实 验 报 告

课程名称 密码学基础

学　　院 计算机科学与技术学院

专　　业 信息安全

姓　　名 管佳乐

学　　号 **16307130212**

开 课 时 间 **2018** 至 **2019** 学年第 **2** 学期

实验项目名　　称	一个简单的区块链钱包	成绩	

一、实验目的

1. 了解区块链
2. 实现一个简单的区块链钱包
3. 验证区块链交易的有效性

二、实验内容

1. 实现区块链钱包的功能
   1.1 能生成公钥，私钥和钱包地址
   1.2 支持查询余额
2. 验证每笔交易的有效性并可以生成新区块
   2.1 主要包括签名和验证，证明来源
   2.2 通过区块链的完整性，能够确保交易不被恶意篡改
   2.3 还应当符合来源少于等于支出多的要求

三、实验步骤

1. 文件说明
   1.1 实验环境
   Python 3.7.3
   PyCharm 2019.1.3
   1.2 使用到的库

```
pip3 install pycrypto
pip3 install ecdsa
pip3 install base58
```

   1.3 文件说明

./data	公钥和私钥存储的文件夹
./lib/account.py	管理账户的类（一个节点可以有多个账户）
./lib/chain.py	管理区块链的类（每个节点维护自己的类，并同步）
./lib/crypto.py	进行加密、散列、验证，计算节点地址的一些方法
./lib/network.py	维护网络数据，包括网络通信和节点的维护
./main.py	主函数，用户入口

2. 实现区块链钱包的功能
   2.1 钱包的类 Account 初始化的过程

```
def __init__(self, name: str):
    self.name = name
    self.pr_dir = './data/' + self.name + '.key'
    self.pu_dir = './data/' + self.name + '.pub'
```

```
        # 生成私钥和公钥
        self.pr = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1)
        self.pu = self.pr.get_verifying_key()
        # 以 bytes 形式保存
        self.pr_s = self.pr.to_string().hex()
        self.pu_s = self.pu.to_string().hex()
        # 计算得到对应的地址
        self.address = get_address(self.pu_s)
        # 把私钥和公钥存到本地
        with open(self.pr_dir, 'w') as file:
            file.write(self.pr_s)
        with open(self.pu_dir, 'w') as file:
            file.write(self.pu_s)
```

2.2 从公钥计算得到地址的函数

```
    def get_address(public_key: str) -> str:
        '''
        通过公钥计算得到交易的地址
        :param public_key: 公钥，以字符串的形式传输，对应 Account.pu_s
        :return: 交易的地址
        '''
        # SHA256
        h = SHA256.new()
        h.update(public_key[2:].encode('utf-8'))
        address = h.hexdigest()

        # RIPEMD
        h = RIPEMD.new()
        h.update(address.encode('utf-8'))
        address = h.hexdigest()
        address = '00' + address

        # Double SHA256
        checksum = double_sha256(address)[:8]

        # base58
        address = address + checksum
        address = base58.b58encode_int(int(address, 16))
        return address.decode('utf-8')
```

3. 生成新的区块

3.1 区块中 tx 字段的生成

一个转账的实例如下

```
"tx": [
  {
    "hash": "c075b2c3e292866ca3101bb60f575f2bdfc0d0588ac980dea8d66243ab0c0863",
    "in": [],
    "out": [
      {
        "n": 0,
        "recipient": "7APfAssCN987NbUioANSBJhpHBBUsJ8LW",
        "value": 20
      }
    ],
    "timestamp": 1560478516.4571671
  },
  {
    "hash": "21fac050125fe09acb99f1990e01cfdc22646549f70abb962008092d68d6ed21",
    "in": [
      {
        "prev_out": {
          "hash": "7ae26c8ef6f15731e877a9bf3a0574542335c5f9664e8da5d5dff3717dba89f9",
          "n": 0
        },
        "public_key": "9a6a2abdd5c38336ede95667d6370b6bf16d92f3cde7715c749ac6ab97ac3b39da53298e17461459d7d831e1eba83dccaa62da5ebd730a022163
        "sig": "c6daeee81f7813a5875506d1be1034f8fbdd920d1bdda520533b911f73e8f08911cd20cf9167b0df4b62a716fce7545adc36d69d621e3424e22ca0b98b8
      }
    ],
    "out": [
      {
        "n": 0,
        "recipient": "5GzskLTJz9CJM3EEZwNTr2J1HN2uqY151",
        "value": 12
      },
      {
        "n": 1,
        "recipient": "7APfAssCN987NbUioANSBJhpHBBUsJ8LW",
        "value": 8
      }
    ],
    "timestamp": 1560478513.250773
  }
]
```


其中转账的外部方法如下（具体实现见 ./lib/account.py）

```
    def transfer(self, destin: str, amount: int, utxo: dict):
        '''
        转账的对外接口，只需要提供转账的收款方和金额，以及需要更新的utxo
        需要检查 转账金额是否为负（不允许贷款）以及是否有足够多的金额完成转账
        :param destin: 收款方
        :param amount: 金额
        :param utxo:
        :return: None / 待广播的 tx 列表
        '''
```

通过这个方法，只需要提供付款方和金额，即可以完成转账。这个函数将对交易的细节进行验证，并生成一个 in 和 out 的列表，传给内部方法 new_transaction，完成资金合并

new_transaction 会根据 in 和 out 的列表，生成 tx 的字段

```
    def new_transaction(self, sources: list, destins: list):
        '''
        transfer 所调用的内部接口，构造生成一个 tx 记录
        考虑到合并支付和找零，in 和 out 字段都可能有多个记录
        :param sources: [(source_hash, source_index)..]
        :param destins: [(destin_recp, destin_value)..'
        :return: 待广播的 tx 列表
        '''
```

3.2 header 字段的生成

```
def new_block(self, account: Account):
        '''
        即为挖矿的过程，需要提供矿工的账号来记录报酬
        :param account: 矿工的账号
        :return:
        '''
```

```
            # 需要提供账号
            if account is None:
                print('No Account to do the mining')
                return None
            # 计算上一个节点的 hash 值
            if len(self.chain) == 0:
                hash_prev_block = 0
            else:
                hash_prev_block = double_sha256(json.dumps(self.chain[-1],
    sort_keys=True))
            # 当前需要记录的 tx
            txs = self.current_transactions
            # 报酬，20 元
            sources = []
            destins = [(account.get_address(), 20)]
            reward = account.new_transaction(sources, destins)
            txs.insert(0, reward)
            # 构造初步的 header（还需要计算 nonce）
            header = {
                'timestamp': time(),
                'hash_prev_block': hash_prev_block,
                'hash_merkle_root': get_merkle_tree_root(txs),
                'nonce': 0,
            }
            # 挖矿的过程
            proof = 0
            while self.valid_proof(header) is False:
                proof = proof + 1
                header['nonce'] = proof
            block = {
                'header': header,
                'tx': txs,
            }
            self.update_utxo(txs)
            # Reset the current list of transactions
            self.current_transactions = []
            self.chain.append(block)
            return block
```

4. 验证交易的有效性

   对于整条链的验证

```
    def valid_chain(self, chain: list) -> bool:
        '''
        验证给定的链是否是有效的，从以下几个方面验证
```

```
            1 每一个块的 hash_prev_block 是否和上一个块的散列值相符
            2 每一个块的 hash_merkle_root 是否和当前的 tx 符合
            3 符合工作量证明
            :param chain: 带验证的链
            :return: 是否有效
            '''
            current_index = 1
            while current_index < len(chain):
                header = chain[current_index]['header']
                tx = chain[current_index]['tx']
                # check header hash
                hash_prev_block = double_sha256(json.dumps(chain[current_index -
1], sort_keys=True))
                if header['hash_prev_block'] != hash_prev_block:
                    print('previous block unmatch')
                    return False
                # check merkle root
                hash_merkle_root = get_merkle_tree_root(tx)
                if header['hash_merkle_root'] != hash_merkle_root:
                    print('merkle root unmatch')
                    return False
                # proof of work
                if self.valid_proof(header) is False:
                    print('valid proof unmatch')
                    return False
            return True
```

对于一个块中交易的验证（在更新 UTXO 之前进行验证）

```
    def valid_tx_list(self, tx_list: list) -> bool:
            '''
            对区块的 tx 字段进行校验
            对第一个 tx，需要满足 完整性的验证 和 挖矿酬劳不超过 20
            对其他的 tx，需要满足 完整性的验证，in 字段签名能够通过验证，签名和来源一致，总的
    in 的金额不少于 out 的金额
            :param tx_list: 区块的 tx 字段
            :return: 是否满足要求
            '''
            # check the first tx
            checksum = tx_list[0]['hash']
            timestamp = tx_list[0]['timestamp']
            tx_input = tx_list[0]['in']
            tx_output = tx_list[0]['out']
            if tx_output[0]['value'] > 20:
                print('too much reward')
                return False
```

```
            text = str(timestamp) + json.dumps(tx_input, sort_keys=True) +
    json.dumps(tx_output, sort_keys=True)
            if checksum != double_sha256(text):
                print('tx checksum failed')
                return False
            # check following
            for i in range(1, len(tx_list)):
                checksum = tx_list[i]['hash']
                timestamp = tx_list[i]['timestamp']
                tx_input = tx_list[i]['in']
                tx_output = tx_list[i]['out']
                input_sum = 0
                output_sum = 0
                text = str(timestamp) + json.dumps(tx_input, sort_keys=True) +
    json.dumps(tx_output, sort_keys=True)
                if checksum != double_sha256(text):
                    print('tx checksum failed')
                    return False
                for j in tx_input:
                    prev_out = j['prev_out']
                    text = json.dumps(prev_out, sort_keys=True)
                    input_sum += self.get_out_value(prev_out['hash'],
    prev_out['n'])
                    # 是否拥有这笔钱
                    if get_address(j['public_key']) !=
    self.get_out_recipient(prev_out['hash'], prev_out['n']):
                        print("recipient unmatch")
                        return False
                    # 验证签名
                    if verify_sig(msg=text, signature=j['sig'],
    pu_s=j['public_key']) is False:
                        print('sig verification failed')
                        return False
                for j in tx_output:
                    output_sum += j['value']
                # 验证数量
                if input_sum < output_sum:
                    print('input cannot cover output')
                    return False
            return True
```

5. 关于 UTXO

```
{
  "7c5d5f01e3c6fcf33d586dc5493bc7bd9e6e807a6b256f8e66e6a0e147007531": [
    {
      "from": 0,
      "n": 0,
      "to": "EQ3roGi3braCv7TtEkkXsuNfJA9gE4Qg4",
      "value": 20
    }
  ],
  "e4baabf008137b3a978954e8dea1d53eaa51d7c3f51b0bb9b8d5326fda6afff8": [
    {
      "from": 0,
      "n": 0,
      "to": "EQ3roGi3braCv7TtEkkXsuNfJA9gE4Qg4",
      "value": 20
    }
  ]
}
```


实验中使用了 UTXO 来进行余额的计算，并且以原 tx 的 hash 值来作为键值进行存储，方便查找

```
    def balance_n_records(self, utxo):
        '''
        根据 utxo 来计算用户的余额 和 当前为花费的 tx 的对应 hash 和 n 值的列表
        :param utxo:
        :return: 余额, 未花费 tx 的索引
        '''
        total = 0
        records = []
        for tx_hash in utxo:
            for item in utxo[tx_hash]:
                if item['to'] == self.address:
                    total += item['value']
                    # (hash, n, value)
                    records.append((tx_hash, item['n'], item['value']))
        return total, records
```

因此在每生成一个包的时候，就应该对 UTXO 进行相应的更新

```
      def receive_block(self, block: dict):
        '''
        收到其他节点的块, 需要进行验证
        然后更新本地相关的数据, 如 utxo 和当前的 tx
        :param block: 收到的块,
        :return: None
        '''
        if self.valid_proof(block['header']) is False:
            print("receive a false block")
            return None
        if len(self.chain) != 0:
            if block['hash_prev_block'] != double_sha256(json.dumps(self.chain[-1],
sort_keys=True)):
                print("receive an unmatch block")
                return None
        self.update_utxo(block['tx'])
```

```
        # current_transaction -= block['tx']
        for tx in block['tx']:
            for ty in self.current_transactions:
                if tx['hash'] == ty['hash']:
                    self.current_transactions.remove(ty)

        self.chain.append(block)
```

6. 网络的策略

   挖矿和交易时进行广播

   如果收到了一个块，则应该根据其链长进行处理
   如果刚刚好是下一个包，那么就通过 receive_block 来进行验证，从而加入到链中
   如果收到的包是之前的包，那么对方的链长可能较短，那么把自己的链提供给他
   如果收到的包是后面的包，那么自己的链长可能较短，则需要更新自己的链

```
        elif msg['type'] == 'broadcast_block':
            # check the index
            if msg['index'] == len(BC.chain):
                # got a chance to be verified and accepted
                BC.receive_block(content)
            elif msg['index'] < len(BC.chain):
                # warn that the other chain is too short
                response_chain(address, msg['index'])
            elif msg['index'] > len(BC.chain):
                # we need to request a new chain
                request_chain()
```

四、实验结果及分析

1. 钱包的生成
   其中 name 域不参与交易，只是用来作为存储密钥的文件名
   实验里面，chiale 的工作端口为 8000，其他用户的端口在 8001-8005 中，也可以在菜单中修改

```
➜ blockchain python3 main.py
Port:8000
Working on 127.0.0.1 : 8000
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>1
1 View Current Account
2 View Balance
3 Create New Account
>3
Input your account name:chiale
private key: 95f75a441c1d753f5bbfd2f5b3916429837f4daaf198b364dadcdb1e6a0044b4
public key: d3e886c342b030da183f3f360fd5238473f44bcb2a9e0e70807cc8ea9594669c003d663a88d7d47424c755f1ee5331d3931fa5d9e8bab3594622
bb96c23b8473
address: NNdkBvQ9N8kwMXPk5YQTYUuMk38SqGwUX
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>1
1 View Current Account
2 View Balance
3 Create New Account
>2
Balance: 0
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>
```

2. 转账和验证

2.1 尝试转账

第一次转账，由于钱不够，转账失败，说明余额检查起到了作用

```
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>3
input the payee's address:P5jjGC2nc2reLQzDvpGiirt3upVPgB4gK
input the amount:20
Only remaining: 0
transaction failed
```

2.2 资金的合并

第二次，挖了两次矿之后，余额为 40，转账 25。把两笔 20 合起来，进行了资金的合并，然后给自己 15，给对方 25。

```
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>2
Balance: 20
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>2
Balance: 40
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>3
input the payee's address:P5jjGC2nc2reLQzDvpGiirt3upVPgB4gK
input the amount:25
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>1
1 View Current Account
2 View Balance
3 Create New Account
>2
Balance: 15
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
```

从 UTXO 也可以看出来这个结果

```
"a0cca25d3bc1569b47f328ac75c0f939fd328be9c6c957c80ae755c702f757a0": [
  {
    "from": "84ag685WwAZ8RRBzJ2j5rPN1crVRP9H86",
    "n": 0,
    "to": "P5jjGC2nc2reLQzDvpGiirt3upVPgB4gK",
    "value": 25
  },
  {
    "from": "84ag685WwAZ8RRBzJ2j5rPN1crVRP9H86",
    "n": 1,
    "to": "84ag685WwAZ8RRBzJ2j5rPN1crVRP9H86",
    "value": 15
  }
]
```

2.3 到对方的账户验证

但是到对方的账户上去看，钱却没有收到

```
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>1
1 View Current Account
2 View Balance
3 Create New Account
>2
Balance: 0
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
```

因为当前还没有人记账，所以先进行一次挖矿

```
>2
Balance: 35
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
```

自己的余额也从 15 变成了 35，然后到对方去查询

```
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
>1
1 View Current Account
2 View Balance
3 Create New Account
>2
Balance: 25
1 Account       2 Mine  3 Transfer      4 Node  5 Update        D Debug E Exit
```

可以看到，交易已经被对方认可了，通过了对方的验证，进入到了对方的 UTXO 中，从而余额变成了 25

在对方的 chain 里面，可以看到我们挖矿和转账的记录


```
      "hash_merkle_root": "8da7ac2ccc73825f6282a341504abf7862b6cc3a4b4b52e2ea35803793f74a20",
      "hash_prev_block": "1c6b67081de227c94697e62b487e504780a808b0fd6f42856d09b4b2444edfa8",
      "nonce": 10366,
      "timestamp": 1560484515.200309
    },
    "tx": [
      {
        "hash": "850ed913298b4ac816de4d39a2a7c0bf559d3d592a5737409bc2826cd6f0c25f",
        "in": [],
        "out": [
          {
            "n": 0,
            "recipient": "84ag685WwAZ8RRBzJ2j5rPN1crVRP9H86",
            "value": 20
          }
        ],
        "timestamp": 1560484515.200263
      },
      {
        "hash": "a0cca25d3bc1569b47f328ac75c0f939fd328be9c6c957c80ae755c702f757a0",
        "in": [
          {
            "prev_out": {
              "hash": "7172e801956a540cbf89295ba5110a9f1e41f30e09a435be4a64bc378487d223",
              "n": 0
            },
            "public_key": "06b75d0a1ffe7b694dc42a0356cbc15ac7080c9f412a4ca5eaff7257ec2095dd91848bed884c9419e9cf9cda9be3c0b9331e5
96c943060672ae100b6170a0690",
            "sig": "027404a3ff3c5cb6c1259429173433e5f95951f5c689abcbe166d75448f2e952f4dc4387d87a0a85556b23dcd5ec3b02b758c2a8eeb2
4673ad08cbb2e6d674cc"
          },
          {
            "prev_out": {
              "hash": "723dbda27c289697c568e9ee480abdcdfcf063c482d49157da39ad28103be62b",
              "n": 0
            },
            "public_key": "06b75d0a1ffe7b694dc42a0356cbc15ac7080c9f412a4ca5eaff7257ec2095dd91848bed884c9419e9cf9cda9be3c0b9331e5
96c943060672ae100b6170a0690",
            "sig": "c2c1bf7e17c3ec0e7bb3d8812435e445958fa3dcdd8b9b3a69a3daec8476fe009c467f114a5092a8bc95cd5086f15c2ac79a183ccb18
31236c120f9cf8f1e2a4"
          }
        ],
        "out": [
          {
            "n": 0,
            "recipient": "P5jjGC2nc2reLQzDvpGiirt3upVPgB4gK",
            "value": 25
          },
          {
            "n": 1,
            "recipient": "84ag685WwAZ8RRBzJ2j5rPN1crVRP9H86",
            "value": 15
```


五、实验总结

1. 对于区块链的体会
   区块链的实验代码量比较大，即使是花了很多时间，也只实现了一个十分简单的区块链。即便如此，也能够体会到区块链技术的高瞻远瞩，能够更好地理解课上所讲的区块链的核心的知识
2. 实验的细节
   在挖矿的回报当中，我最初的实现没有 in 域，而每次挖矿 out 域是一样的，所以造成 hash 值也是同样的，因而对转账的处理产生了影响，所以我在每个 tx 中也加入了时间戳，来解决这个问题
   在实验当中，最开始 UTXO 是使用 list 实现的，改用 dict 之后实现更加简单，也更方便进行测试修改
3. 先确定接口，再实现
   本来说是工欲善其事必先利其器，写了一大堆类。结果到主干的时候，不仅需要的接口变了，甚至连数据结构都变了。很多函数不得不重写甚至完全删掉。本来向考虑怎么把程序写得尽善尽美，在前期操很多心。但是到了实现的时候，还是会出很多问题，所以前期不要过于注意，保持一定的警惕就好，根据输出来反馈会比较容易。在这个基础上再进行优化也会更加容易