In the name of God

Blockchain Technology and cryptocurrencies: Practical Homework #2

Due on December 25, 2019 at 23:59pm

Professor MohammadAli MaddahAli

Alireza Shirzad
95101847
ee.sharif.ir/∼alireza.shirzad

Problem3

a.

First we have to generate secret key, public key and adresses for Faraz, Ata and Shareholders and save them in config.py .

```
Faraz_private_key = CBitcoinSecret(
'cQEHmt4ErW7rsvZh8zjDMKrFAhzd2w2pA3NKBwQaeC3vwDQsv3nq')
Faraz_public_key = Faraz_private_key.pub
Faraz_address = P2PKHBitcoinAddress.from_pubkey(Faraz_public_key)

Ata_private_key = CBitcoinSecret(
'cQn3fLmNCMAwFqGUfjJ2YSLqBfUvtpfMdvncNPRALhnpHZJ3yaBn')
#n2LHdMH4MshPj2a5iTytPt8MFgbmuA7soQ
Ata_public_key = Ata_private_key.pub
Ata_address = P2PKHBitcoinAddress.from_pubkey(Ata_public_key)

ShareHolder1_private_key = CBitcoinSecret(
'cQjG59dJtUrGanTC4fhVcpe5bxnFy7toNWMcpUtG4BXAPSwJWFju')
ShareHolder1_public_key = ShareHolder1_private_key.pub
ShareHolder1_address = P2PKHBitcoinAddress.from_pubkey(ShareHolder1_public_key)

ShareHolder2_private_key = CBitcoinSecret(
'cUE9DHc28vAhrooq8i7PjXRHnsWcT231JyuutLYQBX8NtQbEh65u')
ShareHolder2_public_key = ShareHolder2_private_key.pub
ShareHolder2_address = P2PKHBitcoinAddress.from_pubkey(ShareHolder2_public_key)

ShareHolder3_private_key = CBitcoinSecret(
'cRimZ72WUCRa48HwV6UnBGtGhh783i8tk5JQx45GRV2e6SzuxDCs')
ShareHolder3_public_key = ShareHolder3_private_key.pub
ShareHolder3_address = P2PKHBitcoinAddress.from_pubkey(ShareHolder3_public_key)
```

Listing 1: New keys

Then we need to write a appropriate locking script for 3.1a transaction. We should use a conditional branching in order the transaction to be redeemed by condition **i** or **ii**.The condition to be checked is the number of items pushed on stack by the redeemer. so it's either 3 or more.

```
def MULTI_P2PKH_scriptPubKey(Ata_public_key,Faraz_public_key,ShareHolder1_public_key,
    ShareHolder2_public_key,ShareHolder3_public_key):
return [
OP_DEPTH, 3, OP_EQUAL, OP_IF, 2, Faraz_public_key, Ata_public_key, 2, OP_CHECKMULTISIG,
    OP_ELSE, 1, Ata_public_key, Faraz_public_key, 2, OP_CHECKMULTISIGVERIFY, 3,
    ShareHolder1_public_key, ShareHolder2_public_key, ShareHolder3_public_key, 3,
    OP_CHECKMULTISIG, OP_ENDIF
]

```

Listing 2: Locking Script

One kind of Redeeming script is like below:

```
def MULTI_P2PKH_scriptSig(txin, txout, txin_scriptPubKey, Faraz_private_key,Ata_private_key,
    ShareHolder1_private_key,ShareHolder2_private_key,ShareHolder3_private_key):
Faraz_signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
Faraz_private_key,)
Ata_signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
Ata_private_key,)
ShareHolder1_signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
ShareHolder1_private_key,)
ShareHolder2_signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
ShareHolder2_private_key,)
ShareHolder3_signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
ShareHolder3_private_key,)
```

```
12
13
14 return [
15 OP_0, Faraz_signature, Ata_signature
16 ]
```

Listing 3: Unlocking Script

b.

In this case we only need to make sure that all of the 3 shareholders and one of Faraz and Ata sign the transaction. so we can use CHECKMULTISIGVERIFY for the first condition and CHECKMULTISIG for the second condition.

```
1 def MULTI_P2PKH_scriptPubKey(Ata_public_key,Faraz_public_key,ShareHolder1_public_key,
      ShareHolder2_public_key,ShareHolder3_public_key):
2 return [
3 3, ShareHolder1_public_key, ShareHolder2_public_key, ShareHolder3_public_key, 3,
      OP_CHECKMULTISIGVERIFY, 1, Ata_public_key, Faraz_public_key, 2, OP_CHECKMULTISIG,
4 ]
5 #
```

Listing 4: Unlocking Script

Problem4

a.

We make use of the following code in wikipedia: **b.**

```
scriptPubKey: <expiry time> OP_CHECKLOCKTIMEVERIFY OP_DROP OP_DUP OP_HASH160 <pubKeyHash> OP_EQUALVERIFY OP_CHECKSIG
scriptSig: <sig> <pubKey>
```

Figure 1: TimeLock script

We can use the RETURN script to burn some coins and out a message on blockchain.

```
1 def def Message_P2PKH_scriptPubKey():
2 return [OP_RETURN, b"Happy Birthday Hamed"]
3 ]
```

Listing 5: Unlocking Script

And we get the following result:

String: Happy Birthday Hamed
Hex: 486170707920426972746864617920486d6564

Figure 2: The resulting embeded message in transaction

Problem5

a.

We need to make secret key with the file hash using the following code:

```
1 FileName = input("Please Enter the file name: ")
2 SelectParams('testnet')
3 article = open(FileName).read()
4 article_hash_Object = hashlib.sha256()
5 article_hash_Object.update(article.encode())
6 article_hash = article_hash_Object.hexdigest()
7 SalarSKey = CBitcoinSecret.from_secret_bytes(article_hash.encode())
8 SalarPKey = SalarSKey.pub
```

```
SalarAddress = P2PKHBitcoinAddress.from_pubkey(SalarPKey)
print("Private key: %s" % article_hash)
print("Address: %s" % SalarAddress)
```

Listing 6: Unlocking Script

now using this address we can do a normal transaction.

b.

There is a little change in the code to get the file input from multiple files:

```
FileNumbers = int(input("Please Enter the Number of files: "))
SelectParams('testnet')
article = ''
for i in range(FileNumbers):
FileName = input("Please ENTER a file name:")
article+open(FileName).read()
article_hash_Object = hashlib.sha256()
article_hash_Object.update(article.encode())
article_hash = article_hash_Object.hexdigest()
SalarSKey = CBitcoinSecret.from_secret_bytes(article_hash.encode())
SalarPKey = SalarSKey.pub
SalarAddress = P2PKHBitcoinAddress.from_pubkey(SalarPKey)
print("Private key: %s" % article_hash)
print("Address: %s" % SalarAddress)
```

Listing 7: multifileverify key generation

We get the multiple files and concatenate them. Then We calculate the hash and the process goes on like the previous section.

Problem5

We need to lock the transaction in a way that it could be redeemed only in two way: Either we have X and we can redeem it only by showing our signature, Or Both parties agree and provide their signature to redeem.

```
def coinExchangeScript(public_key_sender, public_key_recipient, hash_of_secret):
return [
OP_DEPTH, 3, OP_EQUAL,
OP_IF, OP_HASH160, hash_of_secret, OP_EQUALVERIFY, OP_CHECKSIG,
OP_ELSE, 2, public_key_sender, public_key_recipient, 2, OP_CHECKMULTISIG,
OP_ENDIF
]
```

Listing 8: Pubkey Script for atimic swap

As you can see we conditioned on the number of OP CODES in unlocking script. But there is a problem. The two unlocking scripts had equal sizes so I added another dummy variable in the beginning of coinExchangeScriptSig2 to differentiate between them. So we have :

```
def coinExchangeScriptSig1(sig_recipient,pub_recipant, secret):
return [
sig_recipient, pub_recipant, secret
]

 unredeemed
def coinExchangeScriptSig2(sig_sender, sig_recipient):
return [
OP_0, OP_0, sig_sender, sig_recipient
]

```

Listing 9: redeeming scripts for atomic swap

And we get the following results:

```
Alice swap tx (BTC) created successfully!
Bob swap tx (BCY) created successfully!
Alice redeem from swap tx (BCY) created successfully!
Bob redeem from swap tx (BTC) created successfully!
```

Figure 3: Result of Atomic Swap with Redeeming

```
Alice swap tx (BTC) created successfully!
Bob swap tx (BCY) created successfully!
Bob return coins (BCY) tx created successfully!
Alice return coins tx (BTC) created successfully!
```

Figure 4: Result of Atomic Swap with returning the fund