پروژه کامپیوتری اول رمزارز

آوا میرمحمدمهدی 810199501

بخش اول: تولید آدرس

قسمت اول:

در ابتدا کلیدهای خصوصی و عمومی را تولید می‌کنیم؛ برای تولید کلید خصوصی از کتابخانه‌ی secrets استفاده شد و 32 بایت (256 بیت) عدد رندوم تولید شد و سپس با استفاده از کتابخانه ecdsa (elliptic curve digital signature algorithm) و با استفاده از خم SECP256k1 یک کلید تولید شد و بعد از اینکه آن را به استرینگ تبدیل کردیم، به ابتدای آن b"\x04" را اضافه کردیم تا کلید عمومی بدست آید.

```
def generate_keys():
    private_key = secrets.token_bytes(32)
    key = ecdsa.SigningKey.from_string(private_key, curve=ecdsa.SECP256k1).verifying_key
    public_key = b"\x04" + key.to_string()
    return private_key, public_key
```

برای تولید کلید خصوصی به فرمت WIF، چون در شبکه تست کار می‌کنیم b"\xef" را به ابتدای کلید عمومی اضافه می‌کنیم؛ درصورتی که شبکه اصلی بود باید به جای آن، b"\x80" را به ابتدای آن اضافه می‌کردیم؛ درصورتی که از فرمت فشرده استفاده شود باید b"\x01" را به آن اضافه کنیم که در اینجا از فرمت فشرده استفاده نمی‌شود؛ حالا دو سری به صورت پی در پی هش رشته‌ی بدست آمده را با استفاده از sha256 پیدا می‌کنیم و آن را به مقدار بدست آمده در مرحله‌ی قبل اضافه می‌کنیم و در آخر base58 آن را محاسبه می‌کنیم تا به فرمت WIF تبدیل شود.

```
def get_checksum(payload):
    first_hash = hashlib.sha256(payload).digest()
    second_hash = hashlib.sha256(first_hash).digest()
    return second_hash[:4]

def generate_WIF_private_key(private_key, use_compressed = False):
    extended_key = b"\xef" + private_key
    if (use_compressed):
        extended_key += b"\x01"
    checksum_of_extended = get_checksum(extended_key)
    extended_key += checksum_of_extended
    wif_private_key = base58.b58encode(extended_key).decode('utf-8')
    return wif_private_key
```

برای تولید آدرس از کلید عمومی استفاده می‌کنیم و ابتدا sha256 و سپس ripemd160 را روی نتیجه‌ی هش قیمت قبل اعمال می‌کنیم و b"\x6f" را به ابتدای آن اضافه می‌کنیم و تابع get_checksum را برای آن فراخوانی می‌کنیم و نتیجه را در انتهای آن اضافه می‌کنیم و در انتها آن را به فرمت base58 تبدیل می‌کنیم. در شبکه اصلی به جای b"\x6f"، b"\x00" را به ایتدای آدرسی که بعد از هش ripemd تولید شده اضافه می‌کنیم و به همین دلیل است که اولین کاراکتر آدرس تولید شده در شبکه اصلی برابر با 1 است ولی اولین کاراکتر تولید شده در شبکه تست برابر با m یا n است.

```
def generate_address(public_key, use_compressed = False):
    sha256_pub = hashlib.sha256(public_key).digest()
    ripemd_pub = hashlib.new('ripemd160')
    ripemd_pub.update(sha256_pub)
    ripemd_pub = ripemd_pub.digest()

    extended_ripemd160 = b"\x6f" + ripemd_pub
    if (use_compressed):
        extended_ripemd160 += b"\x01"
    checksum_of_extended = get_checksum(extended_ripemd160)
    extended_ripemd160 += checksum_of_extended
    bitcoin_address = base58.b58encode(extended_ripemd160).decode('utf-8')
    return bitcoin_address
```

در نهایت کد تولید خروجی و خروجی به صورت زیر خواهد بود:

```
if __name__ == "__main__":
    private_key, public_key = generate_keys()
    print(' private key: ', private_key.hex())
    print(' public key: ', private_key.hex())
    print(' WIF private key: ', generate_WIF_private_key(private_key))
    print(' bitcoin address: ', generate_address(public_key))
```

```
(venv) →  CA python3 Q1_part1.py
 private key:  81eda867aca8922d286e1db27d8c7cf0ce0942a9d6e592eabc4218a85c62fa37
 public key:  81eda867aca8922d286e1db27d8c7cf0ce0942a9d6e592eabc4218a85c62fa37
 WIF private key:  92a91SzvZ53sTQe4cNTz2ySqkypXzE4nmaGLAKsCRemfk2t7uVa
 bitcoin address:  myYcTfx2DBJSa369Jf783aDL4ftyuE5ZX8
```

قسمت دوم:

در این بخش از توابع قسمت قبل برای تولید آدرس استفاده می‌کنیم و با چک کردن حروف دوم تا پنجم آدرس، آنقدر کلید خصوصی و عمومی و آدرس تولید می‌کنیم تا سه حرف آدرس با حروف داده شده منطبق شود.

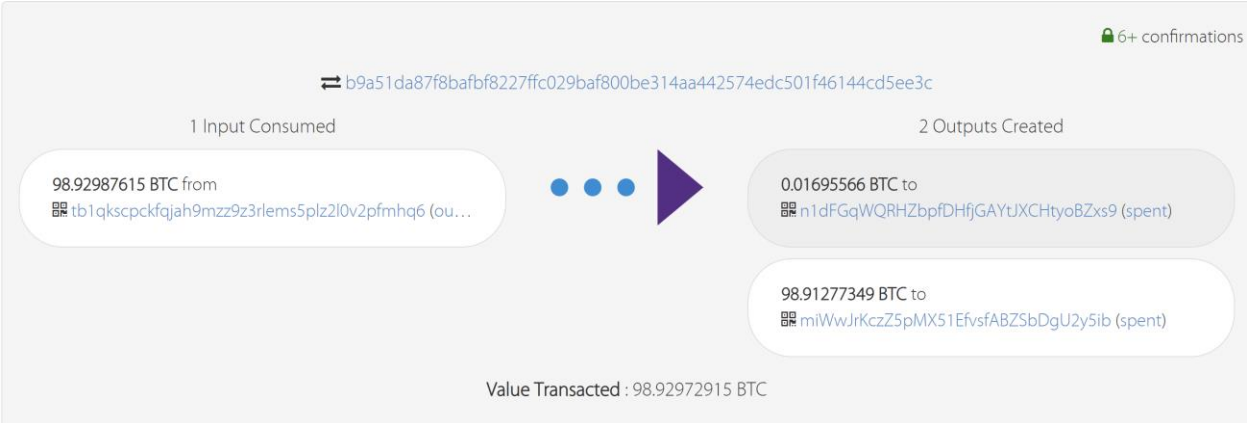
```python
if __name__ == "__main__":
    prefix = input("please enter your prefix: ")

    while True:
        private_key, public_key = generate_keys()
        adr = generate_address(public_key)
        if adr[1:4] == prefix:
            print(' bitcoin address: ', adr)
            print(' WIF private key: ', generate_WIF_private_key(private_key))
            break
```

```
(venv) →  CA python3 Q1_part2.py
please enter your prefix: reb
 bitcoin address:  mrebLETpsFUc1AU19i3YbFN7g4uQPhhKXV
```

بخش دوم: انجام تراکنش

در ابتدا از یکی از private_key هایی که به فرم WIF در بخش قبل ساختیم استفاده می‌کنیم و public_key و address متناظر با آن را پیدا می‌کنیم و سپس از طریق https://coinfaucet.eu/en/btc-testnet با آدرس تولید شده بیت کوین دریافت می‌کنیم. صحت دریافت بیت کوین از این سایت در این لینک قابل مشاهده است. در اینجا مقدار بیت کوین دریافتی برابر با 0.01695566 است.



آدرس من در تمام قسمت‌های این بخش:

n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9

کلید خصوصی من به فرم WIF در تمام قسمت‌های این بخش:

925NvBWuEWdw1Au6CYpgqovtQBhtS3TfjXiLo7NhmoBLMYeeby3

کلید خصوصی من بعد از اعمال تابع bitcoin.wallet.CBitcoinSecret():

409ff4d2516c040a7dd49806ad84a44b81133f08cacf16e59d2a3e4e272de769

```
bitcoin.SelectParams("testnet")
pv_key = "925NvBWuEWdw1Au6CYpgqovtQBhtS3TfjXiLo7NhmoBLMYeeby3"
my_private_key = bitcoin.wallet.CBitcoinSecret(pv_key)
my_public_key = my_private_key.pub
my_address = bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)
```

در تابع P2PKH_scriptPubKey ابتدا چیزی که در سر استک قرار دارد کپی می‌شود (که در اینجا my_public_key است) و سپس از آن HASH_160 گرفته می‌شود و در استک قرار می‌گیرد و سپس Hash160(my_public_key) روی استک قرار می‌گیرد و مقدار آن با هش قبلی گرفغته شده مقایسه می‌شود و درصورتی که برابر بودند ادامه‌ی استک بررسی می‌شود و در غیر اینصورت false برگردانده می‌شود؛ در ادامه OP_CHECKSIG قرار دارد که امضای ما که در ته استک قرار دارد را با توجه به my_public_key مقایسه می‌کند و درصورتی که همخوانی داشتند true برمی‌گرداند و درغیر اینصورت false.

در تابع P2P_scriptSig که مقدار بازکشتی آن قبل از مقدار بازکشتی تابع P2PKH_scriptPubKey روی استک قرار می‌گیرد، با استفاده از private_key خودمان transaction ایجاد شده را امضا می‌کنیم و در نهایت صحت اینکه ما آن transaction را انجام داده‌ایم با کلید عمومی ما چک می‌شود.

1_1) تولید دو خروجی، یکی قابل خرج برای همه و دیگری غیرقابل خرج برای همه:

```
def P2PKH_scriptPubKey(key):
    return [OP_DUP, OP_HASH160, Hash160(key), OP_EQUALVERIFY, OP_CHECKSIG]

def P2PKH_scriptSig(txin, txout, txin_scriptPubKey):
    signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey, my_private_key)
    return [signature, my_public_key]

def get_txout_scriptPubKeys(amount_to_send1, amount_to_send2):
    txout1_scriptPubKey = [OP_TRUE]
    txout2_scriptPubKey = [OP_FALSE]
    txout1 = create_txout(amount_to_send1, txout1_scriptPubKey)
    txout2 = create_txout(amount_to_send2, txout2_scriptPubKey)
    return txout1, txout2

def send_from_P2PKH_transaction(amount_to_send1, amount_to_send2, txid_to_spend, utxo_index):
    txout1, txout2 = get_txout_scriptPubKeys(amount_to_send1, amount_to_send2)
    txin_scriptPubKey = P2PKH_scriptPubKey(my_public_key)
    txin = create_txin(txid_to_spend, utxo_index)
    txin_scriptSig = P2PKH_scriptSig(txin, [txout1, txout2], txin_scriptPubKey)
    new_tx = create_signed_transaction(txin, [txout1, txout2], txin_scriptPubKey, txin_scriptSig)
    return broadcast_transaction(new_tx)


if __name__ == '__main__':
    all_money = 0.01695566
    amount_to_send1 = 0.0168
    amount_to_send2 = 0.00000001
    txid_to_spend = ('b9a51da87f8bafbf8227ffc029baf800be314aa442574edc501f46144cd5ee3c')
    utxo_index = 0
    print("my address: ", my_address)
    print("my public key: ", my_public_key.hex())
    print("my private key: ", my_private_key.hex())
    response = send_from_P2PKH_transaction(amount_to_send1, amount_to_send2, txid_to_spend, utxo_index)
    print("response status code: ", response.status_code)
    print("reponse reason: ", response.reason)
    print("response text: ", response.text)
```

در این قسمت txid_to_spend را برابر با آیدی تراکنشی که توسط آن بیت کوین دریافت کردیم می‌گذاریم و ایندکس نیز برابر با 0 است چون در ایندکس 0 آن بیت کوین به حساب ما آمده و در ایندکس 1 آن بقیه‌ی بیت کوین‌ها به آدرس مالک آن بازگردانده شده است. دو مقدار تعریف می‌کنیم به طوری که مجموع آنها از مقدار موج.ودی ما کمتر باشد و مقداری برای transaction fee باقی بماند تا transaction ما توسط دیگران confirm شود؛ برای اینکه مقدار اول توسط همه قابل برداشت باشد، مقدار txout1_scriptPubKey را برابر با [OP_TRUE] قرار می‌دهیم تا سر استک همواره true باقی‌بماند و درنتیجه مقدار بیت کوین توسط همه قابل برداشت باشد، مقدار txout2_scriptPubKey را نیز برابر با [OP_FALSE] قرار می‌دهیم تا توسط هیچ‌کس قابل برداشت نباشد و سر استک همواره false باقی بماند.

مقدار transaction fee در این قسمت برابر با (0.0168 + 0.0000001) – 0.01695566 است.

هش این transaction برابر است با:

44f28dcfed5759766ff76603585e80cd7fe121aabae9c45ec7f2303e484d6892

و صحت آن در این لینک قابل بررسی است.

2_1) خرج کردن خروجی قابل خرج و بازگشت آن به آدرس خودمان:

برای بازگشت قسمت از پول که برای همه قابل خرج است به خودمان، کد زیر تولید شد:

```python
def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = P2PKH_scriptPubKey(my_public_key)
    txout = create_txout(amount_to_send, txout_scriptPubKey)
    txin_scriptPubKey = [OP_TRUE]
    txin = create_txin(txid_to_spend, utxo_index)
    txin_scriptSig = []

    new_tx = create_signed_transaction(txin, [txout], txin_scriptPubKey, txin_scriptSig)

    return broadcast_transaction(new_tx)


if __name__ == '__main__':
    all_money = 0.0168
    amount_to_send = 0.0166
    txid_to_spend = ('44f28dcfed5759766ff76603585e80cd7fe121aabae9c45ec7f2303e484d6892')
    utxo_index = 0

    print("my address: ", my_address)
    print("my public key: ", my_public_key.hex())
    print("my private key: ", my_private_key.hex())
    response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index)
    print("response status code: ", response.status_code)
    print("reponse reason: ", response.reason)
    print("response text: ", response.text)
```

در اینجا مقدار 0.0002 بیت کوین برای transaction fee در نظر گرفتیم و txin_subscriptPubKey را برابر با [OP_TRUE] قرار دادیم زیر ورودی ما برای همه قابل خرج است و txin_scriptSig را برابر با [] در نظر گرفتیم چون زیر استک است و روی استک TRUE است پس زیر آن اهمیتی برای تشخیص صحت ندارد. و همچنین برای بازکشت بیت کوین از P2PKH استفاده شد و txout_scriptPubKey را برابر با مقدار بازگشتی تابع P2PKH_scriptPKey که در نحوه‌ی کارکرد آن در قبل توضیح داده شد قرار دادیم.

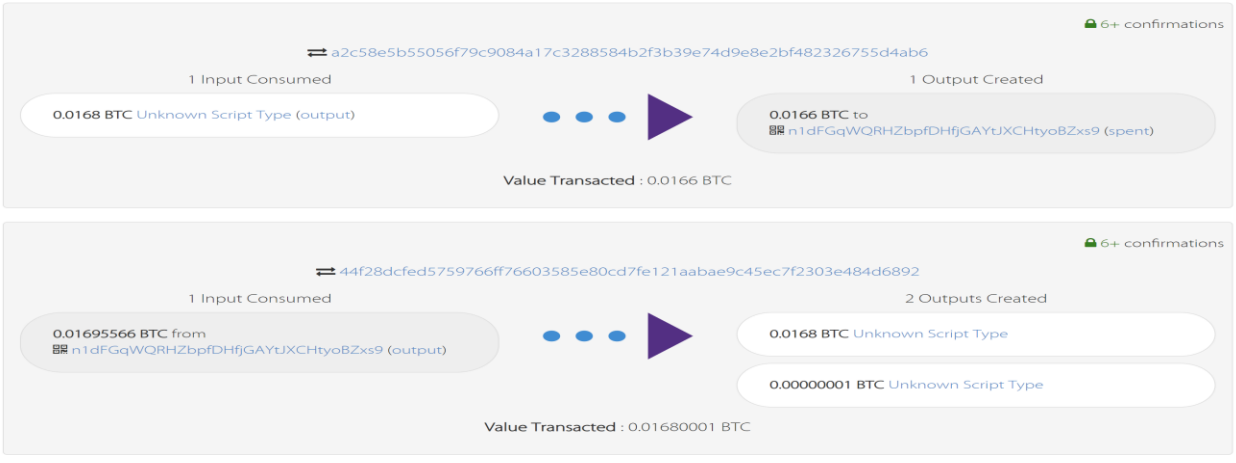
```
(venv) → CA python3 Q2_part1_2.py
my address:  n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9
my public key:  0477ce41e46d710025cb7ae0bc425fbd59d2c1711706c8772d7616e2d073afd245f03c803dfea52c572b49e6ac0d42134c40c60219fc3650c89f7c0e0a678be26e
my private key:  409ff4d2516c040a7dd49806ad84a44b81133f08cacf16e59d2a3e4e272de769
response status code:  201
reponse reason:  Created
response text:  {
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "a2c58e5b55056f79c9084a17c3288584b2f3b39e74d9e8e2bf482326755d4ab6",
    "addresses": [
      "n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9"
    ],
    "total": 1660000,
    "fees": 20000,
    "size": 85,
    "vsize": 85,
    "preference": "high",
    "relayed_by": "5.114.128.12",
    "received": "2023-05-24T06:37:23.854648013Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "44f28dcfed5759766ff76603585e80cd7fe121aabae9c45ec7f2303e484d6892",
        "output_index": 0,
        "output_value": 1680000,
        "sequence": 4294967295,
        "script_type": "unknown",
        "age": 2435049
      }
    ],
    "outputs": [
      {
        "value": 1660000,
        "script": "76a914dc932f50397a45847c79fb3d62536dac7ad75edd88ac",
        "addresses": [
          "n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```

هش این transaction برابر است با:

a2c58e5b55056f79c9084a17c3288584b2f3b39e74d9e8e2bf482326755d4ab6

و صحت آن در این لینک قابل بررسی است.

دو تراکنش بخش 1_1 و 2_1:



2_1) تولید دو خروجی از نوع Multisig و سه آدرس جدید که توسط دو نفر از آنها قابل خرج باشد:

```
pv_key = "925NvBWuEWdw1Au6CYpgqovtQBhtS3TfjXiLo7NhmoBLMYeeby3"
pv_key1 = "92FzYRi2Kz3DZNub9eeHx6466JjDy31J94WCr1T4DhFwYfZri9W"
pv_key2 = "92GcmLAheuU4gBSRYmqF2eCiJtnBDKLrQH3HCQUqPkEZTUCE9gg"
pv_key3 = "91xg28ARkm4RGVqLehMuwPKVnQ99Lw5qSeK4TLGSX5qFQBWeMgc"
my_private_key = bitcoin.wallet.CBitcoinSecret(pv_key)
private_key1 = bitcoin.wallet.CBitcoinSecret(pv_key1)
private_key2 = bitcoin.wallet.CBitcoinSecret(pv_key2)
private_key3 = bitcoin.wallet.CBitcoinSecret(pv_key3)
my_public_key = my_private_key.pub
public_key1 = private_key1.pub
public_key2 = private_key2.pub
public_key3 = private_key3.pub

my_address = bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

def P2PKH_scriptPubKey(key):
    return [OP_DUP, OP_HASH160, Hash160(key), OP_EQUALVERIFY, OP_CHECKSIG]

def P2PKH_scriptSig(txin, txout, txin_scriptPubKey):
    signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey, my_private_key)
    return [signature, my_public_key]

def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = [OP_2, public_key1, public_key2, public_key3, OP_3, OP_CHECKMULTISIG]
    txout = create_txout(amount_to_send, txout_scriptPubKey)
    txin_scriptPubKey = P2PKH_scriptPubKey(my_public_key)
    txin = create_txin(txid_to_spend, utxo_index)
    txin_scriptSig = P2PKH_scriptSig(txin, [txout], txin_scriptPubKey)
    new_tx = create_signed_transaction(txin, [txout], txin_scriptPubKey, txin_scriptSig)
    return broadcast_transaction(new_tx)


if __name__ == '__main__':
    all_money = 0.0166
    amount_to_send = 0.0164
    txid_to_spend = ('a2c58e5b55056f79c9084a17c3288584b2f3b39e74d9e8e2bf482326755d4ab6')
    utxo_index = 0

    print("my address: ", my_address)
    print("my public key: ", my_public_key.hex())
    print("my private key: ", my_private_key.hex())
    response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index)
    print("response status code: ", response.status_code)
    print("reponse reason: ", response.reason)
    print("response text: ", response.text)
```

در اینجا مقدار txout_scriptPubKey برابر با مقدار زیر در نظر گرفته شده است:

[OP_2, public_key1, public_key2, public_key3, OP_3, OP_CHECKMULTISIG]

ابتدا CHECKMULTISIG از روی استک خوانده می‌شود و OP_3 سه عضو دیگر از روی استک را می‌خواند که در اینجا public_key1 و public_key2 و public_key3 هستند و سپس OP_2 دو عضو دیگر از روی استک می‌خواند (در اینجا این دو عضو امضاهایی هستند که توسط دو کلید خصوصی انجام شده‌اند) و چک می‌کند که دو مورد از امضاها با دو مورد از public_key ها همخوانی داشته باشند (به ترتیب قرارگیری این کار را انجام می‌دهد یعنی امضای1 باید با public_key1 همخوانی داشته باشد و اگر اینطور بود مقدار 1 و در غیر اینصورت مقدار 0 را برمی‌گرداند.

مقدار transaction fee در این تراکنش برابر با 0.0002 بیت کوین در نظر گرفته شد.

هش این transaction برابر است با:

1a9e254ee69d1f9cbf4d50240a546701bf1404b462cc3bfc9009e186eb5760fa

و صحت آن در این لینک قابل بررسی است.

2_2) خرج کردن خروجی و بازگشت آن به آدرس اصلی خودمان:

```
bitcoin.SelectParams("testnet")
pv_key = "925NvBWuEWdw1Au6CYpgqovtQBhtS3TfjXiLo7NhmoBLMYeeby3"
pv_key1 = "92FzYRi2Kz3DZNub9eeHx6466JjDy31J94WCr1T4DhFwYfZri9W"
pv_key2 = "92GcmLAheuU4gBSRYmqF2eCiJtnBDKLrQH3HCQUqPkEZTUCE9gg"
pv_key3 = "91xg28ARkm4RGVqLehMuwPKVnQ99Lw5qSeK4TLGSX5qFQBWeMgc"
my_private_key = bitcoin.wallet.CBitcoinSecret(pv_key)
private_key1 = bitcoin.wallet.CBitcoinSecret(pv_key1)
private_key2 = bitcoin.wallet.CBitcoinSecret(pv_key2)
private_key3 = bitcoin.wallet.CBitcoinSecret(pv_key3)
my_public_key = my_private_key.pub
public_key1 = private_key1.pub
public_key2 = private_key2.pub
public_key3 = private_key3.pub

my_address = bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)
def P2PKH_scriptPubKey(key):
    return [OP_DUP, OP_HASH160, Hash160(key), OP_EQUALVERIFY, OP_CHECKSIG]

def scriptSig(txin, txout, txin_scriptPubKey):
    signature1 = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey, private_key1)
    signature2 = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey, private_key2)
    return [OP_0, signature1, signature2]

def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = P2PKH_scriptPubKey(my_public_key)
    txout = create_txout(amount_to_send, txout_scriptPubKey)
    txin_scriptPubKey = [OP_2, public_key1, public_key2, public_key3, OP_3, OP_CHECKMULTISIG]
    txin = create_txin(txid_to_spend, utxo_index)
    txin_scriptSig = scriptSig(txin, [txout], txin_scriptPubKey)

    new_tx = create_signed_transaction(txin, [txout], txin_scriptPubKey, txin_scriptSig)

    return broadcast_transaction(new_tx)


if __name__ == '__main__':
    all_money = 0.0164
    amount_to_send = 0.016
    txid_to_spend = ('1a9e254ee69d1f9cbf4d50240a546701bf1404b462cc3bfc9009e186eb5760fa')
    utxo_index = 0

    print("my address: ", my_address)
    print("my public key: ", my_public_key.hex())
    print("my private key: ", my_private_key.hex())
    response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index)
    print("response status code: ", response.status_code)
    print("reponse reason: ", response.reason)
    print("response text: ", response.text)
```

در این قسمت تابع P2PKH_scriptSig را به شکل بالا تغییر می‌دهیم؛ دو امضا توسط private_key1 و private_key2 می‌سازیم، یعنی transaction را یک بار با کلید خصوصی اول و بار دیگر با کلید خصوصی دوم امضا می‌کنیم و txin_scriptSig را برابر با مقدار [OP_0, signature1, signature2] قرار می‌دهیم، اینکار برای این است که همانطور که در قسمت قبل توضیح دادیم این دو امضا با دو public_key چک می‌شوند و درصورتی که همخوانی داشته باشند، بیت کوین قابل خرج کردن می‌شود و OP_0 به این دلیل در پایین استک قرار دارد که OP_CHECKMULTISIG یک عضو اضافه از استک pop می‌کند و نیاز است که در انتهای استک چیزی وجود داشته باشد در اینجا OP_0 را قرار دادیم ولی تفاوتی ندارد.

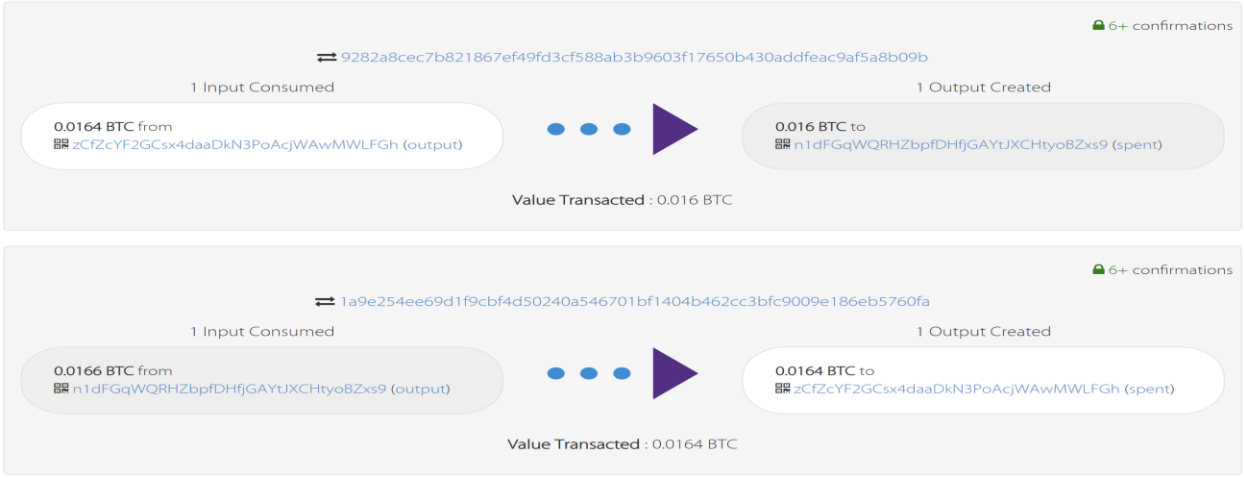
مقدار transaction fee در این تراکنش برابر با 0.0004 بیت کوین در نظر گرفته شد.

هش این transaction برابر است با:

9282a8cec7b821867ef49fd3cf588ab3b9603f17650b430addfeac9af5a8b09b

و صحت آن در این لینک قابل بررسی است.

دو تراکنش بخش 1_2 و 2_2:



3_1) تولید خروجی‌ای که تنها با داشتن دو عدد اول قابل خرج باشد:

```
PRIME_NUM1 = 1021
PRIME_NUM2 = 1019

pv_key = "925NvBWuEWdw1Au6CYpgqovtQBhtS3TfjXiLo7NhmoBLMYeeby3"
my_private_key = bitcoin.wallet.CBitcoinSecret(pv_key)
my_public_key = my_private_key.pub

my_address = bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

def P2PKH_scriptPubKey(key):
    return [OP_DUP, OP_HASH160, Hash160(key), OP_EQUALVERIFY, OP_CHECKSIG]

def P2PKH_scriptSig(txin, txout, txin_scriptPubKey):
    signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey, my_private_key)
    return [signature, my_public_key]

def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = [OP_2DUP, OP_SUB, OP_HASH160, Hash160((PRIME_NUM1 - PRIME_NUM2).to_bytes(1, byteorder="little")), OP_EQUALVERIFY,
                          OP_ADD, OP_HASH160, Hash160((PRIME_NUM1 + PRIME_NUM2).to_bytes(2, byteorder="little")), OP_EQUAL]
    txout = create_txout(amount_to_send, txout_scriptPubKey)
    txin_scriptPubKey = P2PKH_scriptPubKey(my_public_key)
    txin = create_txin(txid_to_spend, utxo_index)
    txin_scriptSig = P2PKH_scriptSig(txin, [txout], txin_scriptPubKey)

    new_tx = create_signed_transaction(txin, [txout], txin_scriptPubKey, txin_scriptSig)

    return broadcast_transaction(new_tx)


if __name__ == '__main__':
    all_money = 0.016
    amount_to_send = 0.011
    txid_to_spend = ('9282a8cec7b821867ef49fd3cf588ab3b9603f17650b430addfeac9af5a8b09b')
    utxo_index = 0
    print("my address: ", my_address)
    print("my public key: ", my_public_key.hex())
    print("my private key: ", my_private_key.hex())
    response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index)
    print("response status code: ", response.status_code)
    print("reponse reason: ", response.reason)
    print("response text: ", response.text)
```

دو عدد اول 1019 و 1021 را در نظر می‌گیریم که هر کدام دو بایت را اشغال می‌کنند.

در این قسمت مقدار txout_scriptPubKey را برابر با مقدار زیر قرار می‌دهیم:

[OP_2DUP, OP_SUB, OP_HASH160, Hash160((PRRIME_NUM1- PRIME_NUM2).to_bytes(1, byteorder = “little”)), OP_EQUALVERIFY, OP_ADD, OP_HASH160, Hash160((PRRIME_NUM1 + PRIME_NUM2).to_bytes(2, byteorder = “little”)), OP_EQUAL)

در استک عناصر از بالا به پایین خوانده می‌شوند ولی در اینجا برای راحتی توضیح دادن، مقدار داخل استک را از ته به سر می‌خوانیم؛ به این شکل عمل می‌شود که ابتدا با دستور OP_2DUP دو عضوی که در ته استک قرار دارند کپی می‌شوند (در اینجا آن دو عضو باید دو عدد اول تعیین شده باشند)؛ این کپی به این دلیل انجام می‌شود که یک بار حاصل جمع دو عدد و یکبار حاصل تفریق دو عدد محاسبه می‌شود پس دو سری از این دو عدد باید موجود باشند. OP_SUB دو عدد از استک برمی‌دارد و حاصل تفریق آنها را در سر استک می‌گذارد و OP_HASH160، هش شده‌ی این حاصل تفریق را محاسبه می‌کند و در سر استک قرار می‌دهد و Hash160 از حاصل تفریق دو اصلی گرفته می‌شود؛ توجه شود که ابتدا مقدار تفریق آنها را با بایت تبدیل می‌کنیم و ترتیب خوانده شدن بیت‌های آن را تعیین می‌کنیم (پرارزش‌ترین بیت در انتهای لیست بایت‌هاست) و 1 نشان‌دهنده‌ی این است که حاصل تفریق

دو عدد 1021 و 1019 می‌شود 2 و در یک بایت جا می‌شود و سپس OP_EQUALVERIFY تساوی هش تفریق دو عدد اصلی با هش دو عدد وارد شده را بررسی می‌کند و درصورتی که مساوی بودند ادامه‌ی کار را انجام می2دهد و درغیراینصورت مقدار false را برمی‌گرداند؛ سپس OP_ADD دو عددی که در استک قرار دارند (سری دوم از دو عدد چون یک سری از آن برای تفریق استفاده شده بود) را برمی‌دارد و حاصل جمع آنها را در استک می‌گذارد و OP_HASH160 هش آن مقدار حاصل جمع را می‌گیرد و در استک می‌گذارد و سپس هش حاصل جمع دو عدد اصلی در استک گذاشته می‌شود (چون مجموع آنها دو بایت اشغال می‌کند در آرگومان to_bytes عدد دو قرار داده شده است) و سپس OP_EQUAL تساوی این دو عدد را بررسی می‌کند و درصورتی که مساوی بودند، true و در غیراینصورت false را در استک می‌گذارد. درواقع در اینجا از خاصیت preimage resistance تابه هش استفاده شده است که با داشتن هش شده‌ی حاصل جمع و تفریق نمی‌توان به حاصل جمع و تفریق رسید تا بتوانیم از آن طریق دو عدد اول را پیدا کنیم.

مقدار transactioin fee در این تراکنش برابر با 0.005 بیت کوین در نظر گرفته شده است.

```
(venv) → CA python3 Q2_part3_1.py
my address:  n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9
my public key:  0477ce41e46d710025cb7ae0bc425fbd59d2c1711706c8772d7616e2d073afd245f03c803dfea52c572b49e6ac0d42134c40c60219fc3650c89f7c0e0a678be26e
my private key:  409ff4d2516c040a7dd49806ad84a44b81133f08cacf16e59d2a3e4e272de769
response status code:  201
reponse reason:  Created
response text:  {
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "53bc834a89e334079517700045f05dd19488aca0f14442b33a0a06dadd3207cd",
    "addresses": [
      "n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9"
    ],
    "total": 1100000,
    "fees": 500000,
    "size": 247,
    "vsize": 247,
    "preference": "high",
    "relayed_by": "5.114.128.12",
    "received": "2023-05-24T08:52:46.079795554Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "9282a8cec7b821867ef49fd3cf588ab3b9603f17650b430addfeac9af5a8b09b",
        "output_index": 0,
        "script": "473044022060bd2fc514145fe108b0e06331d50d317c34b844e5fff8ac6e366ee2b737f5f402206a7337dc56f4ad0e8381298fc3012d7b367cd77d5cdb9fb3f753d6bd0ba884bb814104779ce41e46d710025cb7ae0bc425fbd59d2c1711706c8772d7616e2d073afd245f03c803dfea52c572b49e6ac0d42134c40c60219fc3650c89f7c0e0a678be26e",
        "output_value": 1600000,
        "sequence": 4294967295,
        "addresses": [
          "n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 2435054
      }
    ],
    "outputs": [
      {
        "value": 1100000,
        "script": "6a9914a6bb94c8792c395785787280dc188d114e1f339b8893a91488724900708988ea8735d1745271c0e4891d13feb87",
        "addresses": null,
        "script_type": "unknown"
      }
    ]
  }
}
```


هش این transaction برابر است با:

53bc834a89e334079517700045f05dd19488aca0f14442b33a0a06dadd3207cd

و صحت آن در این لینک قابل بررسی است.

3_2) خرج کردن با داشتن دو عدد اول:

```python
PRIME_NUM1 = 1021
PRIME_NUM2 = 1019

pv_key = "925NvBWuEWdw1Au6CYpgqovtQBhtS3TfjXiLo7NhmoBLMYeeby3"
my_private_key = bitcoin.wallet.CBitcoinSecret(pv_key)
my_public_key = my_private_key.pub

my_address = bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

def P2PKH_scriptPubKey(key):
    return [OP_DUP, OP_HASH160, Hash160(key), OP_EQUALVERIFY, OP_CHECKSIG]

def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = P2PKH_scriptPubKey(my_public_key)
    txout = create_txout(amount_to_send, txout_scriptPubKey)
    txin_scriptPubKey = [OP_2DUP, OP_SUB, OP_HASH160, Hash160((PRIME_NUM1 - PRIME_NUM2).to_bytes(1, byteorder="little")), OP_EQUALVERIFY,
                         OP_ADD, OP_HASH160, Hash160((PRIME_NUM1 + PRIME_NUM2).to_bytes(2, byteorder="little")), OP_EQUAL]
    txin = create_txin(txid_to_spend, utxo_index)
    txin_scriptSig = [PRIME_NUM1, PRIME_NUM2]

    new_tx = create_signed_transaction(txin, [txout], txin_scriptPubKey, txin_scriptSig)

    return broadcast_transaction(new_tx)


if __name__ == '__main__':
    all_money = 0.011
    amount_to_send = 0.01
    txid_to_spend = ('53bc834a89e3340795177000045f05dd19488aca0f14442b33a0a06dadd3207cd')
    utxo_index = 0
    print("my address: ", my_address)
    print("my public key: ", my_public_key.hex())
    print("my private key: ", my_private_key.hex())
    response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index)
    print("response status code: ", response.status_code)
    print("reponse reason: ", response.reason)
    print("response text: ", response.text)
```

در اینجا مقدار txin_scriptPubKey را برابر با txout_scriptPubKey توضیح داده شده در مرحله قبل قرار می‌دهیم و txin_scriptSig را برابر با [PRIME_NUM1, PRIME_NNUM2] قرار می‌دهیم زیرا همانطور که در قسمت قبل توضیح داده شد، لازمه‌ی خرج کردن این تراکنش داشتن دو عدد اول است.

مقدار transaction fee در این تراکنش برابر با 0.001 بیت کوین است.

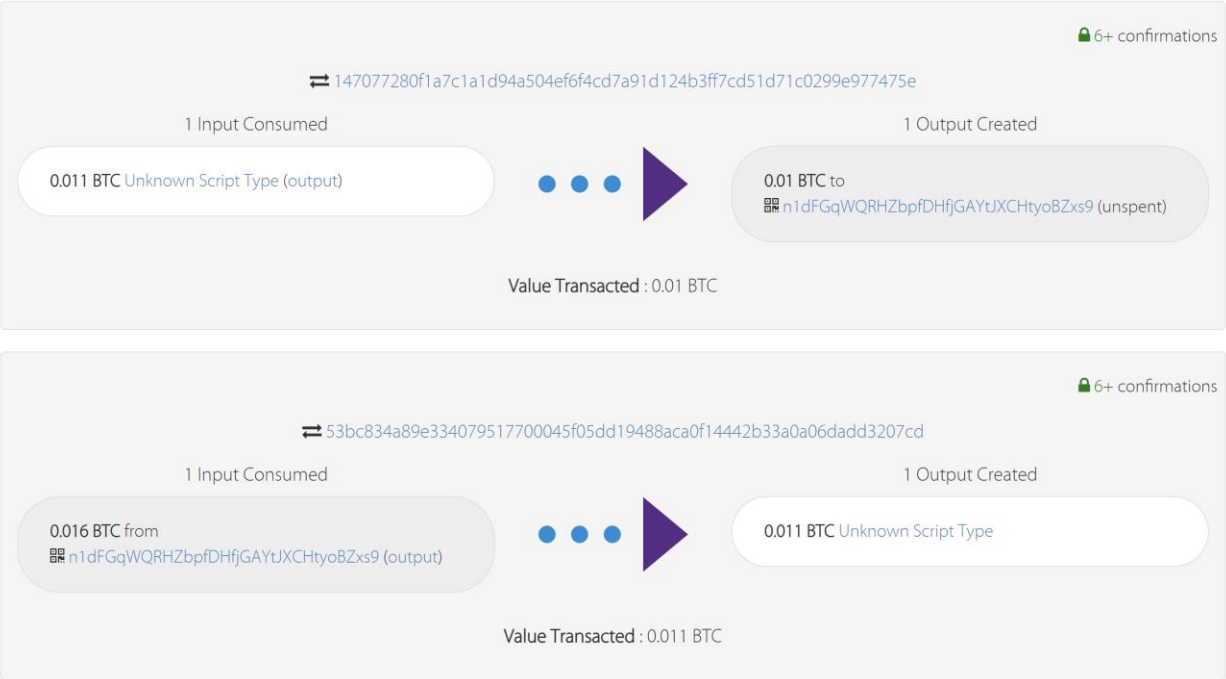
```
(venv) → CA python3 Q2_part3_2.py
my address:  n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9
my public key:  0477ce41e46d710025cb7ae0bc425fbd59d2c1711706c8772d7616e2d073afd245f03c803dfea52c572b49e6ac0d42134c40c60219fc3650c89f7c0e0a678be26e
my private key:  409ff4d2516c040a7dd49806ad84a44b81133f08cacf16e59d2a3e4e272de769
response status code:  201
reponse reason:  Created
response text:  {
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "1470777280f1a7c1a1d94a504ef6f4cd7a91d124b3ff7cd51d71c0299e977475e",
    "addresses": [
      "n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9"
    ],
    "total": 1000000,
    "fees": 100000,
    "size": 91,
    "vsize": 91,
    "preference": "high",
    "relayed_by": "5.114.128.12",
    "received": "2023-05-24T09:06:18.631869369Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "53bc834a89e3340795177000045f05dd19488aca0f14442b33a0a06dadd3207cd",
        "output_index": 0,
        "script": "02fd03032fb03",
        "output_value": 1100000,
        "sequence": 4294967295,
        "script_type": "unknown",
        "age": 2435061
      }
    ],
    "outputs": [
      {
        "value": 1000000,
        "script": "76a914dc932f50397a45847c79fb3d62536dac7ad75edd88ac",
        "addresses": [
          "n1dFGqWQRHZbpfDHfjGAYtJXCHtyoBZxs9"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```

هش این transaction برابر است با:

147077280f1a7c1a1d94a504ef6f4cd7a91d124b3ff7cd51d71c0299e977475e

و صحت آن در این لینک قابل بررسی است.

دو تراکنش بخش3-1 و 3-2:



صحت تمامی تراکنش‌های بخش دوم در این لینک قابل بررسی است.

بخش سوم: استخراج بیت کوین

ابتدا یک کلید خصوصی 32 بیتی تولید می‌کنیم که به فرمت WIF است و در بخش اول نحوه‌ی ساخت آن توضیح داده شد. برای ایجاد تراکنش coinbase، txout_scriptPubKey برابر با خروجی تابع P2PKH_scriptPubKey به ازای کلید عمومی قرار داده می‌شود و آیدی تراکنشی که از آن خرج می‌کنیم را برابر با 64 بیت صفر قرار می‌دهیم و ایندکس آن را برابر با 0xFFFFFFFF قرار می‌دهیم؛ coinbase_data را برابر با هگز نام و شماره دانشجوییم قرار دادم و scriptSig را نیز با توجه با اندازه‌ی coinbase_data ست می‌کنیم، در بایت هر 8 بیت را یک بایت در نظر می‌گیریم ولی در هگز هر 4 بیت را یک واحد پس تقسیم بر دو انجام می‌شود. اندازه‌ی پول را نیز برابر با مقدار کنونی block reward که برابر با 6.24 بیت کوین است قرار می‌دهیم.

```
def get_tx(txin, txout):
    tx = CMutableTransaction([txin], [txout])
    coinbase_scriptig = CScript(
        [int(COINBASE_DATA, 16).to_bytes(len(COINBASE_DATA)//2, 'big')])
    txin.scriptSig = coinbase_scriptig
    return tx

def create_transaction():
    txout_scriptPubKey = P2PKH_scriptPubKey(my_public_key)
    txid_to_spend, index = (64*'0'), int('0xFFFFFFFF', 16)
    txin = create_txin(txid_to_spend, index)
    txout = create_txout(BLOCK_REWARD, txout_scriptPubKey)
    return get_tx(txin, txout)
```

برای بدست آوردن درجه سختی target فرمولی به صورت زیر وجود دارد که از آن استفاده می‌کنیم.

$$Target = coefficient * 2^{8 * (Exponent - 3)}$$

ابتدا bits را مشخص می‌کنیم که با توجه به اینکه درجه سختی ما باید به گونه‌ای باشد که 4 بیت سمت چپ 0 است آن را برابر با 0x1f010000 قرار می‌دهیم؛ ابتدا exponent را جدا می‌کنیم که رقم سوم و چهارم bits است که می‌شود بیت دوم و سوم از سمت چپ و coefficient برابر با بیت پنجم به بعد است و سپس از فرمول بالا برای محاسبه‌ی targetاستفاده می‌کنیم و سپس فرمت آن را به هگز تبدیل می‌کنیم و توسط تابع zfill(64) بقیه جایگاه‌ها را با 0 پر می‌نیم تا عدد 64 بیتی شود.

```
def calculate_target():
    exponent = BITS[2:4]
    coefficient = BITS[4:]
    target = int(coefficient, 16) * (int('2', 16) ** (8 * (int(exponent, 16) - 3)))
    target_hex = format(target, 'x')
    target_hex = str(target_hex).zfill(64)
    print("Target in hex: ", target_hex)
    return bytes.fromhex(target_hex)
```

دلیل اینکه در بعضی از توابع از reverese کردن نتیجه استفاده می‌کنیم این است که در توابع bitcoin، آیدی تراکنش‌ها باید به فرمت رشته باینری و little endian باشند یعنی کم‌ارزش‌ترین بیت در کمترین آدرس ذخیره می‌شود ولی explorer های بلاک چین داده را به فرمت hexadecimal و big endian نمایش می‌دهند یعنی برعکس little endian؛ پس نیاز است که کاراکترها در مواقعی برعکس شوند.

Partial header را با اطلاعاتی که داریم می‌سازیم؛ این اطلاعات شامل ورژن، هش هدر بلوک قبلی، Merkle root و زمان کنونی است. struct.pack(‘<L’, nonce) ورودی خود (nonce) را به فرمت little endian و بدون علامت برمی‌گرداند (L نشان دهنده‌ی long بودن خروجی است)

برای استخراج بیت کوین باید nonce را به گونه‌ای قرار دهیم تا هش حاصل از partial header و nonce از مقدار گفته شده کوچکتر باشد؛ از nonce = 0 شروع می‌کنیم و هربار یک واحد به آن اضافه می‌کنیم تا مقدار مورد نظر پیدا شود.

```
def mine_bitcoin(n, prev_block_hash):
    tx = create_transaction()
    merkle_root = b2lx(sha256(sha256(tx.serialize()).digest()).digest())
    print("Merkle root: ", merkle_root)
    block_body = b2x(tx.serialize())
    print("Block body: ", block_body)
    target = calculate_target()
    partial_header = struct.pack("<L", VERSION) + reverse(bytes.fromhex(prev_block_hash)) + reverse(bytes.fromhex(merkle_root)) + struct.pack('<LL', int(time.time()), int(BITS, 16))
    nonce = 0
    start_time = time.time()
    while nonce <= 0xFFFFFFFF:
        header = partial_header + struct.pack('<L', nonce)
        hash = sha256(sha256(header).digest()).digest()
        if reverse(hash) < target:
            hash_rate = nonce / (time.time() - start_time)
            print_info(nonce, hash, header, hash_rate)
            return
        nonce += 1
```

همان طور که دیده می‌شود برای بدست آوردن merkle root از هش تراکنشی استفاده می‌کنیم که در بلوک قرار دارد و همچنین بدنه‌ی بلوک نیز برابر با این تراکنش است چون بلوک ما تنها از یک تراکنش تشکیل شده است.

هش ریت نیز با توجه به تعداد nounce بررسی شده در واحد زمان محاسبه شده است.

کل بلوک را نیز با توجه به تعداد تراکنش‌های آن که در اینجا برابر با 1 است، سایز بلوک که از مجموع سایز اجزای تشکیل دهنده‌ی آن تشکیل شده و block body و header و magic number که یک مقدار از قبل تعیین شده است، می‌سازیم و آن را چاپ می‌کنیم.

```
def get_block(header, block_body):
    tx_count = b'\x01'
    block_size = len(header) + len(tx_count) + len(block_body)
    magic_number = 0xD9B4BEF9.to_bytes(4, "little")
    block = magic_number + struct.pack("<L", block_size) + header + tx_count + block_body
    return block
```

در نهایت با اجرای برنامه به ازای n = 9501 که هش بلوک قبلی معادل با آن برابر با مقدار

000000004383fe923306472b416c0c2432523b3f5fbc2ca25def2c7155f9b5bb

است مقادیر بدست آمده را چاپ می‌کنیم که مطابق با زیر است:

```
→  CA python3 Q3.py
please enter your block number: 9501
please enter prevoius block hash: 000000004383fe923306472b416c0c2432523b3f5fbc2ca25def2c7155f9b5bb

Merkle root:  1f927305a47674dd04658657bddc859e20101cd29b4bfe9180f83bce2aac9f36

Block body:  0100000001000000000000000000000000000000000000000000000000000000000000000000000000ffffffff1d1c383130313939353031417661 4d69726d6f68616d6d61646d61686469ffffffff0140be402
5000000001976a9146fc8ee79848b9e85d882597b64c3c99d9f49ded488ac00000000

Target in hex:  0001000000000000000000000000000000000000000000000000000000000000

Success Mining!

nonce: 27013

Hash rate: 989776.7465296887 H/s

Block hash: 0000a5fa506776ecc6af10c4a05f690fc37cf2b1353ccc97e5d44ac05a9bd4ce
Block header: 02000000bbb5f955712cef5da22cbc5f3f3b5232240c6c412b47063392fe83430000000369fac2ace3bf88091fe4b9bd21c10209e85dcbd57866504dd7476a40573921f05b4706400000 11f856900
00

Block in hex: f9beb4d9c30000002000000bbb5f955712cef5da22cbc5f3f3b5232240c6c412b47063392fe83430000000369fac2ace3bf88091fe4b9bd21c10209e85dcbd57866504dd7476a40573921f05b470
640000011f8569000001010000000100000000000000000000000000000000000000000000000000000000000000000000000000000000ffffffff1d1c383130313939353031417661 4d69726d6f68616d6d61646d61686469ffffffff01
40be402500000000001976a9146fc8ee79848b9e85d882597b64c3c99d9f49ded488ac00000000

Version: 2

Bits: 0x1f010000

Time stamp: 1685107717
```