南开大学
Nankai University

南　　开　　大　　学

计　算　机　学　院

区块链实验报告

Ex2

徐俊智

年级：2022 级

专业：计算机科学与技术

指导教师：苏明

2024 年 10 月 22 日

目录

一、 实验要求

（a）生成一个涉及四方的多签名交易，这样交易可以由第一方（银行）与另外三方（客户）中的任何一方（客户）共同赎回，而不仅仅只是客户或银行。对于这个问题，你可以假设是银行的角色，这样银行的私钥就是你的私钥，而银行的公钥就是你的公钥。使用 keygen.py 生成客户密钥并将它们粘贴到 ex2a.py 中。

（b）赎回事务并确保 scriptPubKey 尽可能小。可以使用任何合法的签名组合来赎回交易至 faucet 地址，但要确保所有组合都有效。

二、 练习（a）

使用 keygen.py 生成客户密钥

```
xu@LENOVO:~/projects/Blockchain/Ex2$ python3 keygen.py
Private key: cPWj8567uEcuRE1Ac66EGhXHWM7gGjGa9pZvdQCZzxgw5fFmM7Lh
Address: n41dQCTge3eji1EFzC6PwLTjWSwTCR7bjL

xu@LENOVO:~/projects/Blockchain/Ex2$ python3 keygen.py
Private key: cV1wuvFe4sEVhLJZB1qeniMGddURrKLt1Lc3XCH93EN7VFFycubT
Address: mqA7UK6cQ3ZSUc2JPo7eMaSfLZBaCu9eCf

xu@LENOVO:~/projects/Blockchain/Ex2$ python3 keygen.py
Private key: cUfuUESnzdpjEsUFDwRN637bepy8UQzibNjaBUjyxNjNJNk7ofHn
Address: n3pbkEbskupLWyNeJz76sTPXxDJ9cyytEE
```

ex2a.py 脚本的目的是从我的一个 UTXO 中拿出 0.00000050 BTC，创建一个多签名脚本，要求在解锁交易时验证银行（我）的签名和三个客户之一的签名。

锁定脚本（多签名脚本）：

```
ex2a_txout_scriptPubKey = [
    my_public_key,
    OP_CHECKSIGVERIFY,
    OP_1,
    cust1_public_key,
    cust2_public_key,
    cust3_public_key,
    OP_3,
    OP_CHECKMULTISIG
]
```

这个脚本的意思是，谁能提供银行（我）和至少一个客户的签名，让这个脚本运行通过，谁就能花费这笔交易的比特币。由于创建签名只能使用银行（我）和至少一个客户的私钥，其他人创建的签名将无法通过这个脚本的验证，也就无法花费这笔交易。

设置交易参数。lab1 分割了 10 个 UTXO，并且用掉了第 1 个 UTXO。本次实验使用第 3 个 UTXO。设置发送金额为 0.00000050 BTC，那么那么手续费为 0.00000050 BTC 。

```
    amount_to_send = 0.00000050
    txid_to_spend = (
 '387f1017c5f92552d1104c57a923bd932db5f03729aba5609394c2296ed5c6d0')
```

```
    utxo_index = 2
```

调用 send_from_P2PKH_transaction 函数进行比特币交易。首先创建一个交易输出，指定发送的比特币数量和锁定脚本（多签名脚本）。然后创建一个交易输入，引用 UTXO，并生成交易输入的解锁脚本。最后通过 create_signed_transaction 函数将输入和输出组合成一个完整的交易，并通过网络广播，使其能够被确认并记录在区块链上。

在 create_signed_transaction 函数中，通过 VerifyScript 函数进行验证。

当有人试图花费前一笔交易的 UTXO 时，验证脚本涉及当前交易输入的解锁脚本和前一笔交易输出的锁定脚本的结合。

```
# 当前交易输入的解锁脚本
signature,
my_public_key,
# 前一笔交易输出的锁定脚本
OP_DUP,
OP_HASH160,
address_hash,
OP_EQUALVERIFY,
OP_CHECKSIG
```

我们分析一下验证脚本的执行堆栈状态图：

（1）当前交易输入的解锁脚本 (txin_scriptSig) 执行：

1. signature
   - signature 入栈
   - 堆栈状态:[signature]
2. my_public_key
   - my_public_key 入栈
   - 堆栈状态:[signature, my_public_key]

（2）前一笔交易输出的锁定脚本 (txin_scriptPubKey) 执行：

1. OP_DUP
   - 复制栈顶元素
   - 堆栈状态:[signature, my_public_key, my_public_key]
2. OP_HASH160
   - 计算栈顶元素的 SHA-256 哈希然后 RIPEMD-160 哈希
   - 堆栈状态:[signature, my_public_key, my_public_key_hash]
3. address_hash
   - 我的地址的哈希值 address_hash 入栈
   - 堆栈状态:[signature, my_public_key, my_public_key_hash, address_hash]
4. OP_EQUALVERIFY

- 弹出栈顶的两个元素，比较它们是否相等。如果相等，继续执行；否则，脚本失败。
- 堆栈状态:`[signature, my_public_key]`

5. `OP_CHECKSIG`

- 弹出栈顶的两个元素（签名和公钥），检查是否匹配。如果匹配，将 True 压入栈顶；否则，将 False 压入栈顶。
- 堆栈状态:`[TRUE]`

运行

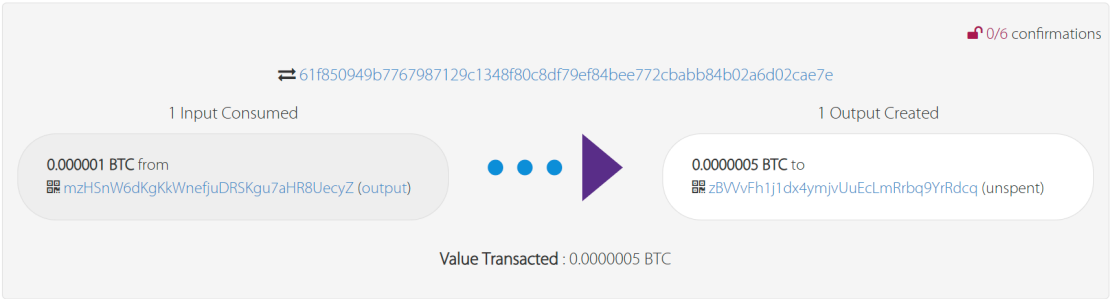
```
python3 ex2a.py
```

命令行输出交易信息

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "61f850949b7767987129c1348f80c8df79ef84bee772cbabb84b02a6d02cae7e
        ",
    "addresses": [
      "mzHSnW6dKgKkWnefjuDRSKgu7aHR8UecyZ",
      "zBVVvFh1j1dx4ymjvUuEcLmRrbq9YrRdcq"
    ],
    "total": 50,
    "fees": 50,
    ……
```

详情见《ex2a 交易输出结果.txt》

查看 https://live.blockcypher.com/网站的交易记录



三、 练习（b）

ex2b.py 脚本的目的是从上一个多签名脚本中赎回 UTXO ,要求在解锁交易时验证 `faucet_address` 的签名。

解锁脚本函数:

```
def multisig_scriptSig(txin, txout, txin_scriptPubKey):
```

```
    bank_sig = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
        my_private_key)
    cust1_sig = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
        cust1_private_key)
    cust2_sig = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
        cust2_private_key)
    cust3_sig = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
        cust3_private_key)

    # 选择两个签名（客户1和银行（我））
    # 签名的顺序必须和脚本里pubkey的放置顺序一致，不然会验证失败。
    # 由于历史原因，脚本的最开始要放置额外的一个值，一般放置OP_0。
    return [OP_0, cust1_sig, bank_sig]
```

设置交易参数，这里的 `txid_to_spend` 要用上一步新生成的。

```
    amount_to_send = 0.00000050
    txid_to_spend = (
'387f1017c5f92552d1104c57a923bd932db5f03729aba5609394c2296ed5c6d0')
    utxo_index = 0
```

调用 `send_from_P2PKH_transaction` 函数进行比特币交易。首先创建一个交易输出，指定发送的比特币数量和锁定脚本（多签名脚本）。然后创建一个交易输入，引用 UTXO，并生成交易输入的解锁脚本。最后通过 `create_signed_transaction` 函数将输入和输出组合成一个完整的交易，并通过网络广播，使其能够被确认并记录在区块链上。

在 `create_signed_transaction` 函数中，通过 `VerifyScript` 函数进行验证。

当有人试图花费前一笔交易的 UTXO 时，验证脚本涉及当前交易输入的解锁脚本和前一笔交易输出的锁定脚本的结合。

```
# 当前交易输入的解锁脚本
OP_0,
cust1_sig,
bank_sig,
# 前一笔交易输出的锁定脚本
my_public_key,
OP_CHECKSIGVERIFY,
OP_1,
cust1_public_key,
cust2_public_key,
cust3_public_key,
OP_3,
OP_CHECKMULTISIG
```

我们分析一下验证脚本的执行堆栈状态图：

（1）当前交易输入的解锁脚本 (`txin_scriptSig`) 执行：

1. `OP_0`
   - `OP_0` 入栈。
   - 堆栈状态:`[0]`

2. cust1_sig

   - cust1_sig 入栈。
   - 堆栈状态:[0, cust1_sig]

3. bank_sig

   - bank_sig 入栈。
   - 堆栈状态:[0, cust1_sig, bank_sig]

（2）前一笔交易输出的锁定脚本 (ex2a_txout_scriptPubKey) 执行:

1. my_public_key

   - my_public_key 入栈。
   - 堆栈状态:[0, cust1_sig, bank_sig, my_public_key]

2. OP_CHECKSIGVERIFY

   - 检查银行(我)的签名是否与公钥签名匹配。如果匹配,消耗 signature 和 [my_public_key, 继续执行；否则，脚本失败。
   - 堆栈状态:[0, cust1_sig]

3. OP_1

   - 1 入栈。
   - 堆栈状态:[0, cust1_sig, bank_sig, 1]

4. cust1_public_key

   - cust1_public_key 入栈。
   - 堆栈状态:[0, cust1_sig, bank_sig, 1, cust1_public_key]

5. cust2_public_key

   - cust2_public_key 入栈。
   - 堆栈状态:[0, cust1_sig, bank_sig, 1, cust1_public_key, cust2_public_key]

6. cust3_public_key

   - cust3_public_key 入栈。
   - 堆栈状态:[0, cust1_sig, bank_sig, 1, cust1_public_key, cust2_public_key, cust3_public_key]`

7. OP_3

   - 3 入栈。
   - 堆栈状态:[0, cust1_sig, 1, cust1_public_key, cust2_public_key, cust3_public_key, 3]

8. OP_CHECKMULTISIG

- 检查三个客户签名是否至少有一个与公钥签名匹配。如果匹配，消耗顶部的 3 和 3 个公钥（cust3_public_key,cust2_public_key,cust1_public_key）和 1 和 cust1_sig 和一个额外的 0；否则，脚本失败。
- 堆栈状态:[TRUE]

OP_CHECKSIG	0xac	交易所用的签名必须是哈希值和公钥的有效签名，如果为真，则返回1
OP_CHECKSIGVERIFY	0xad	与CHECKSIG 一样，但之后运行OP_VERIFY
OP_CHECKMULTISIG	0xae	对于每对签名和公钥运行CHECKSIG。所有的签名要与公钥匹配。因为存在BUG，一个未使用的外部值会从堆栈中删除。
OP_CHECKMULTISIGVERIFY	0xaf	与CHECKMULTISIG 一样，但之后运行OP_VERIFY

运行

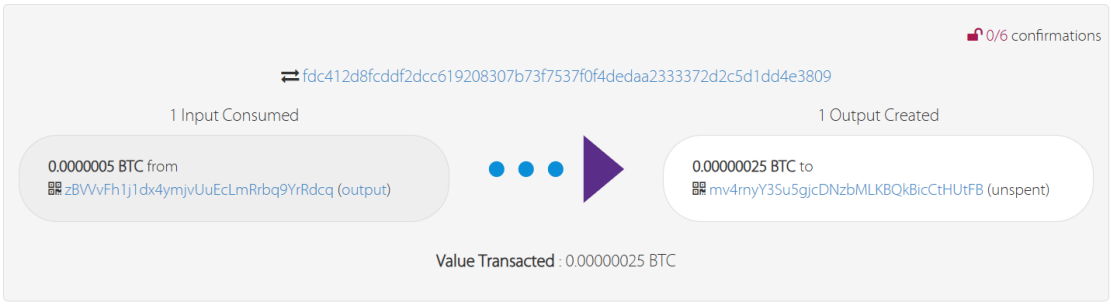
```
python3 ex2b.py
```

输出交易信息

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "fdc412d8fcddf2dcc619208307b73f7537f0f4dedaa2333372d2c5d1dd4e3809
        ",
    "addresses": [
      "zBVVvFh1j1dx4ymjvUuEcLmRrbq9YrRdcq",
      "mv4rnyY3Su5gjcDNzbMLKBQkBicCtHUtFB"
    ],
    "total": 25,
    "fees": 25,
    ……
```

详情见《ex2b 交易输出结果.txt》

查看 https://live.blockcypher.com/网站的交易记录


0/6 confirmations
fdc412d8fcddf2dcc619208307b73f7537f0f4dedaa2333372d2c5d1dd4e3809
1 Input Consumed
1 Output Created
0.0000005 BTC from
zBVVvFh1j1dx4ymjvUuEcLmRrbq9YrRdcq (output)
0.00000025 BTC to
mv4rnyY3Su5gjcDNzbMLKBQkBicCtHUtFB (unspent)
Value Transacted : 0.00000025 BTC