Cryptocurrency Problem Set 1

1. Transaction
    a. Transaction ID
        i. 4c8491652ca97269db07fe0f8c75101fd636804fa336bc5d24911a827947536
    b. Transaction Fee
        i. .0001 BTC
        ii. $0.02
    c. Total transaction size
        i. 0.090638
    d. 20.38 minutes
        i. I calculated the time elapsed between the mined time stamp and the current time, then divided by the number of confirmations, and multiplied by three.
2. Tracking
    a. Classmates
        i. 1LknwxoW83PZDZfaca6HSP3XCDeiD3t14S
        ii. 1DSyzS5X6WmQbj3fZpGRXHL2yicqG6a8Q4
        iii. 1EEwCjPdxHPBGX3puriE7eVBnd1j1gRipW
    b. Highest block

        i. 1B4jhF7ChVxQcfEwhiMC6nwrT9hzECMHmR
3. A malicious wallet would send the user generated private keys to an outside party who can then use the private keys to make transactions on your behalf. Since the Bitcoin foundation vouches for the security and privacy of bread-wallet, I tend to trust the bitcoin foundation's recommendation. If the source code was inspected and verified by independent third-parties, I would feel even safer using the wallet.
4. Verified with wolfram alpha. Link here: http://www.wolframalpha.com/input/?i=2%5E256+-+2%5E32+-+2%5E9+-+2%5E8+-+2%5E7+-+2%5E6+-+2%5E4+-+1+in+hex
5. Trust
    a. The seed used is based on a random or highly chaotic input
    b. Finding the discreet log of a random elliptic curve element and the base point is hard
    c. The elliptic curve encryption program actually uses the seed to return a random enough point on the curve.
    d. Hard to find collision messages generating the same hash as the seed.

6. 

```go
//generate a vanity address based on a pattern specified in the argument of the function
func generateVanityAddress(pattern string) (*btcec.PublicKey, *btcec.PrivateKey) {


    pub, priv := generateKeyPair()

    matched, err := regexp.MatchString(pattern, hex.EncodeToString(pub.SerializeCompressed()))
    if err != nil {
            log.Fatal(err)
        }

    for !matched {
        pub, priv = generateKeyPair()
        _ = priv

        fmt.Printf(hex.EncodeToString(pub.SerializeCompressed()))

        matched, err = regexp.MatchString(pattern, hex.EncodeToString(pub.SerializeCompressed()))
        if err != nil {
            log.Fatal(err)
        }
    }

    fmt.Printf(hex.EncodeToString(pub.SerializeCompressed()))
    return pub, priv

}
```

7. Vanity
    a. 03fa52ca2c567159215aaf541900fff4fb3087d7cf5fd93bf2b9ba8849f**e7ed**433062441eb449306
    b. e7ed is my computing ID without "l" and "k" (elk7ed) since none of the generated keys contain those letters
8. Same security
    a. If you are running the vanity process yourself, the security level is the same. It is still a hard problem to extract the private key out of the vanity public key irrespective of whether the public key is a vanity key or not.
9. 2701eb00e72f2b522f2d9cf44af9132fa6492fda682b84fa344bdbb4b69d1d74
10. 2f1bb2b6a3a394d7ef01d2975657c98a256376838ee580a5feeda5f693fd4cac
11. Changes made to spend.go
    a. Added var amount to the initial command argument readme
    b. Added amount to the required structs
    c. Convert the command line input for amount into int64 before assigning value to args in line 99
    d. Add amount as an argument to createTxOut (line 217)
    e. Check if amount is greater than the last transaction minus the fee, if so, send fatal error (220)
    f. Use amount instead of outcoin as an argument to the function that assigns to txout (229)
    g. Add the actual command line value for amount as the argument for createTxOut in main (193)