Joseph Tobin
Professor Evans
Cryptocurrency Cabal
15 September 2015

Problem Set 1

Problem 1.
a. The transaction ID is:
0b2212ca8330d1acfd5a7dbab29af1d264336590895a805a4de3c104207d42d8

b. The transaction fee was 0.0001 BTC (or $ 0.02 USD).

c. The total value of all the transactions in the block was 40,876.86249813 BTC.

d. I estimate it took 16 minutes and 28 seconds because the block containing the transaction has a timestamp of 2015-09-02 15:02:43 and the block that transacted three blocks later has a timestamp of 2015-09-02 15:19:11. Each subsequent block after the block containing the transaction verifies the transaction (therefore three confirmations equals three blocks later.

Problem 2.
a. Other id's of students likely to be in the class:
1LknwxoW83PZDZfaca6HSP3XCDeiD3t14S
1MeR6TjdB4yZqh3tPCDomrEDsKDTZMTYzL
1412h8TdLoAXbywvAfNC5SgcD1e4JiQtiG

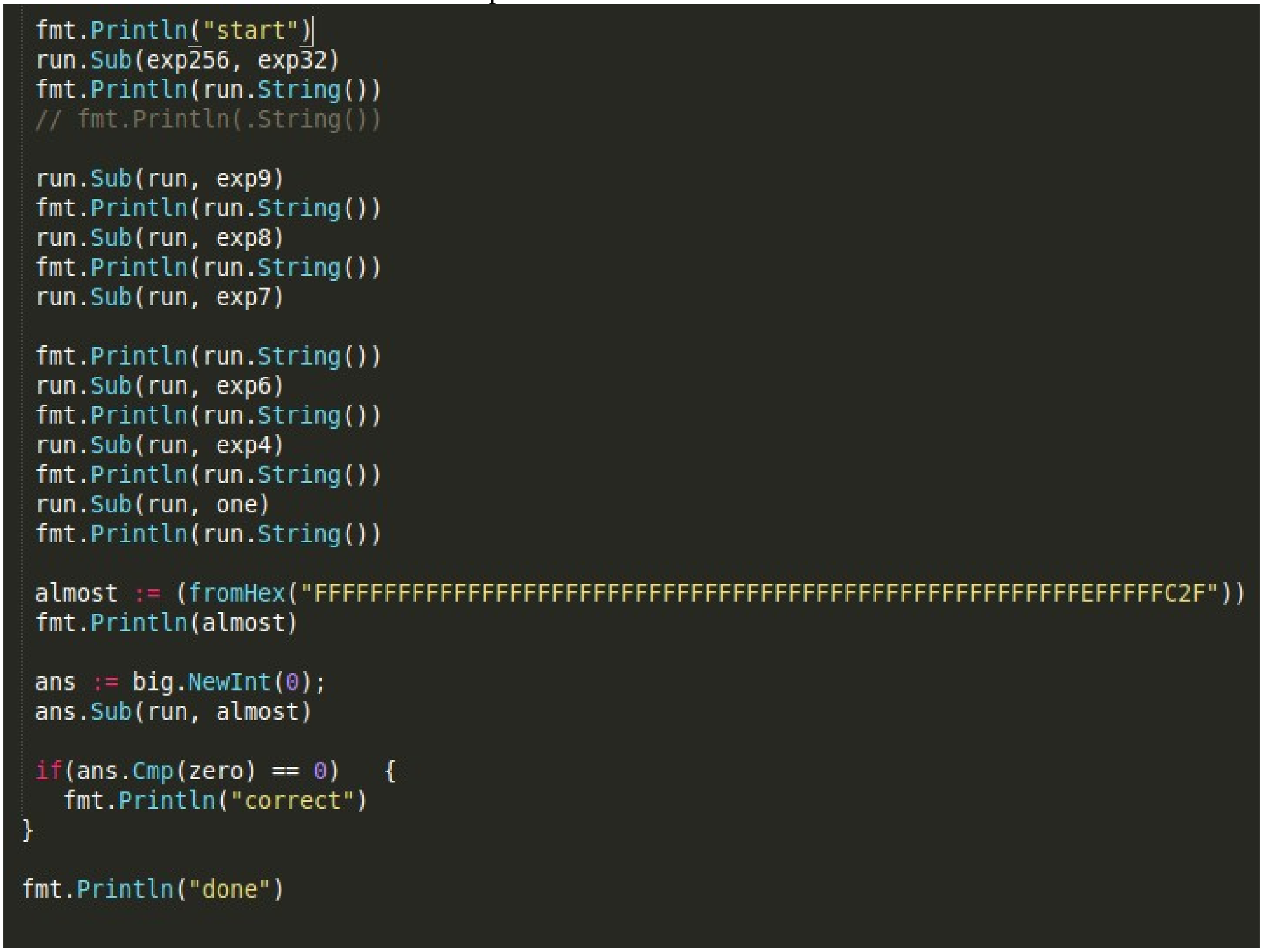
b. I was able to trace back the coin to transaction 634af12a282c4b4c1873abd6d30e4fa0703e865144742934aa221853a2a93765. Although nothing can be revealed through the ip and network propogation, it appears to be associated with the name “yanghe2”.

c. You can use the ip address (144.76.225.228). Unfortunately, nothing came up on “whois”. You can also look at the time it was sent and try to get a general estimate of what time zone/hemisphere the sender is from.

Problem 3.
a. To create an evil wallet, a developer should have the wallet send the wallet's public and private keys to the developer every time they are generated. With these keys, the developer has unlimited control over the wallet. A malicious developer could also not use a random function to generate keys and therefore predict the private keys of every user.

b. I am only confident that my money is safe in the wallet because there are not any complaints about its security online and Professor Evans recommended it. Therefore, I am only somewhat confident. To increase my confidence so that I could store all of my income, I would have to develop the wallet myself (and have it safely tested by others (who are more familiar with cryptosystems) through abstracting the sensitive details). If I could not develop the wallet, I would have to thoroughly examine the code behind the wallet to ensure there are no backdoors or leaks where a malicious attacker can steal my bitcoins. Also, I would be more confident if the government or another (relatively) trustworthy agency insured the money in my wallet.

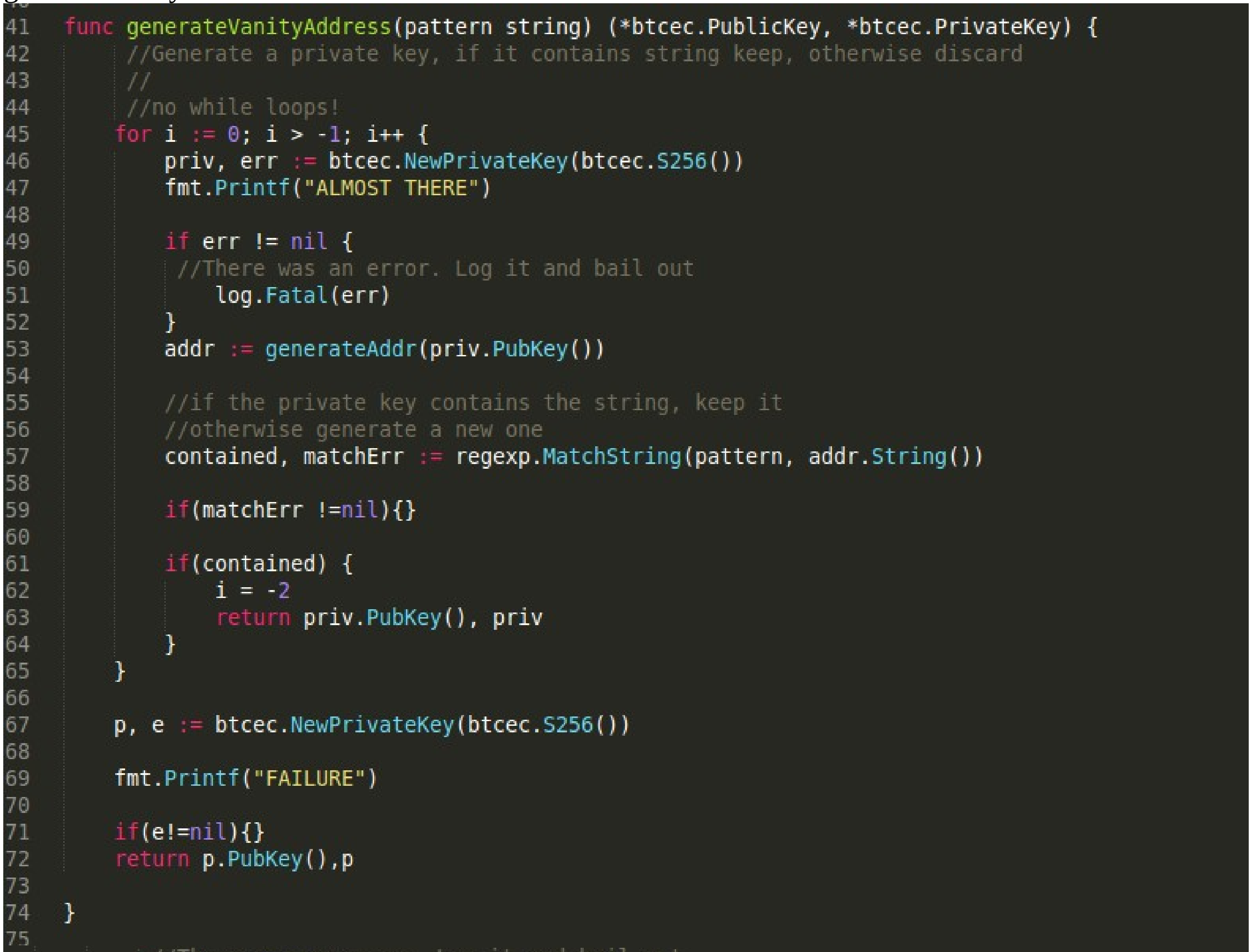
Problem 4.
The modulo was verified. Please not exp9 is 2^9.



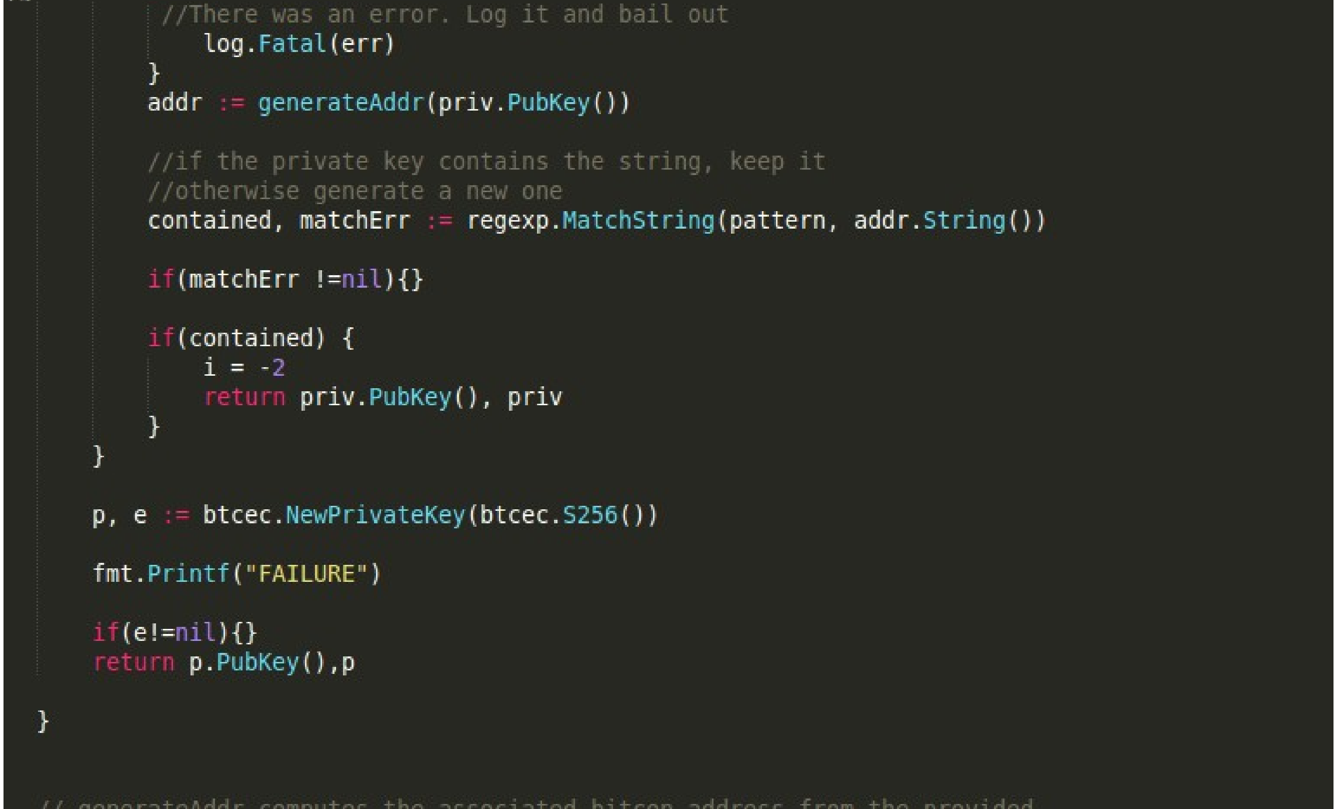
```
  fmt.Println("start")
  run.Sub(exp256, exp32)
  fmt.Println(run.String())
  // fmt.Println(.String())

  run.Sub(run, exp9)
  fmt.Println(run.String())
  run.Sub(run, exp8)
  fmt.Println(run.String())
  run.Sub(run, exp7)

  fmt.Println(run.String())
  run.Sub(run, exp6)
  fmt.Println(run.String())
  run.Sub(run, exp4)
  fmt.Println(run.String())
  run.Sub(run, one)
  fmt.Println(run.String())

  almost := (fromHex("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFFC2F"))
  fmt.Println(almost)

  ans := big.NewInt(0);
  ans.Sub(run, almost)

  if(ans.Cmp(zero) == 0)   {
    fmt.Println("correct")
 }

 fmt.Println("done")
```

Problem 5.
As an ultra-paranoid ultrabillionaire, you have to trust that the number generated is actually random (as in looking into “rand.Reader” to make sure it is producing actually random values (and not values that can be predetermined)). You also must trust that there is nothing in this code that is leaking the generated private key to another party (such as the developer). Finally, you have to trust that the given Elliptic Curve equation (“Koblitz Curve”) is extremely hard to solve and that no one will be able to solve for a extensive period of time.

Problem 6.
generateVanityAddress



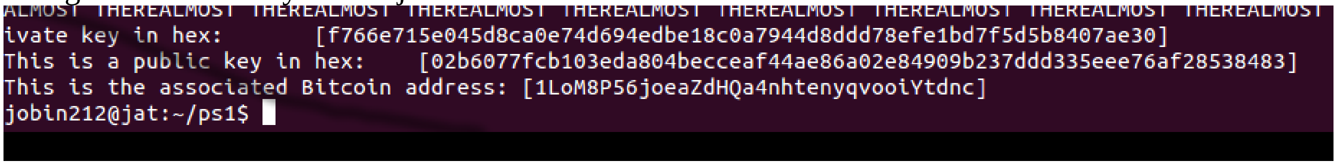
```go
func generateVanityAddress(pattern string) (*btcec.PublicKey, *btcec.PrivateKey) {
     //Generate a private key, if it contains string keep, otherwise discard
     //
     //no while loops!
    for i := 0; i > -1; i++ {
        priv, err := btcec.NewPrivateKey(btcec.S256())
        fmt.Printf("ALMOST THERE")

        if err != nil {
         //There was an error. Log it and bail out
            log.Fatal(err)
        }
        addr := generateAddr(priv.PubKey())

        //if the private key contains the string, keep it
        //otherwise generate a new one
        contained, matchErr := regexp.MatchString(pattern, addr.String())

        if(matchErr !=nil){}

        if(contained) {
            i = -2
            return priv.PubKey(), priv
        }
    }

    p, e := btcec.NewPrivateKey(btcec.S256())

    fmt.Printf("FAILURE")

    if(e!=nil){}
    return p.PubKey(),p

}
```



```go
         //There was an error. Log it and bail out
            log.Fatal(err)
        }
        addr := generateAddr(priv.PubKey())

        //if the private key contains the string, keep it
        //otherwise generate a new one
        contained, matchErr := regexp.MatchString(pattern, addr.String())

        if(matchErr !=nil){}

        if(contained) {
            i = -2
            return priv.PubKey(), priv
        }
    }

    p, e := btcec.NewPrivateKey(btcec.S256())

    fmt.Printf("FAILURE")

    if(e!=nil){}
    return p.PubKey(),p

}
```

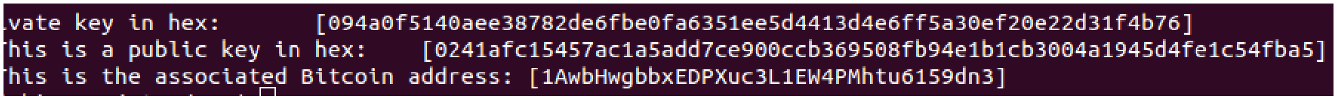
```go
// generateAddr computes the associated bitcoin address from the provided
```

Using method with my name “joe”



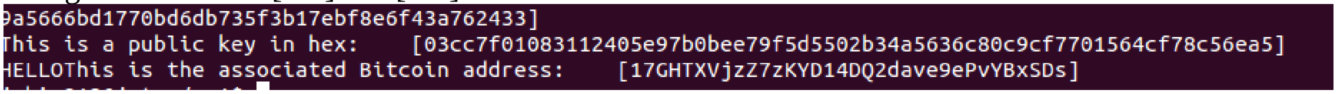
```
ALMOST THEREALMOST THEREALMOST THEREALMOST THEREALMOST THEREALMOST THEREALMOST THEREALMOST THEREALMOST
ivate key in hex:       [f766e715e045d8ca0e74d694edbe18c0a7944d8ddd78efe1bd7f5d5b8407ae30]
This is a public key in hex:    [02b6077fcb103eda804becceaf44ae86a02e84909b237ddd335eee76af28538483]
This is the associated Bitcoin address: [1LoM8P56joeaZdHQa4nhtenyqvooiYtdnc]
jobin212@jat:~/ps1$ 
```

Using method with "3.*1.*4.*1.*5.*9.*"



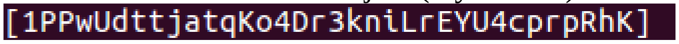
```
vate key in hex:       [094a0f5140aee38782de6fbe0fa6351ee5d4413d4e6ff5a30ef20e22d31f4b76]
This is a public key in hex:    [0241afc15457ac1a5add7ce900ccb369508fb94e1b1cb3004a1945d4fe1c54fba5]
This is the associated Bitcoin address: [1AwbHwgbbxEDPXuc3L1EW4PMhtu6159dn3]
```

Using method with “[0-9]dave[0-9]”



```
9a5666bd1770bd6db735f3b17ebf8e6f43a762433]
This is a public key in hex:    [03cc7f01083112405e97b0bee79f5d5502b34a5636c80c9cf7701564cf78c56ea5]
HELLOThis is the associated Bitcoin address:    [17GHTXVjzZ7zKYD14DQ2dave9ePvYBxSDs]
```

Problem 7.
Created address with name “jat” (my initials):

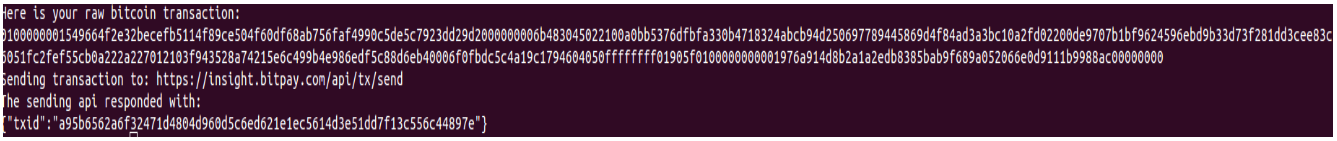


```
[1PPwUdttjatqKo4Dr3kniLrEYU4cprpRhK]
```

Problem 8.
The vanity address is more secure than the first address generated because you generated it more directly than the other address generated. Therefore, there is less chance of your private key leaking to a malicious attacker.

Problem 9.
Transaction number:
d229dd23795cdec59049af6f75ab68df604f50ce894f11b5efec2be3f2649654

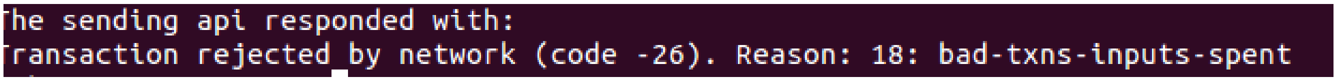
Problem 10.



```
Here is your raw bitcoin transaction:
0100000001549664f2e32becefb5114f89ce504f60df68ab756faf4990c5de5c7923dd29d2000000006b483045022100a0bb5376dfbfa330b4718324abcb94d250697789445869d4f84ad3a3bc10a2fd02200de9707b1bf9624596ebd9b33d73f281dd3cee83c
6051fc2fef55cb0a222a227012103f943528a74215e6c499b4e986edf5c88d6eb40006f0fbdc5c4a19c1794604050ffffffff01905f010000000000001976a914d8b2a1a2edb8385bab9f689a052066e0d9111b9988ac00000000
Sending transaction to: https://insight.bitpay.com/api/tx/send
The sending api responded with:
{"txid":"a95b6562a6f32471d4804d960d5c6ed621e1ec5614d3e51dd7f13c556c44897e"}
```

Problem 11.
Please see attached code.

Problem 12.
It appears the bitcoin network checks the transaction id to see if it has been confirmed, and rejects the double spend transaction if so.



```
The sending api responded with:
Transaction rejected by network (code -26). Reason: 18: bad-txns-inputs-spent
```