Alishan Hassan (ah2xt)
CS4501 - Cryptocurrency Cabal
Problem Set 1

1) a. Transaction ID: **59b5b36f19633243466e26b47364166b54cb67f134fede83fb602e744a42532c**
b. Transaction Fee: **.0001 BTC** - Approximately **0.02 USD**
c. Total Value of all transactions in block: **27,227.19856257 BTC**
d. 3 confirmations took about **30 minutes, or more precisely 28 minutes and 32 seconds**
It has taken 11 days, 6 hours, and 47 minutes for 1708 confirmations, averaging about 9 and a half minutes per confirmation.

2) a. **1FqQV1R3H8dftDupCjCoW7b6kSVxoBZBmh**
1K6JYqH78TiqT7AphzJDUmPQgAhkSNETdM
1J8em4SZRNY6PWwY691EvKapcTHJM73SeC
b. The farthest I was able to go is to the address **15NemgCM8u1V8Ch2WH4Zd6W3RVt7L1qvBs** to the date December 13, 2013.

3) a. One thing an evil developer might be able to do is alter the transaction fee to send a little bit to his own address upon every transaction. If it's small enough, the developer could completely fly under the radar and slowly accumulate bitcoins from several unsuspecting users.
b. To gain more confidence in the wallet, I would strive to make sure that other users aren't reporting any malicious activity. A program, like MultiBit, with a strong reputation automatically improves my confidence in the wallet. Before storing my entire income on the wallet, I would test it with smaller values, examining the blockchains for any discrepancies before committing all of my financial assets. It also helps if the program is open source, so that I, as a user, can verify there are no acts of deception in the code.

4)

```
package main

import(
    "fmt"
    "math/big"
)

func main(){
p := fromHex("FFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFFEFFFFC2F")
a := new(big.Int).Exp(big.NewInt(2),big.NewInt(256),big.NewInt(0))
b := new(big.Int).Exp(big.NewInt(2),big.NewInt(32),big.NewInt(0))
c := new(big.Int).Exp(big.NewInt(2),big.NewInt(9),big.NewInt(0))
d := new(big.Int).Exp(big.NewInt(2),big.NewInt(8),big.NewInt(0))
e := new(big.Int).Exp(big.NewInt(2),big.NewInt(7),big.NewInt(0))
f := new(big.Int).Exp(big.NewInt(2),big.NewInt(6),big.NewInt(0))
g := new(big.Int).Exp(big.NewInt(2),big.NewInt(4),big.NewInt(0))
h := big.NewInt(1)
i := new(big.Int).Sub(a,(new(big.Int).Add(b,(new(big.Int).Add(c,(new(big.Int).Add(d,(new(big.Int).Add(e,(new(big.Int).Add(f,(new(big.Int).Add(g,h))))))))))))
fmt.Println(p)
fmt.Println(i)
}

func fromHex(s string) *big.Int {
    r, ok := new(big.Int).SetString(s, 16)
    if !ok {
        panic("invalid hex in source file: " + s)
    }
    return r
}
```

Confirms value of:
115792089237316195423570985008687907853269984665640564039457584007908834671663

5) That the private key function actually uses elliptic curve cryptography and doesn't just present a preordained key. Furthermore, it has to be verified that the key isn't being remotely transmitted to another user, or if there's any packet sniffing going on with a completely unencrypted connection. You have to make sure that your computer is completely clean, preferably disconnected from any networks and isolated.

6)

```
func generateVanityAddress(pattern string) (*btcec.PublicKey, *btcec.PrivateKey) {
    for  {
        pub, priv := generateKeyPair()
        addr := generateAddr(pub)
        if(strings.Contains(addr.String(), pattern) ){
            fmt.Printf("This is a private key in hex:\t[%s]\n",hex.EncodeToString(priv.Serialize()))
            fmt.Printf("This is a public key in hex:\t[%s]\n",hex.EncodeToString(pub.SerializeCompressed()))
            fmt.Printf("This is the associated Bitcoin address:\t[%s]\n", addr.String())
            return priv.PubKey(), priv
        }
    }
}
```

7)
My vanity address: 17oWXAvtyNKuTRhQshanpRbR9P1PUTixH2
Uses "shan" (my shortened name, since "ALiSHAN" is still being generated as I type this) as the pattern.

8)
Technically, because my vanity address is generated using the same methods as the first address I generated, it has the same level of security (I simply selected one that happened to have my desired pattern included). The only issue is that, like a vanity license plate, it can be easier to identify an particular user, especially if he/she constantly uses the same vanity address.

9)
Transaction ID: 8a831ac9551bea54c3a0b3fa4aaf5953899db6a3076594b9ed60adfebae1ee67

10)

```
Here is your raw bitcoin transaction:
010000000167eee1bafead60edb9946507a3b69d895359af4afab3a0c354ea1b55c91a838a000000006b483045022100d8a344a99ae30e501c0a8695
9c385e6103f532281af4ff8c5c6348098d30c2a902201549552a61d465f2695db8b2efea9af06d1f38e08611114136d9ecb53235ba4012103d1d20f
57ae88b26fc674d4a375a11f0b7093e0abbc4811613a75c72a5ee79344ffffffff01102700000000001976a914bbec91f9ec644276044c18fb12a6
25979df949e688ac00000000
Sending transaction to: https://insight.bitpay.com/api/tx/send
The sending api responded with:
{"txid":"0963e6cb821faaec04443467bd1399c4c6d6d94aa3d85e3b2c26e47fae7a2359"}
PS F:\School\UVA\CS4501 - Cryptocurrency\ps1>
```

11)
Lines added:
var amount = flag.Int("amount", 0, "The amount in Satoshi you want to send.")

```
type requiredArgs struct {
        txid     *wire.ShaHash
        vout     uint32
        toAddress btcutil.Address
        privKey   *btcec.PrivateKey
        amount    uint32
}
```

```
if amnt < amount {
                        log.Fatal(err)
            }

oldTxOut := wire.NewTxOut(int64(amount), subscript)
```