Quentin Moore

Cryptocurrency Problem Set 1

1a) Transaction ID: **970a8fd2f2bae4aaf6ab0f94ce3dcaf588ebc56feb4ff454218f4c6e55c8331f**
1b) Fees = 0.0001 BTC ($0.02 USD)
1c) Total outputs: 0.080676 BTC
1d) The transaction had 3 confirmations after about 29 minutes. I calculated this by calculating the time elapsed from when the transaction was made, and dividing by the total confirmations presently. Then I multiplied by 3 in order to find an estimate for the time it took to get 3 confirmations.

2a) Some likely addresses of my classmates are: 19Y4oNeGcdmBAfDXpJjPbiZ35PnZz57Ar2, 1MtYZBtRw8XcnTgEY8qQphbddnwwFcoEXH, and 1JHoF2bak7KCST3SzeR7e1AwqDm4tiLJjt
2b) I believe the exchange address is 195xBgPZv81anT56H6HPhxbKjcDjWz9g7R. The bitcoin was purchased on 8/14/2015 from Kraken.
2c) I did a whois on the IP that first broadcast the transaction in order to learn more about the geographic location of the sender.

3a) A malicious developer could store a copy of your private keys and be able to control the funds in your wallet. The funds could be taken at any time, because the developer could act like you, using your private key.
3b) I am confident my bitcoin is safe in my Coinbase wallet, since so many people trust Coinbase with far greater amounts of bitcoin than I have (and I hope Coinbase doesnt become Gox :o, even the big exchanges can fall, but it is unlikely).

4) The last 3 hex digits are C2F. This translates to 110000101111. To get a string of all "F"s, the number represented would be 2^256 – 1. Looking at the "0"s, the subtracted powers are 4, 6, 7, 8, 9. Thus the modulus in secp256k1.P is verified to be 2^256 – 2^9 – 2^8 – 2^7 – 2^6 – 2^4 – 1.

5) You need to be sure you are the only person in control of the private key (the writers of the Go function need to have actually implemented what they said they implemented, without backdoors etc). This includes making sure that your hardware is not compromised, and that no attacker could see the key without you knowing. You also need to verify that the generation itself was using truly random numbers to generate the key (otherwise the encryption may be able to be cracked). You also need to trust that discrete log is sufficiently hard enough (otherwise encryption as we know it does not really work anymore)

6)

```
func generateVanityAddress(pattern string) (*btcec.PublicKey, *btcec.PrivateKey) {

	// Generate a private key, use the curve secpc256k1 and kill the program on
	// any errors
	priv, err := btcec.NewPrivateKey(btcec.S256())
		if err != nil {
			// There was an error. Log it and bail out
			log.Fatal(err)
		}

	check := false
	for check == false{
		priv, err := btcec.NewPrivateKey(btcec.S256())
		if err != nil {
			// There was an error. Log it and bail out
			log.Fatal(err)
		}
		addr := generateAddr(priv.PubKey())
		matched, regerr := regexp.MatchString(pattern, addr.String())
		if matched == true{
			check = true
			return priv.PubKey(), priv
		}
		if regerr != nil{
			log.Fatal(regerr)
		}
	}

	return priv.PubKey(), priv
}
```

7) Vanity address: 1EGLyTwTcKxmAF5xMBa17gXbFvBquentxo (notice “quent” at the end)

8) Vanity addresses are no more or less secure than any other address. I found this vanity address by generating key pairs randomly until the bitcoin address had “quent” in it, using the same methods to generate key pairs as I did when generating my first address.

9) Transaction ID: **8f1061d6bfe7242142c3db173244604746b671bdb00b7a4a397a698b06378f33**

10) Transaction ID: **4dc74138a6e9caf2cc9f0c3a6c5f31cb73da4a56a72e6f2988544d3864475b98**

11) Attached spend.go

12) Double spending occurs when someone tries to make multiple transactions in a quick period of time, before the first transaction is confirmed. This is a problem with accepting bitcoin transactions with 0 confirmations (often businesses will wait 1-3 confirmations before confirming receipt of the bitcoin).