علی جاویدان ۸۱۰۸۹۶۰۴۷

قسمت اول: تولید آدرس

سوال ۱)

```
Address: mq8F9UnWfVtVAhgpZN89qjRLy3n9eG4Ssw
Privkey: 91fvmHgYAUbWLvYFNh6XVf41v2FBEPBbfTWRbXzBZ4tJuBpLrSk
```

سوال ۲)

هدف از این سوال تغییر دادن کد سوال قبل بود به طوری که قادر به تولید Vanity Address با رشته ی آغازین مورد نظرخودمان باشیم. با این هدف کد قسمت قبل به نحوی تغییر کرد که تا زمان پیدا کردن چنین آدرسی به تولید و آزمایش آدرس های ساخته شده ادامه دهد.

سوال ۳)

```
Private key WIF: cVdte9ei2xsVjmZSPtyucG43YZgNkmKTqhwiUA8M4Fc3LdPJxPmZ
Public key: 028b9b0e596dbefb2055c0bfd7bb34b90d491030df81a2659ca5dbf941647e28ea
Native Address: tb1qxmt9xgewg6mxc4mvnzvrzu4f2v0gy782fydg0w
Segwit Hash: 36d653232e46b66c576c98983172a9531e8278ea
Segwit Version: p2wpkhv0
Created P2wpkhAddress from Segwit Hash and calculate address:
Native Address: tb1qxmt9xgewg6mxc4mvnzvrzu4f2v0gy782fydg0w
P2SH(P2WPKH): 2N8Z5t3GyPW1hSAEJZqQ1GUkZ9ofoGhgKPf
P2WSH of P2PK: tb1qy4kdfavhluvnhpwcqmqrd8x0ge2ynnsl7mv2mdmdskx4g3fc6ckq8f44jg
P2SH(P2WSH of P2PK): 2NC2DBZd3WfEF9cZcpBRDYxCTGCVCfPUf7Q
```

قسمت دوم: انجام تراکنش

کد کامل شده مورد استفاده این قسمت در مسیر Code/2/2/ در دسترس است.

قبل از هر چیز در خط

```
my_private_key = bitcoin.wallet.CBitcoinSecret(
        "91fvmHgYAUbWLvYFNh6XVf41v2FBEPBbfTWRbXzBZ4tJuBpLrSk")
```

private_key آدرس تولید شده در قسمت اول سوال ۱ را قرار دادیم، که در ادامه ی کد از آن public_key و آدرس P2PKH اسخراج می شود. و بعد از آن در خط

```
destination_address = bitcoin.wallet.CBitcoinAddress(
        'mq8F9UnWfVtVAhgpZN89qjRLy3n9eG4Ssw')
```

آدرس مقصدی که طی یک تراکنش قصد واریز به آن داریم قرار می گیرد.

یکی از توابعی که باید کامل می شد، در زیر آمده:

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

تابعی طوری کامل شده که مقداری را برمی گرداند که از آن در تابع create_OP_CHECKSIG_signature با گرفتن ()CScript از آن برای ساخت scriptPubKey استفاده می شود.
گفتنی است که همانطور که می دانیم scriptPubKey، script ی است که برای Lock کردن می نویسیم. در این اسکریپت مقدار آدرسی که در استک در اختیار داریم را با public_key HASH160 مقایسه کرده و در نهایت signiture ی که برای unlock کردن همراه با public_key ارائه شده با آن مطابق می دهیم و در صورت صحت، آن را تایید می کنیم.

دیگر تابعی که باید کامل می شد تابع

```
def P2PKH_scriptSig(txin, txout, txin_scriptPubKey):
    signature = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey, m
y_private_key)

    return [signature, my_public_key]
```

بود؛ که با برگرداندن مقدار [signature, my_public_key] در خروجی تابع و نگهداری آن در متغیر txin_scriptSig و جلوتر استفاده از آن برای ساخت تراکنش در create_signed_transaction مورد استفاده قرار می گیرد.
در طرف دیگر scriptSig را داریم. اسکریپتی می نویسیم signiture ی متناسب تراکنش ایجاد کند.

در نهایت در __main__ فایل transaction.py، txid تراکنشی که از خروجی آن قصد خرج کردن داریم به همراه index مربوطه خروجی، را قرار می دهیم. مقدار amount_to_send را نیز معادل با مقدار ورودی تراکنشی که قصد ساختن آن را داریم منهای کارمزد تراکنشی که در نظر گرفتیم، می گذاریم.

تمامی تراکنش هایی که در ادامه انجام شده در آدرس زیر قابل مشاهده است.

```
mq8F9UnWfVtVAhgpZN89qjRLy3n9eG4Ssw
```

سوال ۱)

تراکنشی که با کد کامل شده ی ارائه شده انجام شده با شناسه تراکنش زیر قابل پیگیری است. در این تراکنش از آدرس خودمان به خودمان واریزی داشتیم.

```
f02f5b00e424ae7a6e1956f7e5cb7e72c4cd4870d1902f87486174d0c2755e7d
```

مطابق آنچه که در سوال خواسته شده برای داشتن دو خروجی در تراکنش می بایست مشابه آنچه در زیر به آن اشاره شده:

```
txout = create_txout(amount_to_send / n, txout_scriptPubKey)
tx = CMutableTransaction([txin], [txout]*n)
```

دو خروجی با create_txout خروجی با این تفاوت که از نسخه ی ویرایش شده از create_txout استفاده کنیم که خروجی ای غیر قابل خرج/قابل خرج توسط هر شخص بسازد. و این txout های ساخته شده را با CMutableTransaction در قالب یک تراکنش در بیاوریم.

سوال ۲)

در این قسمت از خروجی نوع Multisig استفاده شد.

واریز از آدرس تولید شده در سوال ۱ قسمت اول به آدرس مشترک سه آدرس تولید شده زیر انجام شده است.

```
Address: myeQvK3B5mJjnWUkC9c92TNW89uaonjLZM
Privkey: 92UWGcga2spQvdxbnRfi8CSwqkoU5p5X6PZmNw8TLpiT3GUVxGq

Address: mmUTyhtrm1A2Ki1bKxYp6oQG667LreqA5t
Privkey: 91mTV3ZqxgKGpwfTKQy2sMnnWKJHYaZNiSf5s2X6NzsiRMFzx7e

Address: mjgLEjQt5kVgA8aHMxw9u1scuetfRBcvpk
Privkey: 92r3GRWcM4vgWDS9TNCTWFdKXsn7APDeBKvvec1kcLwYXj128Ur
```

کد مربوط به واریر در مسیر Code/2/2.2/transaction قابل دسترس است. از طرفی کد مربوط به بازگشت دادن وجه ارسالی از آدرس مشترک به آدرس تولید شده اولیه که واریز از آن انجام شده در مسیر Code/2/2.2/spend قرار دارد.

تغییراتی که لازم بود در کد مربوط به فایل transaction.py انجام شود، بازنویسی P2PKH_scriptPubKey به شکلی بود که اسکریپت PubKey به شکل زیر تغییر کند.

```
def P2PKH_scriptPubKey_(destination_pubkey):
    return [my_public_key, OP_CHECKSIGVERIFY, OP_2, destination_pubkey[0], des
tination_pubkey[1], destination_pubkey[2], OP_3, OP_CHECKMULTISIG]
```

که در آن یک Lock ۲ از ۳ تایی روی خروجی تراکنش گذاشته می شود. برای خرج کردن نیز به همین ترتیب کد scriptSig به شکل زیر بازنویسی می شود. دقت شود `[my_public_key, OP_CHECKSIGVERIFY,` که در ابتدای script آمده به نحوی نقش شاهد را در تراکنش بازی می کند.

```
def multisig_scriptSig(txin, txout, txin_scriptPubKey):
    bank_sig = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
my_private_key)
    cust1_sig = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
destination_privkey[0])
    cust2_sig = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
destination_privkey[1])
    cust3_sig = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
destination_privkey[2])

    return [OP_0, cust1_sig, cust2_sig, bank_sig]
```

که در آن sig مربوط به دو عدد از آدرس های مقصد به علائه ی امضای گواه که در اینجا به آن bank_sig اطلاق شده، بازگردانده می شود.

شناسه تراکنش واریز:

```
852c9e863ddee205720ab7031d5e562692ddc07fc0e96649b9a47c2ed9ecb208
```