CryptoCurrencies Project#1

Alireza Arbabi

810199334

Section 1) Bitcoin Address Generator

Q1&2

In the AddressGenerator.py class, there is a class called ***PublicAddressGenerator***, which can be used to create bitcoin Address. By putting **False** in the argument of the class constructor, an address will be generated, and it will be printed with the corresponding private and public key(the x & y points on the secp256k1 curve). By putting **True** in the argument of the class constructor, the program asks for a 3 character vanity string, and then the program tries to create the corresponding vanity Address.

Q1- Simple Address:

```
alireza@alireza:/media/alireza/C0BEE86EBEE85F02/UT/Lessons/Term6/Crypto/Projects/P1$ python3 AddressGenerator1.py
private key: 1ddda3818859bbbf9c6d273665a06b7f36b43cae582315ea80d9fcd8247f9f51
WIF Address: 4izpTr4weKWFcQatnK7riAHYTh6zSD9nZVH8Kp3nz7oHSyu74aT
public key(x , y): (57605217748980711082184506751514553415958811820405834077763146040340619445741, 78344610854722498518420633679590242264371415693973910019463670048845601097566)
Address: mqXPrsQfx42xMd5trUgrqqNlu866Ai93FB
alireza@alireza:/media/alireza/C0BEE86EBEE85F02/UT/Lessons/Term6/Crypto/Projects/P1$
```

Q2- Vanity Address:

```
alireza@alireza:/media/alireza/C0BEE86EBEE85F02/UT/Lessons/Term6/Crypto/Projects/P1$ python3 AddressGenerator1.py
Enter your vanity string (3 char): zqi
private key: 09dc50e8c3e910c3d3824779729da2707fd5dd183a40fa42ecf120724f9a3728
WIF Address: 4j5hqo3DhtbhBuMk8Pw4S6TJ1Fq3oKaFPnAqTomykcekazQYPu1
public key(x , y): (36899279086099797980690573164977193859101170017289608896988711635144476784149, 83580453543266980018482490919872697019834995474985656276112144869339029245768)
Address: mzqiLEnPJbt45tRKEUDcRcrn61D1yvN2YP
alireza@alireza:/media/alireza/C0BEE86EBEE85F02/UT/Lessons/Term6/Crypto/Projects/P1$
```

Section 2) Creating Transaction

Q1-P1)

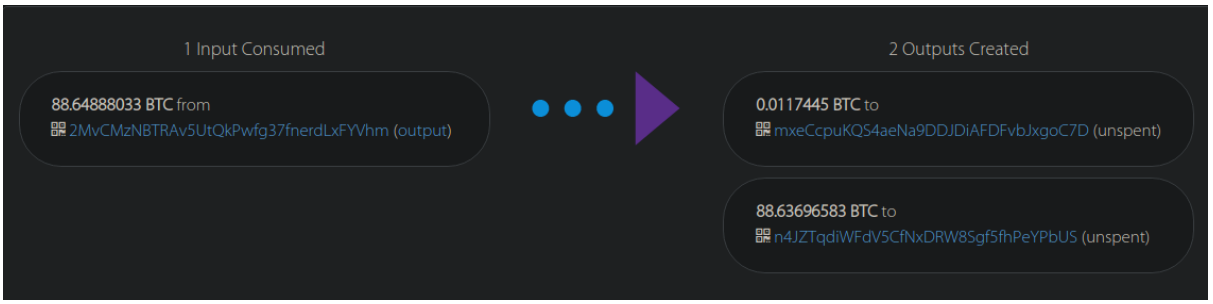
Our main address data:

```
private_key: 931cPqJ5igurei6mQSQYfte9buw3KGF6VERz9hDekukF5Dcwf23
address: mxeCcpuKQS4aeNa9DDJDiAFDFvbJxgoC7D
```

Our second address data:

```
private_key: 93S78wP3T666D8wLtccQ79bsys631vSR64TMLKVishP6wG9m8Yp
address: myonnzBhpPdX6q5JymPgd3jrkW2hbQmF4B
```

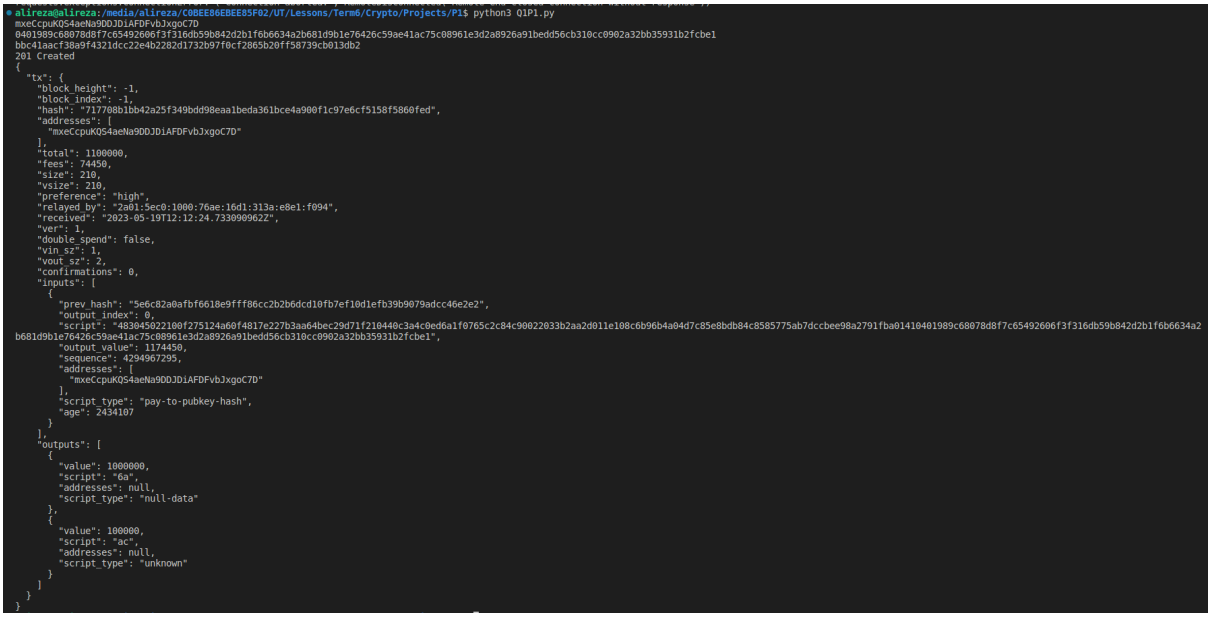
Now here we get some coins from coinfaucet.eu to the main user address:



Here is the Link to the transaction page. After receiving bitcoin from the faucet, we were asked to create a transaction with 2 outputs, an unspendable output , and an output which can be spent by everyone. We can create the first output by only putting OP_RETURN script in the public

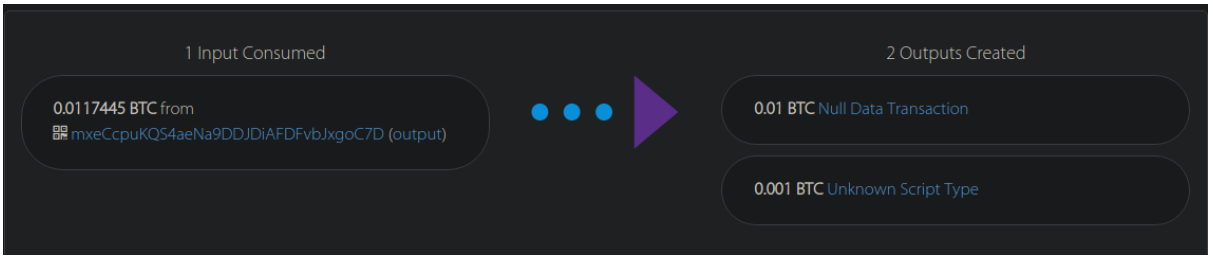
script of the first output, and we can create the second output by putting only OP_CHECKSIG script in the public script of the second output.

We can execute this transaction by running Q1P1.py:

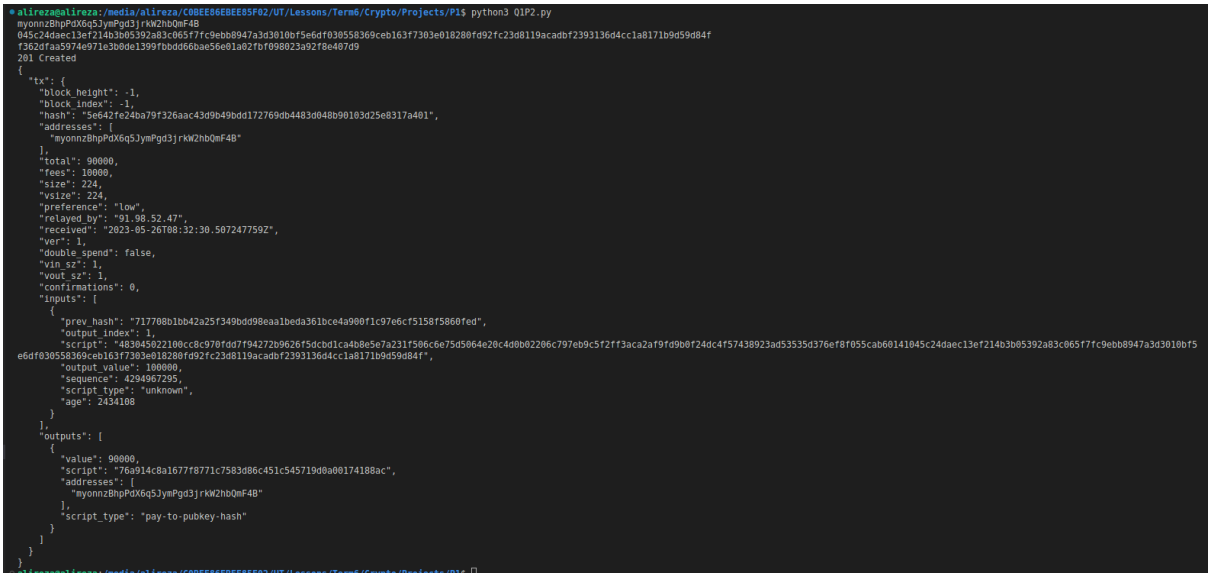


So here is the new transaction with the above features: (Link)

Transaction Hash: 717708b1bb42a25f349bdd98eaa1beda361bce4a900f1c97e6cf5158f5860fed

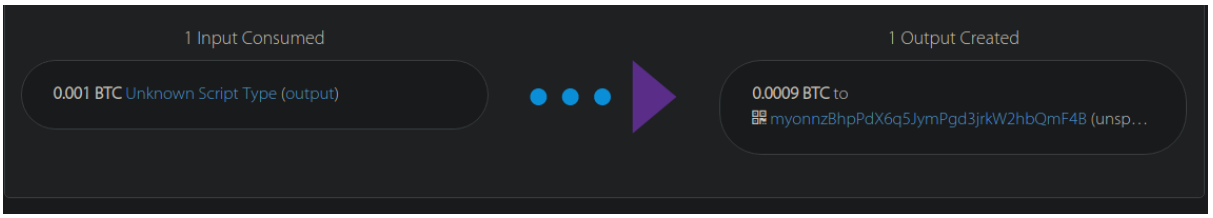


As described before, the second output of the latest transaction can be used by everyone. So we send it back to our another address (address 2) by creating another transaction. This is the output of the Q1P2.py file:



Here is the overview of the transaction above: (Link)

Transaction hash: 5e642fe24ba79f326aac43d9b49bdd172769db4483d048b90103d25e8317a401

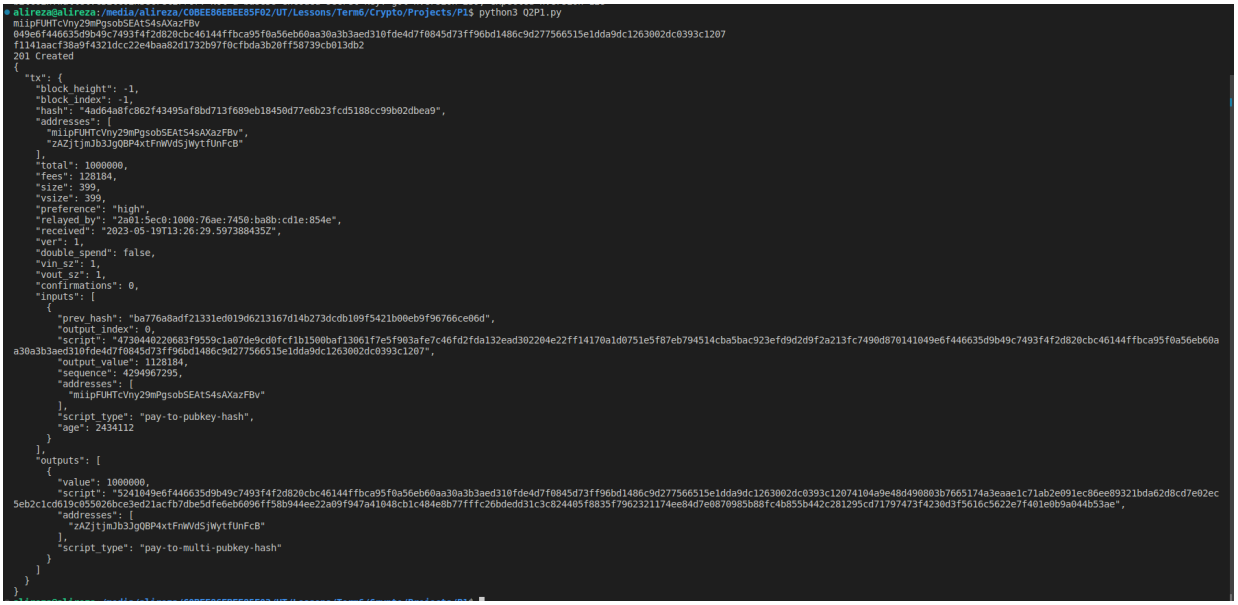


Q1-P2)

3 new generated addresses:

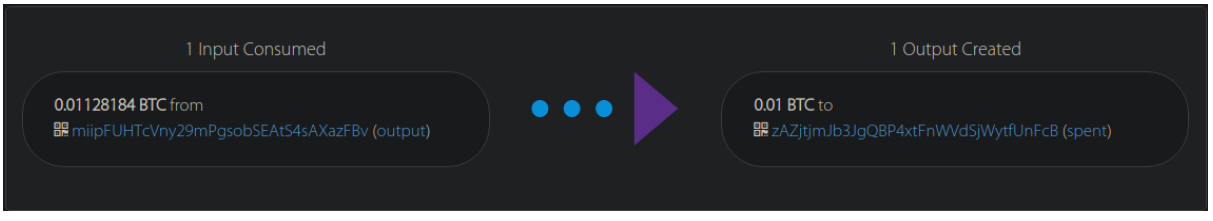
```
private_key 0 : 93R6C1uspBV9RFtNXjc1oEFZXA494HMjZBNUpKueZhPx2iFVAWG
address: miipFUHTcVny29mPgsobSEAtS4sAXazFBv
private_key 1 : 92h8FzaffymGV8JZgYtTwNmeK2QNVXvKEms7yVtDUoyRtHpG2YN
address: mjLRVq15q6Qwpn4fQyzSLsjzYCq6aLRmjs
private_key 2 : 91rEuvZai6xBjbSAX4qdL4tWBj8ANktypVhKkjwQciUfB71noZx
address: mhPavQtru2CbGb3acy14ArLC8w95MRBSvw
```

First we should send some coins to an address, so we receive 0.11 from a faucet to the "miipFUHTcVny29mPgsobSEAtS4sAXazFBv" address. Now we have to create a 2-of-3 P2MS transaction. We can do it by executing the Q2P1.py file:



Link of the transaction below: Link

Transaction Hash: 4ad64a8fc862f43495af8bd713f689eb18450d77e6b23fcd5188cc99b02dbea9

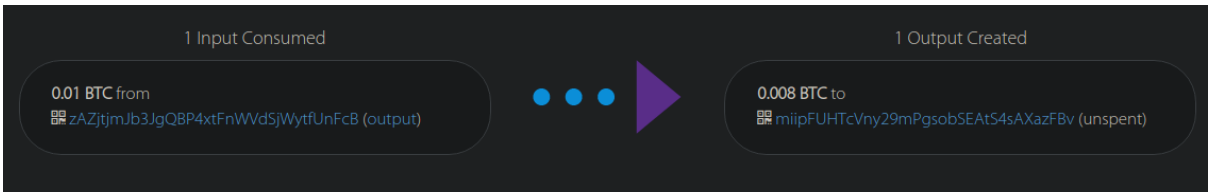


Now we have to spend that money using 2 of 3 users whose public keys were put in the transaction above. We can do this by running Q2P2.py file as it shown below:

```
alireza@alireza:/media/alireza/C0BEE86EBEE85F02/UT/Lessons/Term6/Crypto/Projects/P1$ python3 Q2P2.py
miipFUHTcVny29mPgsobSEAtS4sAXazFBv
049e6f446635d9b49c7493f4f2d820cbc46144ffbca95f0a56eb60aa30a3b3aed310fde4d7f0845d73ff96bd1486c9d277566515e1dda9dc1263002dc0393c1207
f1141aacf38a9f4321dcc22e4baa82d1732b97f0cfbda3b20ff58739cb013db2
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "23b1a612d5ab9a4c4f229aee0c4dd559f9bd92345b330d049a94dae12d241032",
    "addresses": [
      "miipFUHTcVny29mPgsobSEAtS4sAXazFBv",
      "zAZjtjmJb3JgQBP4xtFnWVdSjWytfUnFcB"
    ],
    "total": 800000,
    "fees": 200000,
    "size": 231,
    "vsize": 231,
    "preference": "high",
    "relayed_by": "91.98.59.121",
    "received": "2023-05-19T14:34:25.641390981Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "4ad64a8fc862f43495af8bd713f689eb18450d77e6b23fcd5188cc99b02dbea9",
        "output_index": 0,
        "script": "0048304502210 0b67ddacf5402f51b5abf354ea49556cb3d65267f762b00f7b8c70be7f35c129a02201bc3e921d633a66255b833dab7c48f02848ef6b1ba46526920695114e955700601473044022044e879b0064323a9982786520b40aae1a596140bedd
fc6b5fd2827aaaa1d61e90220147beef5a237ecf6fc671de4db52f29956179949f4b391d2894c1430d28e16fb01",
        "output_value": 1000000,
        "sequence": 4294967295,
        "addresses": [
          "zAZjtjmJb3JgQBP4xtFnWVdSjWytfUnFcB"
        ],
        "script_type": "pay-to-multi-pubkey-hash",
        "age": 2434113
      }
    ],
    "outputs": [
      {
        "value": 800000,
        "script": "76a9142326a756092cbb75f42822417207372d1e390f6e88ac",
        "addresses": [
          "miipFUHTcVny29mPgsobSEAtS4sAXazFBv"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```


Here is the link of the transaction above, As it shown below:

Transaction hash: 23b1a612d5ab9a4c4f229aee0c4dd559f9bd92345b330d049a94dae12d241032



Q1-P3)

Private key: 92h8FzaffymGV8JZgYtTwNmeK2QNVXvKEms7yVtDUoyRtHpG2YN

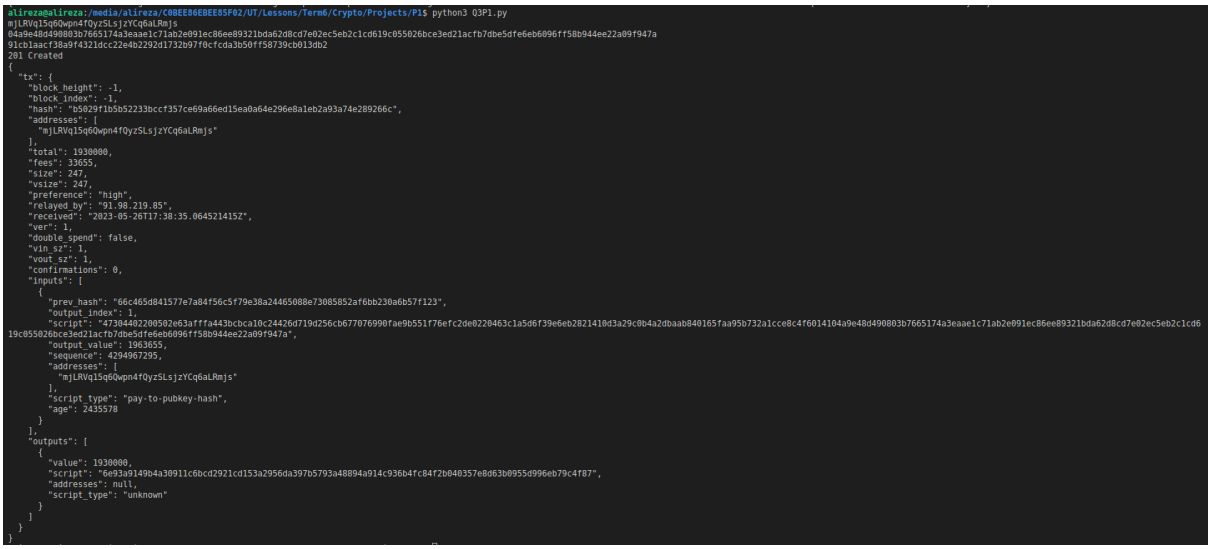
Prime number 1: 89

Prime number 2: 13

The locking script to achieve what the question wants is as follows:

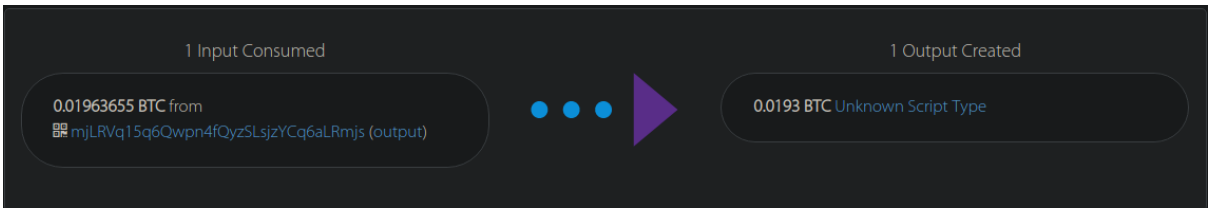
```
def customized_locking_script(sum_nums:bytes, sub_nums:bytes):
    return [OP_2DUP, OP_ADD, OP_HASH160, Hash160(sum_nums), OP_EQUALVERIFY, OP_SUB, OP_HASH160, Hash160(sub_nums), OP_EQUAL]
    # return [OP_2DUP, OP_ADD, sum_nums, OP_EQUALVERIFY, OP_SUB, sub_nums, OP_EQUAL]

def customized_unlocking_script(prime_num1:bytes, prime_num2:bytes):
    return [prime_num1, prime_num2]
```

We can run the questioned script by running the file Q3P1.py, and the output is as follows:



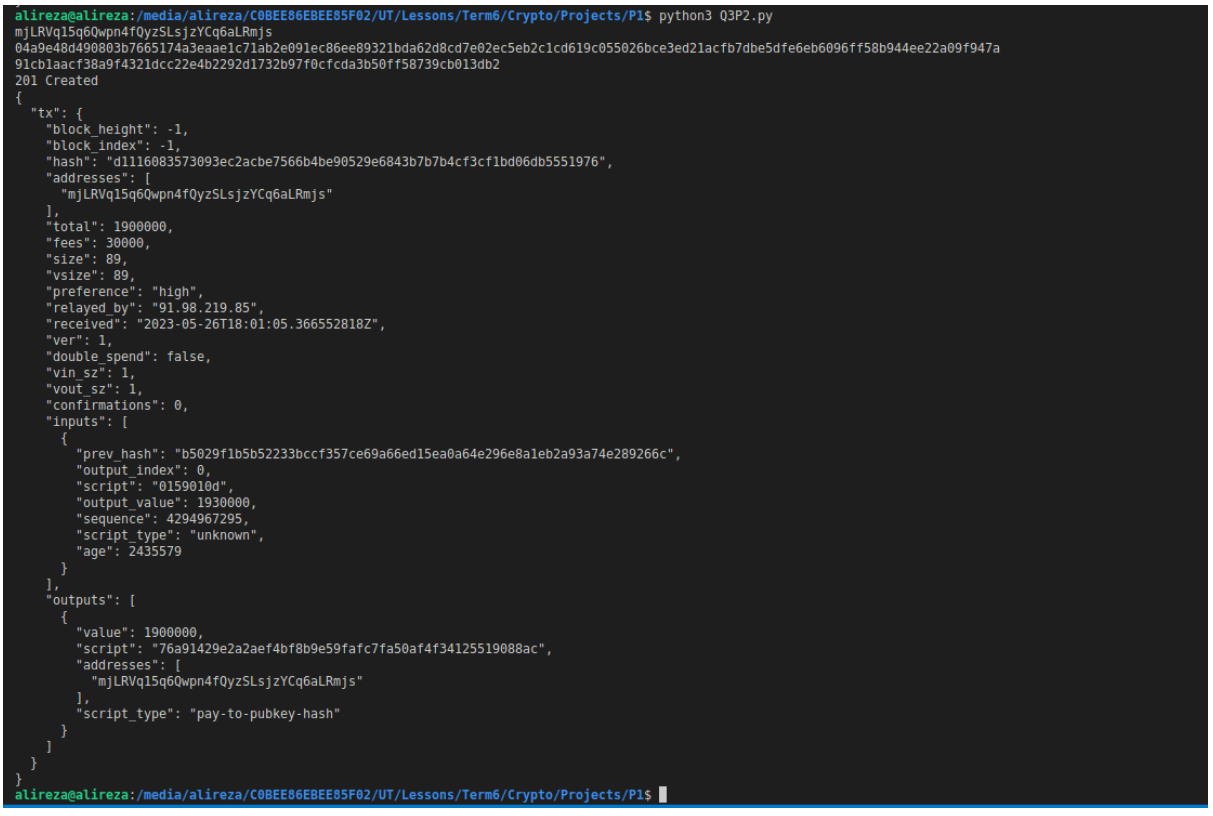
Here is the link of the transaction Above, As it shown below:

Transaction hash: b5029f1b5b52233bccf357ce69a66ed15ea0a64e296e8a1eb2a93a74e289266c

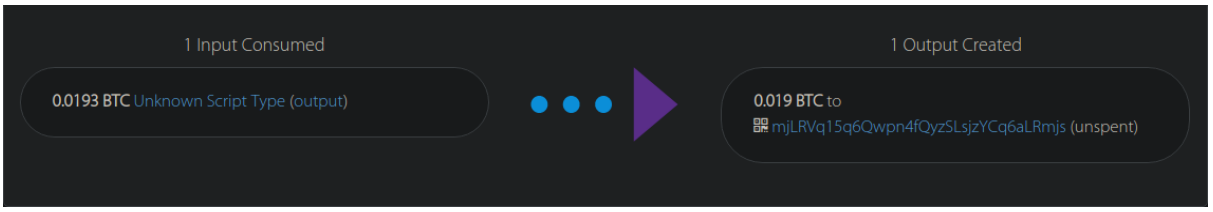


Now we have to spend this transaction by using those 2 prime numbers which was defined before, By running Q3P2.py file we have this transaction with hash:

d1116083573093ec2acbe7566b4be90529e6843b7b7b4cf3cf1bd06db5551976



Here is the schematic of the corresponding transaction:



Section 3) Mining a block

Step1) Find my block information:

My block num: 9334

My block hash:

00000000261b4765edf334510ef4e167cbaf58406af6281891b4d57595c10fc9

Step2) create coinbase transaction:

Transaction_input = 32 bytes 0

Transaction_input_index = 0xFFFFFFFF

Coinbase_Data = 810199334AlirezaArbabi (ASCII to hex)

Output script = Our address to receive block reward

Step3) Calculate Merkle Root. As we have only 1 transaction, the merkle root will be the SHA256 hash of the coinbase transaction.

Step4) Mine desired block. Now we have to put prev_block_hash, merkle_root, and nonce into the block header. We have to find the nonce in a way that the hash of the block header becomes lower than $2^{(256-16)}$.

By running the BlockMiner.py code, we have:

```
alireza@alireza:/media/alireza/C0BEE86EBEE85F02/UT/Lessons/Term6/Crypto/Projects/P1$ python3 BlockMiner.py
Nonce found: 8041
Previous Block Number:  9334
Previous Block Hash:  00000000261b4765edf334510ef4e167cbaf58406af6281891b4d57595c10fc9
Mining reward: 3.125
Coinbase data:  810199334AlirezaArbabi
Coinbase hex data:  383130313939333334416c6972657a61417262616269
transaction_out:
{
   Input: [CTxIn(COutPoint(), CScript([x('383130313939333334416c6972657a61417262616269')]), 0xffffffff)]
   Output: [CTxOut(3.125*COIN, CScript([OP_DUP, OP_HASH160, x('09accaa873a364f6921ffb9002e4e8a3c9eb9671'), OP_EQUALVERIFY, OP_CHECKSIG]))]
}
Block hash: 000069d553cb21798f9743a66bd8f9d372435bb88fc1adbfaadd315994654469
Block header: 01000000c90fc19575d5b4911828f66a4058afcb67e1f40e5134f3ed65471b260000000007ab2f95fd721a88756e8655dc68021de25997c158d46570c6d560
d4e70668eb08df706400000000000000000000000000000000000000000000000000000000000000000000000000000000000000010000000000000000000000000000000000000000000000000000000000000000
000000691f0000
Block Body in hex: 01000000c90fc19575d5b4911828f66a4058afcb67e1f40e5134f3ed65471b260000000007ab2f95fd721a88756e8655dc68021de25997c158d46570c
6d560d4e70668eb08df706400000000000000000000000000000000000000000000000000000000000000000000000000000000000000100000000000000000000000000000000000000000000000000000000000000000
00000000000691f000001000000010000000000000000000000000000000000000000000000000000000000000000000000000000000ffffffff171638313031393933334416c6972657a61417
262616269ffffffff01205fa012000000001976a91409accaa873a364f6921ffb9002e4e8a3c9eb967188ac00000000
alireza@alireza:/media/alireza/C0BEE86EBEE85F02/UT/Lessons/Term6/Crypto/Projects/P1$
```