broadcasted. This isn't necessarily the sender's IP, but it may give a region from which the sender is likely to have sent their BTC from. That's all I could find.

Problem 3. Suppose a malicious developer wanted to distribute a bitcoin wallet implementation that would steal as much bitcoin as possible from its users with a little chance as possible of getting caught. (a) Explain things a malicious developed might do to create an evil wallet. (b) How confident are you your money is safe in the wallet you are using, and what would you do to increase your confidence if you were going to store all of your income in it?

a. Things a malicious developer might do to create an evil wallet include:
   - Changing the intended "transaction fee" to send to their wallet instead of being left out.
   - Display only a few decimal places in mBTC and send anything that wouldn't be shown to the developer's wallet at any transaction.
   -
b. I am very confident that my money is safe in the wallet that I'm using because I just saw it on Blockchain.info, and could check other blockchain recorders for confirmation. I might choose to use an open-sourced wallet that I could look through to ensure to funny business is going on.

Problem 4. Verify that the modulus used as `secp256k1.P` in `btcec.go` is correct. You can do this either using math/big, Go's bit integer library to do computations on such large numbers, or by computing it by hand. (For your answer, just show how you verified the modulus. Including a snippet of code is fine.)

I verified secp256k1.P by hand.

"FFF…FFF" = $2^{256} - 1$

"FFF…FFEFFFFFC2F" has 0's in the 4^{th}, 6^{th}, 7^{th}, 8^{th}, 9^{th}, and 32^{nd} places (if you count the first spot as place 0), which means you would subtract 2^{32}, 2^{9}, 2^{8}, 2^{7}, 2^{6}, and 2^{4}. Thus, secp256k1.P matches what is shown in the problem set.

Problem 5. What are *all* the things you need to trust if you are going to send money to the key generated by running `keypair.go`? You should assume that you are an ultra-paranoid multi-billionaire who intends to transfer her entire fortune to the generated address.

If I were an ultra-paranoid multi-billionaire, there would be too many things to trust for me to risk transferring my entire fortune. Just some include:

- That btcec.NewPrivateKay() actually generates a private key.
- That the alleged "private key" that was generated was actually shown to me.
- That the public key given corresponds to the so-called "private key" as well as the Bitcoin address.
- That there is no was that the code stored my private key somewhere else that can be accessed by a hacker or sent to someone through the Internet.
- That my computer is squeaky-clean and virus-free.
- The entire foundation of computing machinery.

Problem 6. Define a function, `func generateVanityAddress(pattern string) (*btcec.PublicKey, *btcec.PrivateKey)` where `pattern` is a regular expression. It should return a valid key pair where the corresponding public address matches the pattern.

```
func generateVanityAddress(pattern string) (*btcec.PublicKey, *btcec.PrivateKey) {
	done := 0;
	for done < 500000 { //just to give some kind of limit
		priv, err1 := btcec.NewPrivateKey(btcec.S256())
		if err1 != nil {
			log.Fatal(err1)
		}
		pub := priv.PubKey()
		addr := generateAddr(pub)
		match, err3 := regexp.MatchString(pattern, addr.String())
		if err3 != nil {
			log.Fatal(err3)
		}
		if match {
			return priv.PubKey(), priv
		}
```

```
                done += 1
        }
        return nil, nil;
}
```

Problem 7. Use your `generateVanityAddress` function to create your own vanity address. Its up to you to decide what to put in your vanity address, but it should be clear that your address is not a typical random one. If you are extra vain, create a address where your name appears at the beginning (after the initial `1`). (Note that uppercase 'O' and 'I' and lowercase 'l' are not used in any address, so if your name includes these letters you will have to be creative.)

I chose "mikep" since my name is Michael Parisi and didn't want it to take that much time. This is a private key in hex:
[f574235387d28b8eff56eac1f6a53eaa39fda432b3944b2fbf240b8547f70038]
This is a public key in hex:
[02dd55fbaa6ac45d6a277096bd9d93cc6cce59d571b62e993e9e0683d21207bc8b]
Bitcoin address: [1NMHTqvMGai8A1mSX**mikep**R4YUviokR9HR]

Problem 8. Is your vanity address more or less secure than the first address you generated?

Vanity addresses are as secure as any other BTC address generated another key pair. The amount of computation required to find the private key from an address is the same no matter what the address is.

Problem 9. Make a **small** (e.g., 1 mBTC) transfer from your wallet address to your vanity address. You can do this using your MultiBit wallet. Find the transaction in the blockchain (you can do this by searching for your vanity address at insight.bitpay.com or blockchain.info). You will need the transaction ID for the next exercise. (For your answer, just provide the transaction ID.)

Transaction id:
d780f86c2f0cd678b6596f39260683a5d8caffd54f0e04139deb43c52e157578

Problem 10. Transfer some bitcoin from your vanity address to someone else in the class (you can use one of the addresses you identified in Problem 2). To do this you can run `spend.go` in ps1. You can provide the parameters needed for the transaction at the command line (it is not necessary to modify the code).

Transaction id:
e8e71cb95c11b50d84ac3de476ffcce9e63fa1e7704842913e3b4f19dc6c6306

Problem 11. The provided `spend.go` code sends the full amount of the input transaction (less the network fee) to the destination address. Modify the program to add an `-amount <value>` flag that takes the amount to transfer in satoshi. If the amount available in the specified transaction output (less the network fee) exceeds the amount to send, your program should print an error message. Otherwise, it should send the requested amount to the toaddress, and send the change back to your address.

Code attached

Problem 12. (Bonus) Try to double spend the same bitcoin. Figure out as much as you can about what happens when the double spend transactions are attempted. See if you can get a transaction to appear on the list of double spends. See how close you can get to obtaining two verified transactions spending the same coin (e.g., can you achieve two transactions with at least one confirmation each?)

not attempted