Cody Robertson
(ctr3sf)
Problem set #1

Problem 1:

a) The transaction ID is ab8230eab89fba2c07f8ef2c1b79ec733f2d41ca2f75b05e0247722a58771700.
b) The transaction fee for this transaction is .0001 BTC or .02 USD
c) The total transacted bitcoin is .034476 BTC or 7.95 USD with .006789 BTC or 1.57 of that transferring to my wallet.
d) For three confirmations, I would estimate that it took 30 minutes. If takes roughly 10 minutes for every block to be mined and confirmed, then for three confirmations it would take three 10 minute periods.

Problem 2:

a) The three addresses I found that seemed to be students were 1N96tMSyeeANTRFApr3zA6ML3Qow1gZR9A, 1NyBMbtqZLAmB3CSbjHzDHvedRJJJ7CupY, and 1fiWBi5u7oDzQrMNjeNLbmAvvHv545oy9.
b) The furthest back address that I traced it was 1Cr5Bh3E8Q4a9UdjAEAbFhk7TKWEuMBV5k, which seemed to me to be an exchange considering that it was dealing with bitcoin by the thousands and USD by the millions. I cannot tell what exchange it came from though.
c) I cannot find any directly useful evidence through blockchain.info or the transaction that is not any better than a random guess. I would have to have external information such as evidence on other websites, to link this to an identity. I do suppose I could watch for patterns in times when transactions were made and try to discover roughly what part of the world you may be in, but that is so broad that it would not be useful.

Problem 3:

a) A malicious developer could "increase" the transaction fee of each transaction, scraping a very small amount of BTC into their own wallet per transaction. Provided a user doesn't investigate heavily into the blockchain through an external source, it would be moderately straightforward to hide this information in the malicious client and a bit sneakier than outright stealing it from the client. One could also wait for small drops or rises in the bitcoin value to also mask a small transaction away from the wallet.
b) On the whole, I'm a little still wary of bitcoin as a secure currency. This wallet seems like it isn't trying to steal my bitcoin so I'm on decent terms with it. To make this wallet safer, I would definitely re-set up Multibit HD to use a strong, long, and random password as well as many more wallet words try to secure the wallet itself. I would still be super paranoid about making sure to personally follow the bitcoin to where it needed to go through third party blockchain exploration tools. The wallet would also be the only thing that I would use this computer for in the future to limit the chances of fatal errors or intrusions into the system.

Problem 4:

Code from the Go file I wrote to solve this is here:

```
package main

import (
	"fmt"
	"math/big"
	"encoding/hex"
)

func main(){
	const prec = 100
	firstpow := new(big.Int).Exp(big.NewInt(2), big.NewInt(256), nil)
	secondpow := new(big.Int).Exp(big.NewInt(2), big.NewInt(32), nil)
	thirdpow := new(big.Int).Exp(big.NewInt(2), big.NewInt(9), nil)
	fourthpow := new(big.Int).Exp(big.NewInt(2), big.NewInt(8), nil)
	fifthpow := new(big.Int).Exp(big.NewInt(2), big.NewInt(7), nil)
	sixthpow := new(big.Int).Exp(big.NewInt(2), big.NewInt(6), nil)
	seventhpow := new(big.Int).Exp(big.NewInt(2), big.NewInt(4), nil)
	one := new(big.Int).Exp(big.NewInt(2), big.NewInt(0), nil)
	answer := new(big.Int).Sub(firstpow, secondpow)
	answer.Sub(answer, thirdpow)
	answer.Sub(answer, fourthpow)
	answer.Sub(answer, fifthpow)
	answer.Sub(answer, sixthpow)
	answer.Sub(answer, seventhpow)
	answer.Sub(answer, one)

	fmt.Println("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFF
FFFC2F")
	fmt.Println(hex.EncodeToString(answer.Bytes()))
}
```

With the output being as followed:

```
FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F
fffffffffffffffffffffffffffffffffffffffffffffffffffffffefffffc2f
```

Problem 5:

I would have to believe that the private key was actually made private and no one else would be able to see it when I generated it. I would also want to prove that it was actually generated properly from ECDSC, rather than blindly trusting to be secure and not generated to be traced. I would also need a trusted expert to go through and verify to me that the libraries that I was using to generate my keys were secure and not containing malicious components. I would also need to verify that the way that the keypair.go's method of generating keys and addresses was also unique and impossible to duplicate.

Problem 6:

```
func generateVanityAddress(pattern string) (*btcec.PublicKey,
*btcec.PrivateKey){
	for {
		pub, priv := generateKeyPair()
		addr := generateAddr(pub)
		isMatch, err := regexp.MatchString("(?i)" + pattern,
addr.String())
		if err != nil {
			log.Fatal(err)
		}
		if isMatch {
			return pub, priv
		}
	}
}
```

Problem 7:

1codyRf1zL1NXdw6LGg4epxvMWwUvZw9v

Problem 8:

I think that as long as I am the one generating my own keys the vanity address is as secure as the original keys. Things get less secure when I constantly reuse the key multiple times to "show" as many people as possible. If I were to let an external vanity mining pool generate my keys, which would open the door to easy theft as has happened before as evidenced by the reddit post you liked to in the assignment. The vanity address does open me up to having my identity guess by my selection of included term.

Problem 9:

92b3c6848a2a4a1c40d25f4d0d255f431f27dfd14276a52b5b9b2dab89a605e4

Problem 10:

I actually cannot make the proper transaction with the unmodified spend.go file. I keep getting the same errors from two different processing APIs, Insight and Blockchain.info, as followed: They are transaction rejected by network (code -26). Reason 64: non-canonical and OP_EQUALVERIFY: non-equal data. I've spent the past hour just trying to get the bitcoin to transfer from the vanity wallet to someone else in the class.

Problem 11:

As a result of transaction not going through I am unable to test and really have time to properly code and test the new altered file, but here is a general rundown of what would be altered. A new flag would be created for the amount. That amount would added in place of the oldTxOut.Value for the createTxOut function call in the main method at line187. In the createTxIn() function, the proper checks to make sure one doesn't overdraft an account would be created here.