Problem Set 1: Bitcoin Transactions

Cyrus Malekpour (cm7bv@virginia.edu)

Problem 1

a. b95e09c831546f49e8afbe4fdb8e81c77d7bd2f254e05d0d43a684059f854bc4

b. 0.0001 BTC ($0.02)

c. 0.085377 BTC (including fee), roughly $20.02

d. It took about 9 minutes and 25 seconds. This can be calculated because the block was received at 2015-08-28 18:29:14, and the third block after it was received was 2015-08-28 18:38:39. Each block provides a single confirmation, so the total time from receive to third confirmation is 9 mins and 25 seconds.

Problem 2

a. 3 addresses that are likely students

- 15j1jdJsMa4vR71gcZ6FCfYeAWJdPwpn7W
- 163vXnDXSc2hEKMrWHkERzBJWuuKJve5Nc
- 19Y4oNeGcdmBAfDXpJjPbiZ35PnZz57Ar2

b. The sum of bitcoin bought for the entire class appears to be 0.2 BTC. The 0.2 BTC was transfered to address 19WmbY4nDcjAEv6wb5rcd5E6MutVMXBZzy from 14J6ep326owXDpc1waViGJNB1onSFy9eou. The latter address appears to be owned by an exchange that is splitting up its funds.

c. No, I don't believe it's possible to determine the location of who sent the transaction. The blockchain doesn't encode any information about transaction location, and who your wallet sends the transaction to is random and not neccesarily based on where you are physically located.

The only way to get the real location is to be one of the first nodes that receives the transaction, but there is no way to know if you are communicating with the real sender or just someone else relaying it. Blockchain.info has geographic information, but that is only the IP geolocation of the node their network first received the transaction from, not the transaction's origin. If I wanted to hide my location, I could send the transactions to several nodes geographically spread across the globe, and nobody would be able to determine my true identity.

Problem 3

a. A malicious developer could create transactions that silently pass Bitcoin to an address the developer owns, and mark them as "transaction fees". It could also generate private keys that the developer knows (or is also able to generate), meaning they could steal your Bitcoin from any of your addresses. This could be as simple as using a poor source of randomness, or as complex as somehow relaying the key after it is generated. However, one of the benefits of Bitcoin are that all transactions are public. If my wallet were malicious, I couldn't stop it from stealing my bitcoin but I could the theft when it happens.

b. I am somewhat confident, because MultiBit is open source and verified. Other people are looking at the code and checking to make sure it's valid. However, if I was going to store significant income in my wallet, I'd need to take some further steps. I would download MultiBit's code, verify it is not malicious myself, and compile it on an offline machine that has never been connected to the internet. I would generate my private keys on that device, and store them in a paper wallet. I would only use each key/address for a single transaction and generate a new pair afterwards.

Problem 4

I used the following code.

```
package main

import (
    "fmt"
    "math/big"
)

/* Calculates base^exponent and returns a big.Int */
func calcBigNum(base int64, exponent int64)  *big.Int {
    start := big.NewInt(1)
    bigbase := big.NewInt(base)
    var c int64
    c = 0
    for (c < exponent) {
        start.Mul(start, bigbase)
        c = c + 1
    }

    return start
}

func main() {

    /* Pulled from secp256k1.P in btcec.go */
    value := "FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F"

    val := new(big.Int)
    val.SetString(value, 16)

    val.Sub(val, calcBigNum(2, 256))
    val.Add(val, calcBigNum(2, 32))
    val.Add(val, calcBigNum(2, 9))
    val.Add(val, calcBigNum(2, 8))
    val.Add(val, calcBigNum(2, 7))
    val.Add(val, calcBigNum(2, 6))
    val.Add(val, calcBigNum(2, 4))
    val.Add(val, big.NewInt(1))

    fmt.Printf("Final Value (0 = modulus is correct): %s\n", val)
}
```

Problem 5

If you're transferring a large amount of money to a `keypair.go` address, you will want to be sure the software really does what it claims to do. At a high level, you'll need to trust basic Go standard library features like the `encoding/hex` library or `fmt`. BTC Suite's implementation of `NewPrivateKey` must generate private keys using a strong source of cryptographic randomness, and avoid transmitting those keys or making them accessible to attackers. You will want to trust your computer to be secure and clear from malware that could spy on your keys. You'd also likely want to have a trustworthy method of storing the private keys. They should be kept somewhere offline and far away from (physical) intruders.

At a very basic level, you also need to trust that the discrete logarithm problem is hard and will remain hard for at least the forseeable future. You will need to trust that public keys can be derived from private keys, but it is difficult to do the reverse. The values used by Bitcoin for the generator point and modulus should be chosen to be secure, rather than for some ulterior motive.

Problem 6

The following function generates and returns a keypair for the given pattern

```
func generateVanityAddress(pattern string) (*btcec.PublicKey, *btcec.PrivateKey) {
    for {
        pubkey, privkey := generateKeyPair()
        address := generateAddr(pubkey)
        matches, err := regexp.MatchString(pattern, address.String())
        if err != nil {
            panic(err)
        }
        if matches {
            fmt.Printf("%s | %s\n", pattern, address.String())
            return pubkey, privkey
        }
    }
}
```

Problem 7

My vanity address contains the string "CYRUS", my first name.

```
Address: 14t1SegVFDmMiMhdBJ9CYRUSyXjN69Pbwa
Public Key: 039512a9de1280569ae33744b131c46bbca080da9e61f85a3fd0d6cd0a4512c2a3
Private Key: nope
```

Problem 8

Overall, it is slightly more secure. Mathematically, your address is not any less secure than other addresses. It is presumably generated with the same level of (strong) cryptographic randomness. However, because the address is "human readable", humans will tend to look at several more characters beyond the vanity section. (ex: if the vanity part has length 5, they will examine 8 or 9 characters). This raises the number of address characters an attacker would have to match to fool a human, putting it beyond the reach of what modern computers can do.

Problem 9

733645c27fb453ffa576ad66461244ecb656ae67d173f88341786bd80d9cff08

Problem 10

```
go run spend.go -privkey "[removed]" -toaddress "15j1jdJsMa4vR71gcZ6FCfYeAWJdPwpn7W"
-txid "733645c27fb453ffa576ad66461244ecb656ae67d173f88341786bd80d9cff08" -vout 0
```

Final txid was e1eff7361f2eb4cbf171b10835aa104531207df2239a2de5d5cad877bb5d4264

Problem 11

I have completed this in `spend.go`, which I am submitting alongside this document as `cm7bv-spend.go`. An example can be seen in https://blockchain.info/tx/ff6d432bac6f8f9d5a6cec8d5b317cc34f12e6ecc223187351fcc2e43ba877f9, where I sent some Bitcoin to one of my Multibit addresses and the change back to the sending address.