تمرین کامپیوتری اول رمز ارز

سنا ساری نوایی شماره دانشجویی : 810199435

قسمت اول : تولید آدرس

سوال 1:

هدف این بخش تولید آدرس برای انجام تراکنش بیت کوینی میباشد. قبل از ایجاد آدرس، باید کلید خصوصی و عمومی خودمان را بسازیم.

برای تولید کلید خصوصی، از کتابخانه ecdsa کمک میگیریم. در اینجا کلید خصوصی با استفاده از الگوریتم منحنی بیضوی SECP256k1 تولید میشود. کلید خصوصی با استفاده از متد ecdsa.SigningKey.generate() تولید میشود که پارامتر منحنی را به عنوان آرگومان برای تعیین منحنی بیضوی استفاده میکند.

```
def generatePrivateKey ():
    private_key = ecdsa.SigningKey.generate(curve=ecdsa.SECP256k1)
    return private_key

print("Private Key : " + private_key.to_string().hex())
```

حال باید کلید عمومی را از روی کلید خصوصی بدست بیاوریم. در این تابع کلی خصوصی را به عنوان ورودی به تابع میدهیم و با استفاده از متد get_varifying_key() کلید عمومی را به ما میدهد.

```
def generatePublicKey(private_key):
    public_key = private_key.get_verifying_key()
    return public_key

print("Public Key : " + (b'\x04' +
generatePublicKey(private_key).to_string()).hex())
```

در صورت سوال از ما خواسته شده کلید خصوصی به فرم WIF یا همان Wallet Import Format باشد.

فرمت WIF برای نمایش کلیدهای خصوصی به روش کاربرپسندتر ارائه میشود. رشته WIF از یک version byte، کلید خصوصی و checksum تشکیل شده است.

برای کلید خصوصی testnet، مقدار prefix آن 0xEF میباشد. پس آن را به ابتدای کلید خصوصی اضافه میکنیم. سپس دو بار Hash SHA-256 را بر روی آن اعمال میکنیم. 4 بایت اول را به عنوان checksum در نظر میگیریم و به انتهای کلید خصوصی مان اضافه میکنیم. سپس نتیجه نهایی با استفاده از طرح رمزگذاری base58.b58encode() برای تولید رشته WIF کدگذاری میشود.

```
def wif(private_key):
    TestnetPrivateKey = b'\xEF'
    key =  TestnetPrivateKey + private_key

    sha1 = hashlib.sha256(key)
    sha1 = sha1.digest()

    sha2 = hashlib.sha256(sha1)
    sha2 = sha2.digest()

    checksum_bytes = sha2[:4]
    result = key + checksum_bytes
    result = base58.b58encode(result)
    return result
print("Private WIF : " + wif(private_key.to_string()).decode('utf-8'))
```

حال باید آدرسمان را تولید کنیم. در تابع p2pkh(public_key) یا به عبارتی pay-to-public-key-hash یک آدرس برای کلید عمومی داده شده ایجاد میگردد.

آدرس P2PKH نوعی آدرس بیت کوین است که معمولاً برای دریافت پرداخت استفاده می شود. این شامل یک version byte، یک هش کلید عمومی و یک checksum است.

برای تبدیل کلید عمومی به فرمت p2pkh، ابتدا از کلید عمومی hashlib.sha256 میگیریم و سپس ripemd160 را روی آن تنظیم میکنیم. Version byte برای آدرس P2PKH روی 0x6F تنظیم شده است که برای شبکه تست بیت کوین استفاده می شود. پس آن را به ابتدای hash_160 اضافه میکنیم. سپس از آن دوبار SHA-256 میگیریم. در نهایت مانند قبل، 4 بایت اول آن را به عنوان checksum در نظر میگیریم و به انتهای آدرس اضافه میکنیم. سپس نتیجه نهایی با استفاده از طرح رمزگذاری Base58Check برای تولید آدرس P2PKH کدگذاری می شود.

```
def p2pkh(public_key):
    sha256 = hashlib.sha256(public_key)
    sha256 = sha256.digest()
    hash_160 = hashlib.new('ripemd160')
    hash_160.update(sha256)
    hash_160 = hash_160.digest()
    hash_160 = b'\x6F' + hash_160

    sha1 = hashlib.sha256()
    sha1.update(hash_160)
    sha1 = sha1.digest()

    sha2 = hashlib.sha256()
    sha2.update(sha1)
```

```
    sha2 = sha2.digest()

    checksum_bytes = sha2[:4]
    result = hash_160 + checksum_bytes
    result = base58.b58encode(result)
    return result

print("Public P2PKH : " + p2pkh(b'\x04' +
generatePublicKey(private_key).to_string()).decode('utf-8'))
```

نتیجه کد بخش بالا به شکل زیر میباشد.

```
Private Key : 630c9cff7c75484f9869cb48423961f61f03dbd270d5756a28a9f9513d394b70
Public Key : 04cf0e16608f4dae3d5e987d8a6fdad32cd6c19bf1c18e0460f932d9edbe79785f423bed4069b54a5aea19b9a416518e20dcd0cc4f5c017d61c1672254d54cff1c
Private WIF : 92LYF6g8P1mVvK5WbEMo5ke2ogP5vVpKAMY1ogjsanBPMrnJvdG
Public P2PKH : moHSYHfe16qTKkjDZ771JvGB4UPZg79w69
```

تفاوت میان آدرس های شبکه اصلی و آزمایشی در همان prefix است که در ابتدای تابع WIF به کلیدمان اضافه میشود. ما به دلیل اینکه با شبکه آزمایشی کار میکنیم، از 0x6F استفاده کردیم اما اگر میخواستیم با شبکه اصلی کار کنیم، باید از x80 بهره میبردیم. برای کلید عمومی نیز در شبکه آزمایشی از x6F و در شبکه اصلی از x00 استفاده میکنیم. این تفاوت ها منجر میشود که WIF در شبکه آزمایشی با 9 و در شبکه اصلی با 5 شروع شود. از طرف دیگر P2PKH در شبکه اصلی با 1 و در شبکه آزمایشی با m یا n شروع میشود.

سوال 2:

در این بخش میخواهیم یک vanity address تولید کنیم. میخواهیم 3 کاراکتر موردنظرمان در جایگاه 2 تا 4 آدرس تولید شده قرار بگیرد و از آن آدرس استفاده کنیم. پس در یک حلقه while True آنقدر آدرس تولید میکنیم تا بالاخره به آدرس مدنظرمان برسیم.

```
def vanity(goal):
    while True:
        private_key = generatePrivateKey()
        public_key = b'\x04' + generatePublicKey(private_key).to_string()
        address = p2pkh(public_key)
        if goal == address[1:4]:
            return address, private_key, public_key

vanity_address, private_key, public_key = vanity(b'san')

print("Vanity Address : " + vanity_address.decode('utf-8'))
```

خروجی این کد در شکل زیر آمده است:

```
PS D:\ut-01-2\Crypto\CA\CA1> & C:/Users/Sana/AppData/Local/Programs/Python/Python310/python.exe d:/ut-01-2/Crypto/CA/CA1/Code/Part1_Q2.py
Private Key : a25479c4d2a891a65198ad0d2a4739d81d6ca669763babdb19678b12d31cbd0c
Public Key : 0464027e8035123673b39efdc1a05b637012ec314158b3dc436bd2104a7a98dc8497ad877c1bcff24bbc1d6b761adc6c507909cba017542338a72df096886939c7
Private WIF : 92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGkeTwN7ef79ETNC
Vanity Address : msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7
```

Private Key : a25479c4d2a891a65198ad0d2a4739d81d6ca669763babdb19678b12d31cbd0c

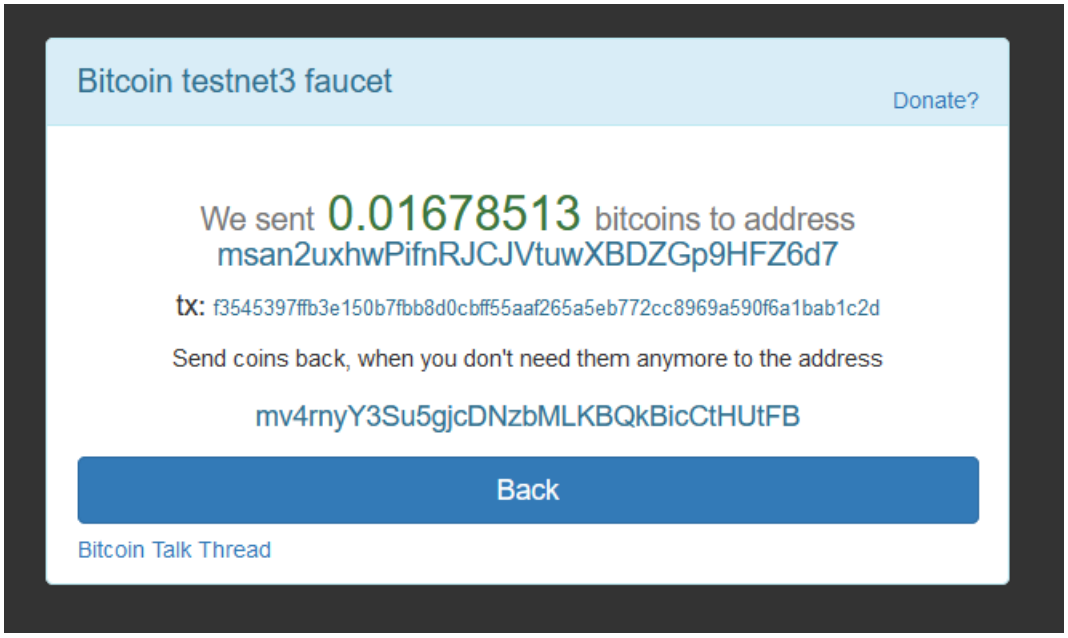
Public Key : 0464027e8035123673b39efdc1a05b637012ec314158b3dc436bd2104a7a98dc8497ad877c1bcff24bbc1d6b761adc6c507909cba017542338a72df096886939c7

Private WIF : 92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGkeTwN7ef79ETNC

Vanity Address : msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7

قسمت دوم : انجام تراکنش

قبل از اینکه به سراغ سوال های این بخش برویم، ابتدا با استفاده از سایت https://coinfaucet.eu/en/btc-testnet بیت کوین دریافت میکنیم. به این صورت است که vanity address بدست آمده در بخش قبل را در اینجا وارد میکنیم و بیت کوین دریافت می نماییم. خروجی این بخش به صورت زیر است. مقدار بیت کوین دریافت شده برابر 0.01678513 میباشد.



با کلیک بر روی tx مشخص شده در تصویر، میتوانیم تراکنش را مشاهده نماییم.

تمرین کامپیوتری اول رمز ارز

سنا ساری نوایی
شماره دانشجویی : 810199435

Private WIF : 92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGkeTwN7ef79ETNC

Vanity Address : msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7

سوال 1:

بخش اول

کد این بخش به صورت زیر میباشد.

```
import bitcoin.wallet
from bitcoin.core import COIN, b2lx, serialize, x, lx, b2x
from utils import *

bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret('92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGke
TwN7ef79ETNC')
my_public_key = my_private_key.pub
my_address =
bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

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

def P2PKH_scriptSig(txin, txout_1, txout_2, txin_scriptPubKey):
    signature = create_OP_CHECKSIG_signature2(txin, txout_1, txout_2,
txin_scriptPubKey, my_private_key)
    return [signature, my_public_key]

def send_P2PKH_tx_two_outputs(amount_to_send1, amount_to_send2,
txid_to_spend, utxo_index):
```

```
    txin = create_txin(txid_to_spend, utxo_index)
    txout_1 = create_txout(amount_to_send1, [OP_1])
    txout_2 = create_txout(amount_to_send2, [OP_0])

    txin_scriptPubKey = P2PKH_scriptPubKey()
    txin_scriptSig = P2PKH_scriptSig(txin, txout_1, txout_2,
txin_scriptPubKey)
    tx = create_signed_transaction2(txin, txout_1, txout_2,
txin_scriptPubKey, txin_scriptSig)

    return broadcast_transaction(tx)

if __name__ == '__main__':
    amount_to_spend_1 = 0.01677
    amount_to_spend_2 = 0.000001
    txid_to_spend =
('f3545397ffb3e150b7fbb8d0cbff55aaf265a5eb772cc8969a590f6a1bab1c2d')
    utxo_index = 1

    print("my_address: ", my_address)
    print("my_public_key: ", my_public_key.hex())
    print("my_private_key: ", my_private_key.hex())

    response = send_P2PKH_tx_two_outputs(amount_to_spend_1,
amount_to_spend_2, txid_to_spend, utxo_index)

    print("response status code: " ,response.status_code, "response
reason: ", response.reason)
    print("response text: ", response.text)
```

در اینجا تراکنش P2PKH را با دو خروجی انجام داده ایم. دو مقداری که برای انجام تراکنش با دو خروجی تعریف میکنیم، باید جمع آنها از جمع کل بیت کوین ما کمی کمتر باشد و مقداری را برای transaction fee قرار میدهیم. مقدار txid_to_spend را برابر همان tx بدست آمده میگذاریم. برای اینکه خروجی اول توسط هیچ فردی قابل خرج کردن نباشد، مقدار utxo_index را برابر 1 قرار میدهیم. سپس دو مقدار مختلف برای مقادیری که میخواهیم به دو خروجی موردنظر بیتکوین بدهیم را مشخص میکنیم و توابع موردنظر را فراخوانی میکنیم.

در توابع P2PKH_scriptPubKey و P2PKH_scriptSig ، scriptPubKey و scriptSig را برای یک تراکنش P2PKH تعریف میکنند. scriptPubKey اسکریپت قفل کننده ای است که شرایطی را مشخص می کند که در آن خروجی می تواند صرف شود، و scriptSig اسکریپت باز کردن قفل است که داده های لازم را برای برآورده کردن آن شرایط فراهم می کند.

تابع send_P2PKH_tx_two_outputs یک تراکنش با دو خروجی را ایجاد و پخش میکند.

تمرین کامپیوتری اول رمز ارز

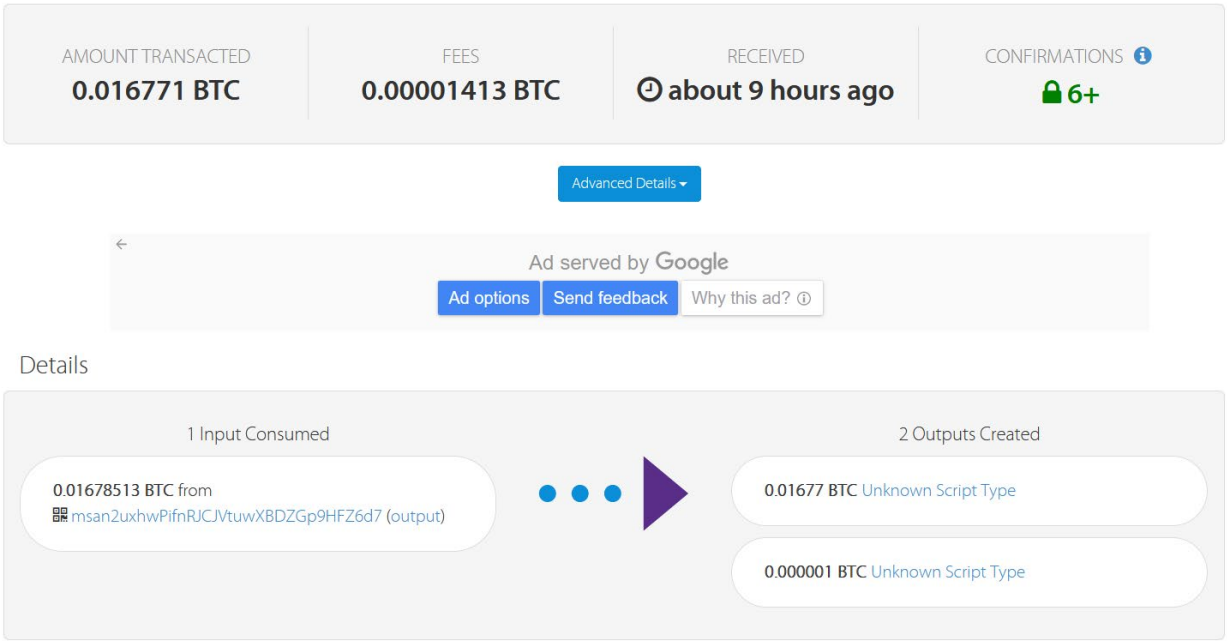
سنا ساری نوایی
شماره دانشجویی : 810199435

هش بدست آمده برابر است با:

"hash": "992bbf6f8082685f5c543abe4ca42eb6e22a893fc9b79270ec0185d4c95ea1a1"

تراکنش انجام شده در لینک زیر قابل مشاهده است.

https://live.blockcypher.com/btc-/testnet/tx/992bbf6f8082685f5c543abe4ca42eb6e22a893fc9b79270ec0185d4c95ea1a1



بخش دوم

کد این بخش به صورت زیر میباشد.

```
import bitcoin.wallet
from bitcoin.core import COIN, b2lx, serialize, x, lx, b2x
from utils import *

bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret('92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGke
TwN7ef79ETNC')
my_public_key = my_private_key.pub
my_address =
bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

def P2PKH_scriptPubKey():
    return [OP_DUP, OP_HASH160, Hash160(my_public_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_P2PKH_tx(amount_to_spend, txid_to_spend, utxo_index,
txout_scriptPubKey):
    txin = create_txin(txid_to_spend, utxo_index)
    txout = create_txout(amount_to_spend, txout_scriptPubKey)

    txin_scriptPubKey = [OP_1]
    txin_scriptSig = []
    tx = create_signed_transaction(txin, txout, txin_scriptPubKey,
txin_scriptSig)

    return broadcast_transaction(tx)

if __name__ == '__main__':
    amount_to_spend = 0.01675
    txid_to_spend =
('992bbf6f8082685f5c543abe4ca42eb6e22a893fc9b79270ec0185d4c95ea1a1')
    utxo_index = 0

    print("my_address: ", my_address)
```

```
    print("my_public_key: ", my_public_key.hex())
    print("my_private_key: ", my_private_key.hex())

    txout_scriptPubKey = P2PKH_scriptPubKey()
    response = send_P2PKH_tx(amount_to_spend, txid_to_spend,
utxo_index, txout_scriptPubKey)

    print("response status code: " ,response.status_code, "response
reason: ", response.reason)
    print("response text: ", response.text)
```

در این بخش میخواهیم نتیجه تراکنش بخش قبل را به حساب خودمان برگردانیم. برای اینکار txin_scriptPubKey را برابر [OP_1] قرار دادیم. همچنین مقداری که میخواهیم به حساب خودمان برگردانیم باید کمی کمتر از خود بیتکوین باشد زیرا باید transaction fee آن را در نظر بگیریم. مقدار txid_to_spend را اپدیت میکنیم و برابر tx بخش قبل قرار میدهیم. از طرف دیگر txin_scriptSig باید خالی باشد زیرا اهمیتی برای تشخیص صحت تراکنش ندارد.

```
sana@DESKTOP-ESNMQ0P:/mnt/d/ut-01-2/Crypto/CA/CA1/Code$ python3 Part2_Q1_2.py
my_address:  msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7
my_public_key:  0464027e8035123673b39efdc1a05b637012ec314158b3dc436bd2104a7a98dc8497ad877c1bcff24bbc1d6b761adc6c507909cba017542338a72df096886939c7
my_private_key:  a25479c4d2a891a65198ad0d2a4739d81d6ca669763babdb19678b12d31cbd0c
response status code:  201 response reason:  Created
response text:  {
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "cc0fdb6bc13d0dc67db2936a1f138343fc110db4361b938e2f9b57ef00821aba",
    "addresses": [
      "msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7"
    ],
    "total": 1675000,
    "fees": 2000,
    "size": 85,
    "vsize": 85,
    "preference": "low",
    "relayed_by": "104.28.214.161",
    "received": "2023-05-27T12:36:33.933785797Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "992bbf6f8082685f5c543abe4ca42eb6e22a893fc9b79270ec0185d4c95ea1a1",
        "output_index": 0,
        "output_value": 1677000,
        "sequence": 4294967295,
        "script_type": "unknown",
        "age": 2435653
      }
    ],
    "outputs": [
      {
        "value": 1675000,
        "script": "76a914845a9b96dfeac40f8cd6118afa717608067a086988ac",
        "addresses": [
          "msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```

هش بدست آمده برابر است با:

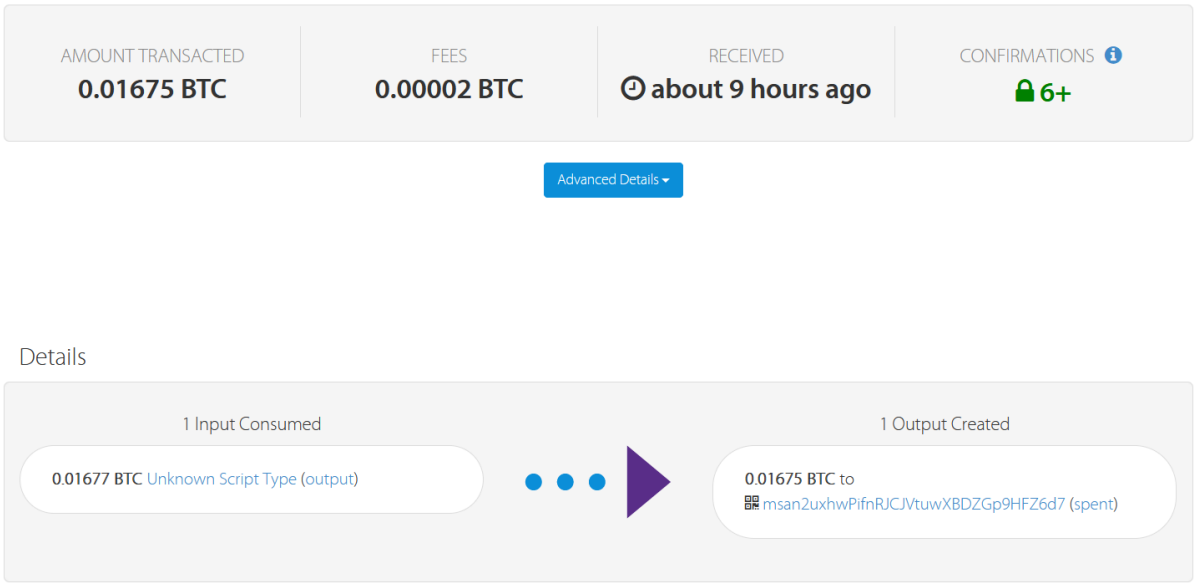
"hash": "cc0fdb6bc13d0dc67db2936a1f138343fc110db4361b938e2f9b57ef00821aba"

تراکنش انجام شده در لینک زیر قابل مشاهده است.

تمرین کامپیوتری اول رمز ارز

سنا ساری نوایی
شماره دانشجویی : 810199435

https://live.blockcypher.com/btc-testnet/tx/cc0fdb6bc13d0dc67db2936a1f138343fc110db4361b938e2f9b57ef00821aba/



سوال 2:

بخش اول

کد این بخش به صورت زیر میباشد

```
import bitcoin.wallet
from bitcoin.core import COIN, b2lx, serialize, x, lx, b2x
from utils import *

bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret("92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGkeTwN7ef79ETNC")
my_public_key = my_private_key.pub
my_address =
bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

private_key1=
bitcoin.wallet.CBitcoinSecret('922NDRTK861R1aAmbJ9PWFy53EFRXYQmEodoCRmngUgf5uAvtua')
public_key1 = private_key1.pub
```

```
private_key2 =
bitcoin.wallet.CBitcoinSecret('91jxGSKPdLsqnpDR1cdkC2Qg1xi6TuiUxxS5KYa
x5BUqhezx68B')
public_key2 = private_key2.pub

private_key3 =
bitcoin.wallet.CBitcoinSecret('92WRR9mH2pbvVkHHPhjcLcGBhXKvdSQEE3spUsT
BvG621T42khZ')
public_key3 = private_key3.pub

def P2PKH_scriptPubKey():
    return [OP_DUP, OP_HASH160, Hash160(my_public_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_P2PKH_tx(amount_to_spend, txid_to_spend, utxo_index,
txout_scriptPubKey):
    txin = create_txin(txid_to_spend, utxo_index)
    txout = create_txout(amount_to_spend, txout_scriptPubKey)

    txin_scriptPubKey = P2PKH_scriptPubKey()
    txin_scriptSig = P2PKH_scriptSig(txin, txout, txin_scriptPubKey)
    tx = create_signed_transaction(txin, txout, txin_scriptPubKey,
txin_scriptSig)

    return broadcast_transaction(tx)

if __name__ == '__main__':
    amount_to_spend = 0.01635
    txid_to_spend =
('cc0fdb6bc13d0dc67db2936a1f138343fc110db4361b938e2f9b57ef00821aba')
    utxo_index = 0

    print("my_address: ", my_address)
    print("my_public_key: ", my_public_key.hex())
    print("my_private_key: ", my_private_key.hex())

    txout_scriptPubKey = [OP_2, public_key1, public_key2, public_key3,
OP_3, OP_CHECKMULTISIG]
    response = send_P2PKH_tx(amount_to_spend, txid_to_spend,
```

```
utxo_index, txout_scriptPubKey)

    print("response status code: " ,response.status_code, "response
reason: ", response.reason)
    print("response text: ", response.text)
```

در این بخش میخواهیم از میان 3 آدرس بیت کوین که تولید کرده ایم، 2 تا از آنها قابل خرج کردن باشد.

ابتدا 3 آدرس موردنظر را بدست میاوریم.

```
Private Key : 39c9134a18933bc0731afad1b6b20de7ebc27e97d0caeb6b910edae2efe5db9d
Public Key : 04195be239b86edfadba20287c89461ae16bdd568514ce57c9fb83d072710615adecb32cad17b7869dc94cd26a4b12d8b3f48e2fbfa4762cd9eded3f13a678d3ec
Private WIF : 922NDRTK861R1aAmbJ9PWFy53EFRXYQmEodoCRmngUgf5uAvtua
Vanity Address : msan8ncb8tUdqhkqj4mpc8F2n3L6747mT1

Private Key : 1484990d0877cce5107284528c07ee677752493dc6ffc39a2b9cb0ff2b907fe0
Public Key : 047d255108aad50a0217a8d4f30f654babd0d9bd1afc0096aac7d27edd9deda77c2f6096decd91d43ff527e611564312ef768dc23611935e0044c32eb8250115e7
Private WIF : 91jxGSKPdLsqnpDR1cdkC2Qg1xi6TuiUxxS5KYax5BUqhezx68B
Vanity Address : msanxoRnva9jsij4tR68FAKVKHSo9wcxeo

Private Key : 797cfd3b778566886a74ef679a24d7cdfa25e120b6ce8ccee185995075ee8e77
Public Key : 04fef7f71c4199d5c109ddf59bc8a223ec843eb17f389478b1b094cd9ba1925b9e0a650720b657e0b093dbfd16f618ca39371019119273 68a1285c2e1e2aa42604
Private WIF : 92WRR9mH2pbvVkHHPhjcLcGBhXKvdSQEE3spUsTBvG621T42khZ
Vanity Address : msanPUpjSNBiTet8VLr5w6jSW8nZqLaU3r

my_address:  msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7
my_public_key:  0464027e8035123673b39efdc1a05b637012ec314158b3dc436bd2104a7a98dc8497ad877c1bcff24bbc1d6b761adc6c507909cba017542338a72df096886939c7
my_private_key:  a25479c4d2a891a65198ad0d2a4739d81d6ca669763babdb19678b12d31cbd0c
```

مقدار txout_scriptPubKey به صورت زیر میباشد که آن را بررسی میکنیم. در ابتدا OP_CHECKMULTISIG را از استک میخوانیم. پس از آن به OP_3 میرسیم و 3 عضو از استک را پس از این میخوانیم که همان کلید عمومی های 3 آدرس تولید شده مان میباشند. در نهایت OP_2 را میخوانیم که یعنی پس از این دو عضو دیگر هم بخوان که همان 2 عضو از 3 آدرس تولید شده مان میباشد.

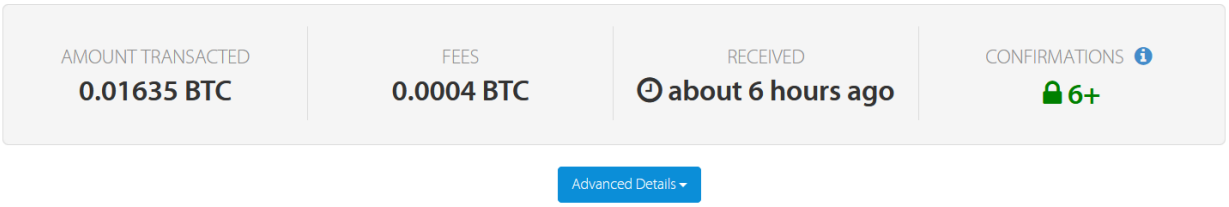
[OP_2, public_key1, public_key2, public_key3, OP_3, OP_CHECKMULTISIG]

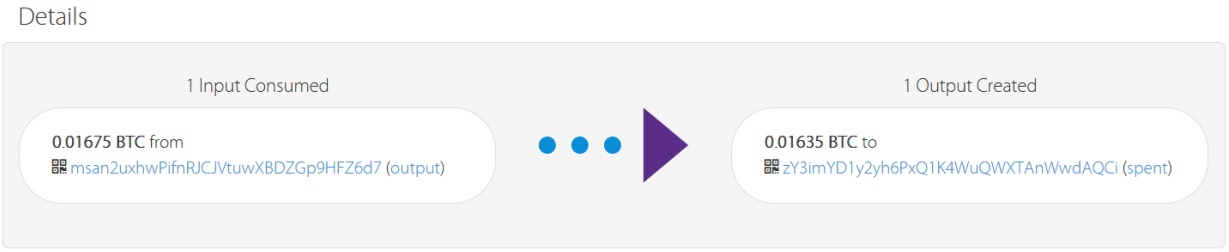
هش بدست آمده برابر است با:

"hash": "ecfa0348313ef309d7f18c5fb0007bdacf573dfec53921cb5e05e2bb609113b2"

تراکنش انجام شده در لینک زیر قابل مشاهده است.

https://live.blockcypher.com/btc-/testnet/tx/ecfa0348313ef309d7f18c5fb0007bdacf573dfec53921cb5e05e2bb609113b2





بخش دوم

حال میخواهیم پول تراکنش قبل را به حساب خودمان برگردانیم. کد این بخش به صورت زیر میباشد.

```
import bitcoin.wallet
from bitcoin.core import COIN, b2lx, serialize, x, lx, b2x
from utils import *

bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret("92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGke
TwN7ef79ETNC")
my_public_key = my_private_key.pub
my_address =
bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

private_key1=
```

```
bitcoin.wallet.CBitcoinSecret('922NDRTK861R1aAmbJ9PWFy53EFRXYQmEodoCRm
ngUgf5uAvtua')
public_key1 = private_key1.pub

private_key2 =
bitcoin.wallet.CBitcoinSecret('91jxGSKPdLsqnpDR1cdkC2Qg1xi6TuiUxxS5KYa
x5BUqhezx68B')
public_key2 = private_key2.pub

private_key3 =
bitcoin.wallet.CBitcoinSecret('92WRR9mH2pbvVkHHPhjcLcGBhXKvdSQEE3spUsT
BvG621T42khZ')
public_key3 = private_key3.pub

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

def P2PKH_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_P2PKH_tx(amount_to_spend, txid_to_spend, utxo_index,
txout_scriptPubKey):
    txin = create_txin(txid_to_spend, utxo_index)
    txout = create_txout(amount_to_spend, txout_scriptPubKey)

    txin_scriptPubKey = [OP_2, public_key1, public_key2, public_key3,
OP_3, OP_CHECKMULTISIG]
    txin_scriptSig = P2PKH_scriptSig(txin, txout, txin_scriptPubKey)
    tx = create_signed_transaction(txin, txout, txin_scriptPubKey,
txin_scriptSig)

    return broadcast_transaction(tx)

if __name__ == '__main__':
    amount_to_spend = 0.0151
    txid_to_spend =
('f3de3c69f3e55293e03d30deb651a73d19e7b3e2c5215aee26026d4438668e26')
    utxo_index = 0
```

```
    print("my_address: ", my_address)
    print("my_public_key: ", my_public_key.hex())
    print("my_private_key: ", my_private_key.hex())

    txout_scriptPubKey = P2PKH_scriptPubKey()
    response = send_P2PKH_tx(amount_to_spend, txid_to_spend,
utxo_index, txout_scriptPubKey)

    print("response status code: " ,response.status_code, "response
reason: ", response.reason)
    print("response text: ", response.text)
```

مقدار پولی که میخواهیم به حسابمان برگردد را کمی کمتر از کل پول در نظر میگیریم و بقیه آن را transaction fee میگذاریم. برای این قسمت تابع P2PKH_scriptSig را کمی تغییر داده ایم.این تابع یک scriptSig را برای یک تراکنش P2PKH با دو امضا ایجاد میکند. ورودی تراکنش، خروجی تراکنش و scriptPubKey برای ورودی را به عنوان ورودی می گیرد. از تابع create_OP_CHECKSIG_signature برای ایجاد دو امضا، یکی برای هر کلید خصوصی، استفاده می‌کند و فهرستی حاوی [OP_0، signature1، signature2] را برمی‌گرداند.

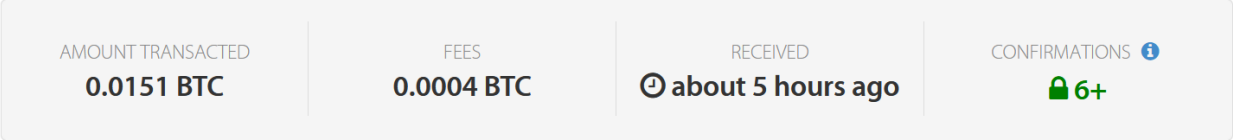
خروجی به صورت زیر میباشد.

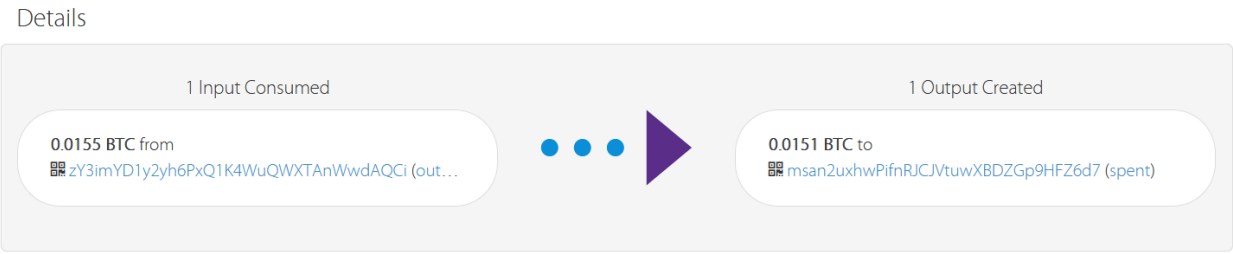
هش بدست آمده برابر است با:

"hash": "b72401a24a1814f85eb836dbb1b271d643e79241e7b52996271c567f6cc8ac7b"

تراکنش انجام شده در لینک زیر قابل مشاهده است.

https://live.blockcypher.com/btc-testnet/tx/b72401a24a1814f85eb836dbb1b271d643e79241e7b52996271c567f6cc8ac7b/





سوال 3:

بخش اول

کد این بخش در زیر آورده شده است.

```
import bitcoin.wallet
from bitcoin.core import COIN, b2lx, serialize, x, lx, b2x
from utils import *

bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret("92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGke
TwN7ef79ETNC") # Private key in WIF format
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
my_public_key = my_private_key.pub
my_address =
bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

private_key1=
bitcoin.wallet.CBitcoinSecret('922NDRTK861R1aAmbJ9PWFy53EFRXYQmEodoCRm
ngUgf5uAvtua')
public_key1 = private_key1.pub
```

```
private_key2 =
bitcoin.wallet.CBitcoinSecret('91jxGSKPdLsqnpDR1cdkC2Qg1xi6TuiUxxS5KYa
x5BUqhezx68B')
public_key2 = private_key2.pub

private_key3 =
bitcoin.wallet.CBitcoinSecret('92WRR9mH2pbvVkHHPhjcLcGBhXKvdSQEE3spUsT
BvG621T42khZ')
public_key3 = private_key3.pub

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

def P2PKH_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_P2PKH_tx(amount_to_spend, txid_to_spend, utxo_index,
txout_scriptPubKey):
    txin = create_txin(txid_to_spend, utxo_index)
    txout = create_txout(amount_to_spend, txout_scriptPubKey)

    txin_scriptPubKey = [OP_2, public_key1, public_key2, public_key3,
OP_3, OP_CHECKMULTISIG]
    txin_scriptSig = P2PKH_scriptSig(txin, txout, txin_scriptPubKey)
    tx = create_signed_transaction(txin, txout, txin_scriptPubKey,
txin_scriptSig)

    return broadcast_transaction(tx)

if __name__ == '__main__':
    amount_to_spend = 0.0151
    txid_to_spend =
('f3de3c69f3e55293e03d30deb651a73d19e7b3e2c5215aee26026d4438668e26')
    utxo_index = 0

    print("my_address: ", my_address)
    print("my_public_key: ", my_public_key.hex())
    print("my_private_key: ", my_private_key.hex())
```

```
    txout_scriptPubKey = P2PKH_scriptPubKey()
    response = send_P2PKH_tx(amount_to_spend, txid_to_spend,
utxo_index, txout_scriptPubKey)

    print("response status code: " ,response.status_code, "response
reason: ", response.reason)
    print("response text: ", response.text)
```

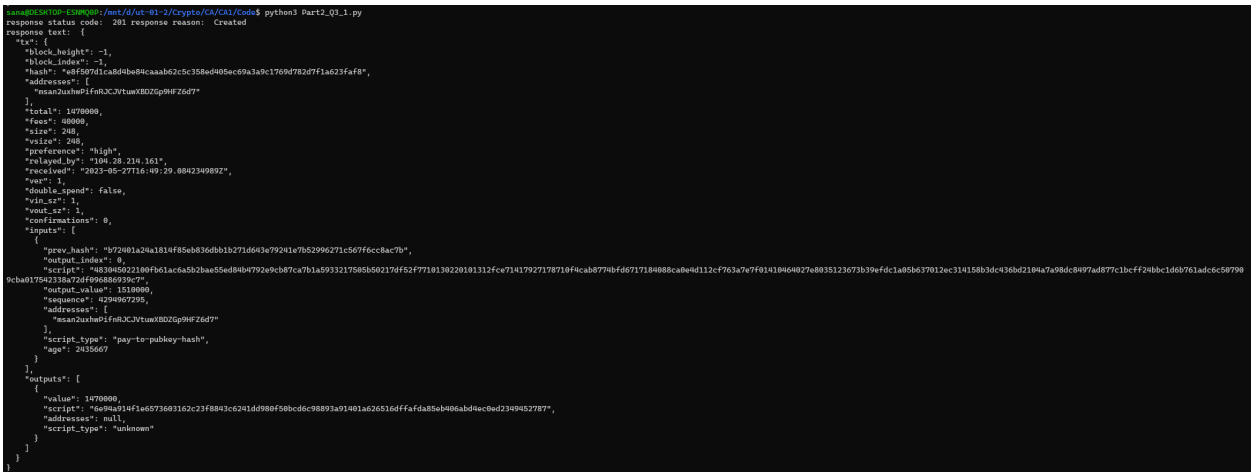
در اینجا در ابتدا دو عدد اول رندوم به دلخواه انتخاب میکنیم و جمع و تفریق آن را بدست میاوریم.

txout_scriptPubKey = [OP_2DUP, OP_SUB, OP_HASH160, Hash160((Sub).to_bytes(2, byteorder="little")), OP_EQUALVERIFY, OP_ADD, [OP_HASH160, Hash160((Sum).to_bytes(2, byteorder="little")), OP_EQUAL

در اینجا به دلیل اینکه جمع و تفریقمان هردو، دو بایت داشتند، مقدار 2 را برای تابع to_bytes در نظر میگریم. باقی قسمت های آن را باهم بررسی میکنیم.

- OP_2DUP: این اپکد دو مورد بالای پشته را کپی می کند.
- OP_SUB: این اپکد آیتم دوم روی پشته را از آیتم بالایی کم می کند.
- OP_ADD: این اپکد دو آیتم بالای پشته را اضافه می کند.
- OP_HASH160: این اپکد هش RIPEMD-160 آیتم بالای پشته را محاسبه می کند.
- OP_EQUALVERIFY: این اپکد دو آیتم بالای پشته را برای برابری با هم مقایسه می کند و اگر برابر نباشند، اجرای اسکریپت با شکست مواجه می شود.

هش بدست آمده با هش ارائه شده در scriptPubKey مقایسه می شود. اگر هش ها مطابقت داشته باشند، خروجی تراکنش می تواند خرج شود.

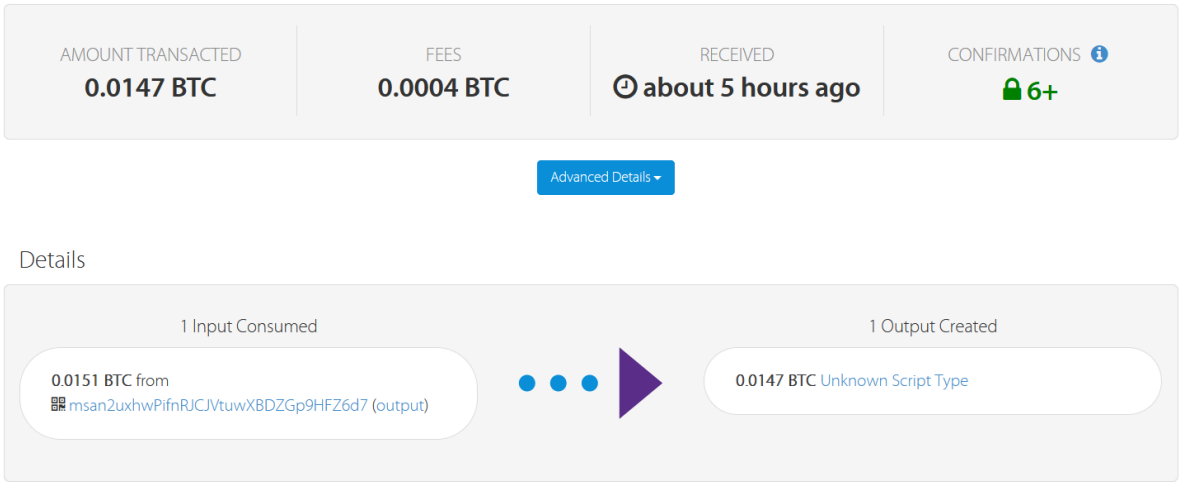


هش بدست آمده برابر است با:

"hash": "e8f507d1ca8d4be84caaab62c5c358ed405ec69a3a9c1769d782d7f1a623faf8"

تراکنش انجام شده در لینک زیر قابل مشاهده است.

https://live.blockcypher.com/btc-/testnet/tx/e8f507d1ca8d4be84caaab62c5c358ed405ec69a3a9c1769d782d7f1a623faf8



بخش دوم

کد این بخش به صورت زیر میباشد.

```
import bitcoin.wallet
from bitcoin.core import COIN, b2lx, serialize, x, lx, b2x
from utils import *

bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret("92pQfWUUM6zHkwiZSzK2QENzNswnH7BYtctSGke
TwN7ef79ETNC") # Private key in WIF format
XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
my_public_key = my_private_key.pub
my_address =
bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

def P2PKH_scriptPubKey():
    return [OP_DUP, OP_HASH160, Hash160(my_public_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_P2PKH_tx(amount_to_spend, txid_to_spend, utxo_index,
txout_scriptPubKey, prime_1, prime_2):
    txin = create_txin(txid_to_spend, utxo_index)
    txout = create_txout(amount_to_spend, txout_scriptPubKey)

    txin_scriptPubKey = [OP_2DUP, OP_SUB, OP_HASH160,
Hash160((Sub).to_bytes(2, byteorder="little")), OP_EQUALVERIFY,
OP_ADD,
                         OP_HASH160, Hash160((Sum).to_bytes(2,
byteorder="little")), OP_EQUAL]
    txin_scriptSig = [prime_2, prime_1]
    tx = create_signed_transaction(txin, txout, txin_scriptPubKey,
txin_scriptSig)

    return broadcast_transaction(tx)

if __name__ == '__main__':
    prime_1 = 4639
    prime_2 = 6269
    Sum = prime_1 + prime_2  #10908
    Sub = prime_2 - prime_1  #1630

    amount_to_spend = 0.0143
    txid_to_spend =
('e8f507d1ca8d4be84caaab62c5c358ed405ec69a3a9c1769d782d7f1a623faf8')
    utxo_index = 0

    txout_scriptPubKey = P2PKH_scriptPubKey()
    response = send_P2PKH_tx(amount_to_spend, txid_to_spend,
utxo_index, txout_scriptPubKey, prime_1, prime_2)

    print("response status code: " ,response.status_code, "response
reason: ", response.reason)
    print("response text: ", response.text)
```

حال در اینجا مقدار txin_scriptPubKey را تغییر میدهیم و همان مقدار txout_scriptPubKey بخش قبل قرار میدهیم. scriptPubKey ارائه شده در بخش قبل، برای خروجی تراکنش استفاده می شود، در حالی که scriptPubKey ارائه شده در این بخش برای ورودی های تراکنش استفاده می شود.

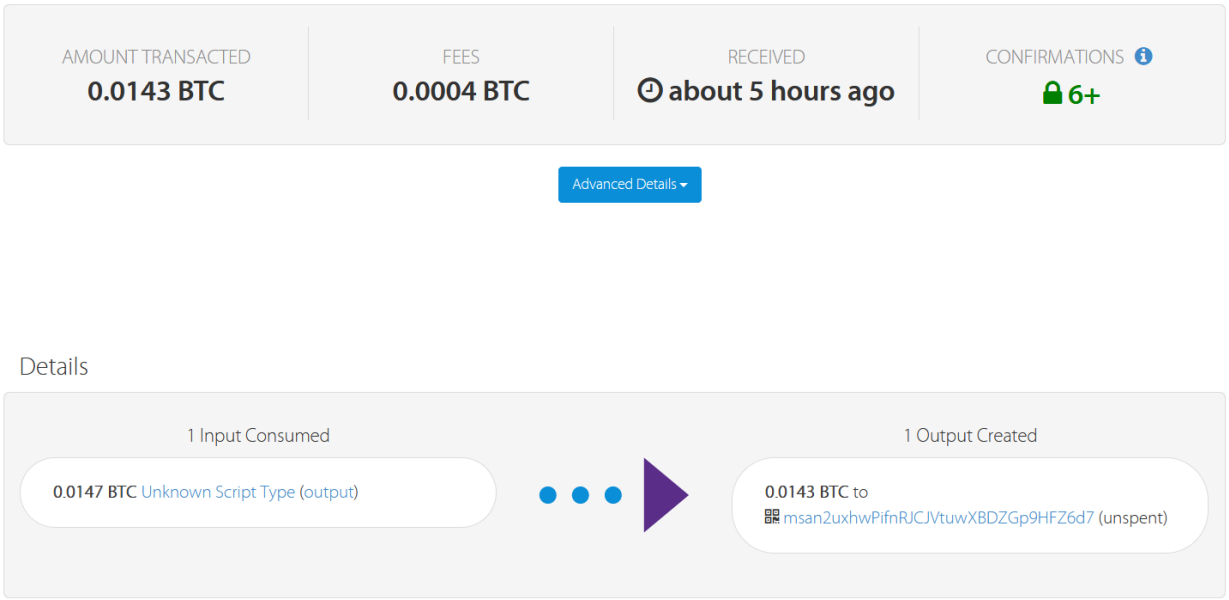

```
sana@DESKTOP-ESNMQ0P:/mnt/d/ut-01-2/Crypto/CA/CA1/Code$ python3 Part2_Q3_2.py
response status code:  201 response reason:  Created
response text:  {
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "fe87aa70502407a3e260262c2609b7165e9e6e6613382749f13606e88f598bc7",
    "addresses": [
      "msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7"
    ],
    "total": 1430000,
    "fees": 40000,
    "size": 91,
    "vsize": 91,
    "preference": "high",
    "relayed_by": "104.28.214.161",
    "received": "2023-05-27T17:08:54.21559294Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "e8f507d1ca8d4be84caaab62c5c358ed405ec69a3a9c1769d782d7f1a623faf8",
        "output_index": 0,
        "script": "027d18021f12",
        "output_value": 1470000,
        "sequence": 4294967295,
        "script_type": "unknown",
        "age": 2435669
      }
    ],
    "outputs": [
      {
        "value": 1430000,
        "script": "76a914845a9b96dfeac40f8cd6118afa717608067a086988ac",
        "addresses": [
          "msan2uxhwPifnRJCJVtuwXBDZGp9HFZ6d7"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```


هش بدست آمده برابر است با:

"hash": "fe87aa70502407a3e260262c2609b7165e9e6e6613382749f13606e88f598bc7"

تراکنش انجام شده در لینک زیر قابل مشاهده است.

https://live.blockcypher.com/btc-/testnet/tx/fe87aa70502407a3e260262c2609b7165e9e6e6613382749f13606e88f598bc7



قسمت سوم : استخراج بلوک

باید یک آدرس شبکه اصلی تولید کنیم. برای اینکار، از mainnet بهره میبریم. اما همانطور که گفتیم، برای mainnet، prefix های متفاوتی نسبت به testnet قرار میدهیم. برای کلید خصوصی مقدار 0x80 و برای کلید عمومی 0x00 میباشد.

آدرس شبکه و همچنین کلید عمومی و خصوصی به صورت زیر میباشد.

'Private Key: b'\x19\x19xew\xa7Z2\x01\xc1{\xe8j#]1\xfa/\x91\xa1q#\xaa$F\x8f4*?;\x87k

Public Key:
b'\x04J\xda!\xa1\x88\x08e\xc4C\x16*;\x7f;n\x19SR\xe6`\xccA\x94\x90\xea\xa1G\x15\x857,-3\xd3\xf9\xafz\xa7\x99\xf9\x13\x7fv\x10\xa9\xf1\x9d\xa3\x83\x7f\x15\xe0\x076\\\x8e\x85\'xa0\xd1\xb7\xf4Ov\xf0

'Private WIF: b'5J1LiPRjoLbvB6eAtETsDnfBQLDGTmVzvDBESc7bG51ZUy23jQ3

'Public P2PKH: b'1sanFc4uXxFW8T2AieRBmahU2gXiC4kys

به توضیح بخش های مختلف کد میپردازیم.

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

تابع P2PKH_scriptPubKey لیستی از اپکدهای اسکریپت بیت کوین را برمی گرداند که نشان دهنده اسکریپت Pay-to-Public-Hash (P2PKH) است. این اسکریپت برای مشخص کردن خروجی تراکنش بیت کوین استفاده می شود که می تواند توسط صاحب کلید عمومی مربوطه خرج شود.

```
def make_transaction(txid, utxo_index, block_reward, data):
    txout_scriptPubKey = P2PKH_scriptPubKey()

    txin = create_txin(txid, utxo_index)
    txout = create_txout(block_reward, txout_scriptPubKey)
    txin.scriptSig =  CScript([int(data, 16).to_bytes(len(data)//2, 'big')])

    return CMutableTransaction([txin], [txout])
```

تابع make_transaction یک تراکنش جدید بیت کوین را با یک ورودی و خروجی ایجاد می کند. ورودی یک UTXO است که توسط شناسه تراکنش (txid) و شاخص خروجی (utxo_index) مشخص شده است. خروجی یک خروجی P2PKH با پاداش بلوک و داده مشخص است. این تابع از توابع create_txin و create_txout از ماژول utils برای ایجاد اشیاء ورودی و خروجی به ترتیب استفاده می کند. همچنین ویژگی scriptSig شی ورودی را به یک اسکریپت بیت کوین که شامل داده های مشخص شده است، تنظیم می کند.

```
def target(bits):
    coefficient = bits[4:]
    exponent = bits[2:4]

    target = int(coefficient, 16) * (int('2', 16) ** (8 * (int(exponent, 16) - 3)))
    target_hex = str(format(target, 'x')).zfill(64)
    print("target: ", target_hex,"\n")
    return bytes.fromhex(target_hex)
```

تابع بعدی، مقدار هدف یک بلوک بیت کوین را با توجه به بیت های دشواری آن محاسبه می کند. تابع ضریب و توان را از بیت ها استخراج می کند و از آنها برای محاسبه مقدار هدف استفاده می کند. مقدار هدف یک عدد 256 بیتی است که نشان دهنده حداکثر مقدار هش است که یک هش بلاک معتبر می تواند داشته باشد. تابع مقدار هدف را به عنوان یک شی بایت برمی گرداند.

```
def get_block():
    block_size = len(b'\x01') + len(header) + len(block_body)
    block = MAGIC_NUMBER.to_bytes(4, "little") + struct.pack("<L", block_size) + header + b'\x01' + block_body
    print(b2x(block))
```

تابع get_block یک بلوک بیت کوین را با به هم پیوستن هدر و بدنه آن می سازد. این تابع اندازه بلوک را محاسبه می کند که شامل هدر بلوک، تعداد تراکنش ها و داده های تراکنش است. سپس بلوک را با الحاق شماره جادویی، اندازه بلوک، هدر، تعداد تراکنش ها و داده های تراکنش می سازد. تابع بلوک را به عنوان یک شی بایت برمی گرداند.

```python
if __name__ == '__main__':
    prev_hash_blocks = '00000000b34ff50a225ae6d5a127ae5a8ad8f3c96a0a3a3bdb41976b8a2eb450'

    version = 2
    block_reward = 6.25
    data = '810199435SanaSariNavaei'.encode('utf-8').hex()
    txid = (64*'0')
    utxo_index = int('0xFFFFFFFF', 16)
    bits = '0x1f010000'

    tx = make_transaction(txid, utxo_index, block_reward, data)
    block_body = b2x(tx.serialize())
    merkle_root = b2lx(sha256(sha256(tx.serialize()).digest()).digest())
    print("block body: ", block_body,"\n")
    print("merkle root: ", merkle_root,"\n")

    target = target(bits)
    get_time = int(time.time())
    partial_header = struct.pack("<L", version) + bytes.fromhex(prev_hash_blocks)[::-1] + bytes.fromhex(merkle_root)[::-1] + struct.pack('<LL', get_time, int(bits, 16))

    nonce = 0
    time_now = time.time()
    max_num =  0xFFFFFFFF
    while nonce <= max_num:
        header = partial_header + struct.pack('<L', nonce)
        block_hash = sha256(sha256(header).digest()).digest()
        if block_hash[::-1] < target:

            print ("nonce: ", nonce)
            print("Hash rate: ", nonce / (time.time() - time_now))
            print("Header: ", header)
            print("Block Hash: ", block_hash)
            print("Block Body: ", block_body)
            break

        nonce += 1
```

این تابع اصلی است که متغیرهای لازم را برای استخراج یک بلوک بیت کوین تنظیم می کند، یک تراکنش جدید ایجاد می کند، ریشه merkle را محاسبه می کند، و شروع به استخراج بلوک با افزایش nonce می کند تا زمانی که یک هش معتبر پیدا شود.

با اجرا کردن کد، به بخش زیر میرسیم.

```
sana@DESKTOP-ESNMQ0P:/mnt/d/ut-01-2/Crypto/CA/CA1/Code$ python3 Part3.py
block body:  0100000001000000000000000000000000000000000000000000000000000000000000000000000000ffffffff181738313031393934333553616e6153617269e6176616569ffffffff014
0be4025000000001976a9140990ff12f9ced28befb1dabd6a8f9ab5de172f5988ac00000000

merkle root:  5e507d1da03021236dface506eb3400f54b83e63251c282651731a4dc4f3c4fa

target:  0001000000000000000000000000000000000000000000000000000000000000

nonce:  21636
Hash rate:  255187.41256365768
Header:  b'\x02\x00\x00\x00P\xb4.\x8ak\x97A\xdb;:\nj\xc9\xf3\xd8\x8aZ\xae\'\xa1\xd5\xe6Z"\n\xf5O\xb3\x00\x00\x00\x00\xfa\xc4\xf3\xc4M\x1asQ&(\x1c%c>\xb8T\x0
f@\xb3nP\xce\xfam#!0\xa0\x1d}P^\xf1)sd\x00\x00\x00\x01\x1f\x84T\x00\x00'
Block Hash:  b'u{%O\xe3\x95\x12\x8b.s\xee\xb0\xdc[B\x95\xf2N\x02,\xba\xfb\xd0\x8f\x87\x134H\xd2\x89\x00\x00'
Block Body:  0100000001000000000000000000000000000000000000000000000000000000000000000000000000ffffffff181738313031393934333553616e6153617269e6176616569ffffffff014
0be4025000000001976a9140990ff12f9ced28befb1dabd6a8f9ab5de172f5988ac00000000
sana@DESKTOP-ESNMQ0P:/mnt/d/ut-01-2/Crypto/CA/CA1/Code$
```