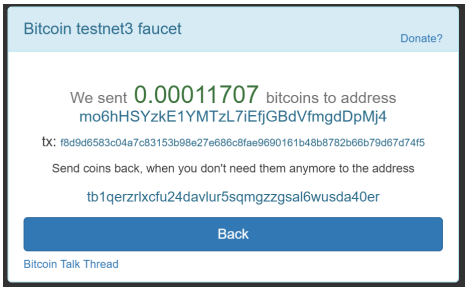
Blockchain2024 - 第一次作业

该文档内记录第一次作业 Ex1 中的交易、代码运行输出结果，以及相关截图。

使用 keygen.py 生成一个 testnet 私钥和地址，在 faucet 中获取可消费的输出，结果如下：



可以看到 faucet 向地址 `mo6hHSYzkE1YMTzL7iEfjGBdVfmgdDpMj4` 发送了一定数量的比特币，该地址即通过 keygen.py 生成的地址。上图还显示了该交易的 Hash 值，即 `tx: [f8d9d6583c04a7c83153b98e27e686c8fae9690161b48b8782b66b79d67d74f5`。在 Blockcypher 查询该交易的状态

可以看到该交易有一个输入和两个输出。

由于我们希望有多个可花费的输出来完成后面的练习，所以用 split_test_coins.py 来将已有的比特币拆分为多个输出，并重新转移回原来的地址。完成 split_test_coins.py 中的 TODO：

```
######################################################################
# TODO: set these parameters correctly
amount_to_send = <自由设置> # amount of BTC in the output you're splitting minus fee
txid_to_spend = (
    '<上一次交易的 Hash 值>')
utxo_index = <上一次交易的输出中某个 UTXO 的索引>
n=<自由设置> # number of outputs to split the input into
######################################################################
```

其中 `amount_to_send` 自由设置，该值与总输入的差额被视为交易费用；`txid_to_spend` 要是上一步中交易的 Hash 值，即从 faucet 获取比特币的交易 Hash 值；根据 Blockcypher 给出的信息，faucet 获取比特币的交易是第一个输出，所以 `utxo_index` 为 0；防止后续实验中出现错误，设置 `n=10`。

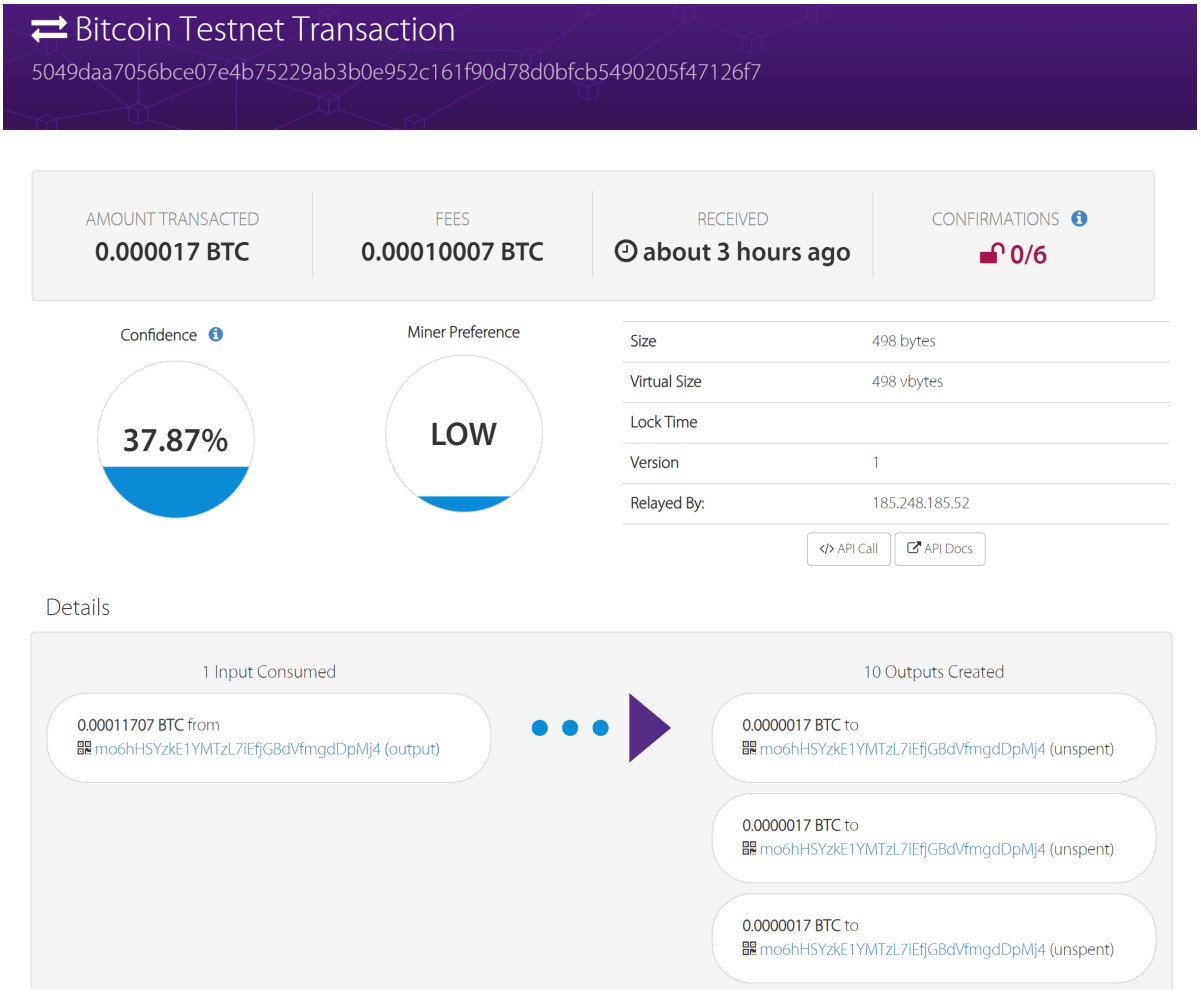
运行 split_test_coins.py 得到输出如下：

```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "5049daa7056bce07e4b75229ab3b0e952c161f90d78d0bfcb5490205f47126f7",
    "addresses": [
```

```
      "mo6hHSYzkE1YMTzL7iEfjGBdVfmgdDpMj4"
    ],
    "total": 1700,
    "fees": 10007,
    "size": 498,
    "vsize": 498,
    "preference": "low",
    "relayed_by": "185.248.185.52",
    "received": "2024-10-05T10:06:56.352988196Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 10,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "f8d9d6583c04a7c83153b98e27e686c8fae9690161b48b8782b66b79d67d74f5",
        "output_index": 0,
        "script":
"48304502210080 6efa1599a32b6eb500903887a6449116775453afe746086f844eb22d6c98ef02202c53d2b
68ab1dd19a0e94ef51da09b21cc8194f09910aab30e5c48e19f4854b401210347143598f3dbf223ef4e49b88
10470ceda09735817e75b69fdabd32e1e6284fa",
        "output_value": 11707,
        "sequence": 4294967295,
        "addresses": [
          "mo6hHSYzkE1YMTzL7iEfjGBdVfmgdDpMj4"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 170,
        "script": "76a914532a7e35436a8ebeca4114753f4ee38433b656ce88ac",
        "addresses": [
          "mo6hHSYzkE1YMTzL7iEfjGBdVfmgdDpMj4"
        ],
        "script_type": "pay-to-pubkey-hash"
      },
      {
        "value": 170,
        "script": "76a914532a7e35436a8ebeca4114753f4ee38433b656ce88ac",
        "addresses": [
          "mo6hHSYzkE1YMTzL7iEfjGBdVfmgdDpMj4"
        ],
        "script_type": "pay-to-pubkey-hash"
      },
      # 共 10 个类似的输出，在此省略，具体输出请见 输出结果.txt
      {
        "value": 170,
        "script": "76a914532a7e35436a8ebeca4114753f4ee38433b656ce88ac",
        "addresses": [
          "mo6hHSYzkE1YMTzL7iEfjGBdVfmgdDpMj4"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```

其中第一行 `201 Created` 说明交易已经成功创建并广播。上述信息包括此次交易的 Hash 值，交易的输出总值 `total`，其单位为 `satoshi`，共 `1700 satoshis`，交易的手续费 `fee`。关键信息还有 `prev_hash`，即作为输入的交易的 Hash 值，在这里就是从 faucet 获取比特币的交易，以及该交易其他相关信息。此次交易共 10 个输出，每个输出的值都是 `170 satoshis`，都指向相同的地址。

通过 Blockcypher 验证交易是否被网络获取：



在 ex1.py 中的几个 TODO 部分，我们需要实现比特币的 P2PKH 交易。比特币的每一笔交易都需要指定花费的比特币的来源，从而验证交易的合法性与防止双花。在作业的情况中，faucet 发起向我发送比特币的交易时，会使用 scriptPubKey 脚本将该比特币锁定，我（接收方）想使用这些比特币时，需要 scriptSig 脚本解锁，确保我是这些比特币的拥有者。P2PKH 交易的 scriptPubKey 脚本和 scriptSig 脚本由比特币脚本语言编写，这种脚本语言包含类似 OP_DUP 这样的操作码。ex1.py 发起向 faucet 发送比特币的交易中，同样需要使用脚本锁定这个比特币，同时需要脚本解锁已有的比特币。

对于 ex1.py 的 TODO 部分，实现如下：

```
def P2PKH_scriptPubKey(address):
    ###############################################################################
    # TODO: Complete the standard scriptPubKey implementation for a
    # PayToPublicKeyHash transaction
    return [OP_DUP, OP_HASH160, address, OP_EQUALVERIFY, OP_CHECKSIG]
    ###############################################################################
```

```
def P2PKH_scriptSig(txin, txout, txin_scriptPubKey):
    signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
                                             my_private_key)
    ################################################################################
    # TODO: Complete this script to unlock the BTC that was sent to you
    # in the PayToPublicKeyHash transaction. You may need to use variables
    # that are globally defined.
    return [signature, my_public_key]
    ################################################################################
```

```
################################################################################
# TODO: set these parameters correctly
amount_to_send = <自由设置>
txid_to_spend = (
    '<上一次交易的 Hash 值>')
utxo_index = <自由设置>
################################################################################
```

验证的过程如下：先将 scriptSig 中的两个值 `[signature, my_public_key]` 放入栈中（公钥位于栈顶），然后执行 scriptPubKey。先 `OP_DUP` 复制栈顶，再 `OP_HASH160` 对复制出来的公钥进行哈希，然后 `address` 入栈，`OP_EQUALVERIFY` 检查栈顶的两个值是否相等，最后 `OP_CHECKSIG` 验证签名有效。

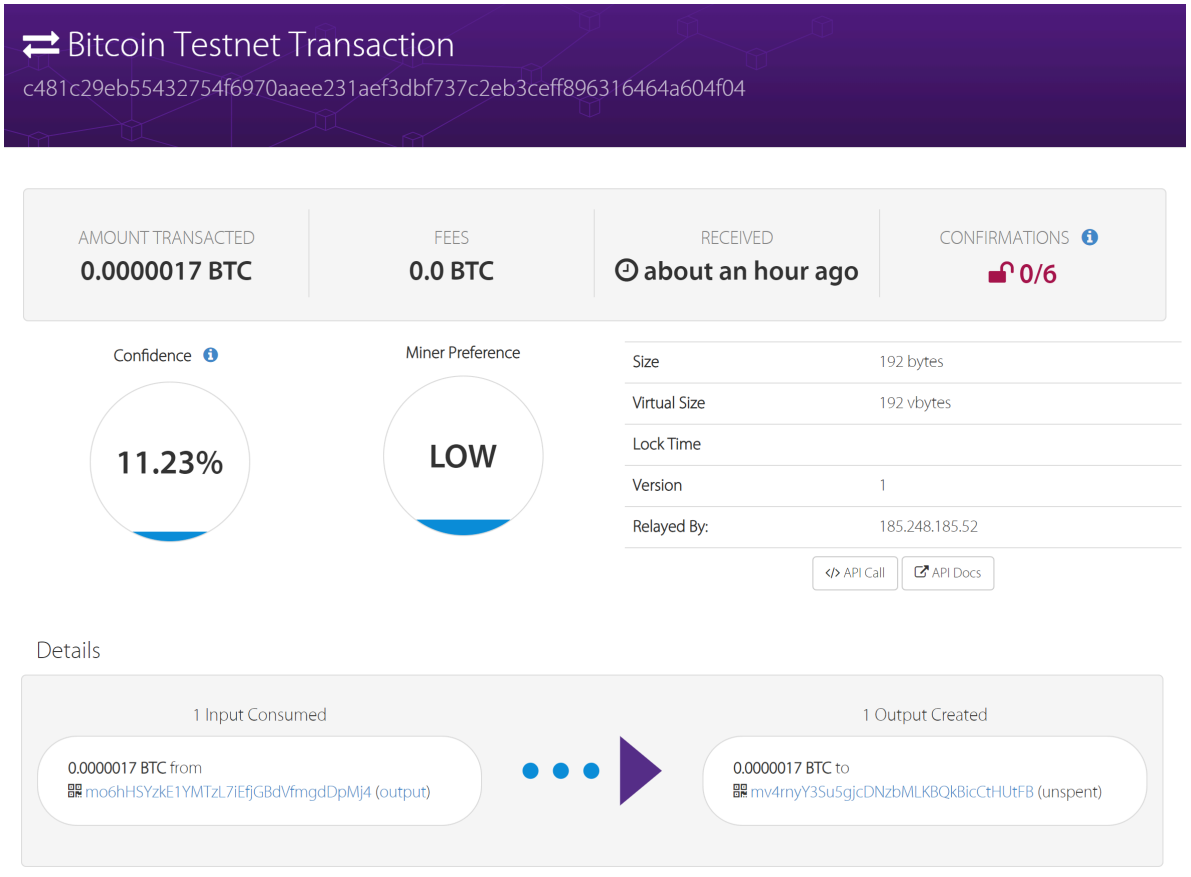
运行 ex1.py 得到结果如下：


```
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "c481c29eb55432754f6970aaee231aef3dbf737c2eb3ceff896316464a604f04",
    "addresses": [
      "mo6hHSYzkE1YMTzL7iEfjGBdVfmgdDpMj4",
      "mv4rnyY3Su5gjcDNzbMLKBQkBicCtHUtFB"
    ],
    "total": 170,
    "fees": 0,
    "size": 192,
    "vsize": 192,
    "preference": "low",
    "relayed_by": "185.248.185.52",
    "received": "2024-10-05T12:22:11.600207063Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "5049daa7056bce07e4b75229ab3b0e952c161f90d78d0bfcb5490205f47126f7",
        "output_index": 0,
        "script":
"48304502210Ob108ed2294a74d63b08a8cfd9c4dc84cf0237b2daace5b36b60ea2aa5c62807802201c5863d
488aa255d1d28d839794d18573059c396bb1e5f2e61497f09a43859ca01210347143598f3dbf223ef4e49b88
10470ceda09735817e75b69fdabd32e1e6284fa",
        "output_value": 170,
        "sequence": 4294967295,
        "addresses": [
          "mo6hHSYzkE1YMTzL7iEfjGBdVfmgdDpMj4"
```

```
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 170,
        "script": "76a9149f9a7abd600c0caa03983a77c8c3df8e062cb2fa88ac",
        "addresses": [
          "mv4rnyY3Su5gjcDNzbMLKBQkBicCtHUtFB"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```


交易已被成功生成，输出信息同样包含了该交易的 Hash，以及比特币交易相关的输入和输出信息，以及脚本类型 `pay-to-pubkey-hash`。

在 Blockcypher 中验证以上交易被网络获取：



如果不修改以上信息再次运行 ex1.py，则会导致双花，这时 ex1.py 的运行结果为


```
409 Conflict
{}
```


出现错误，说明之前的交易已经被网络接收，无法两次花费同一个 UTXO。