

بسم الله الرحمن الرحيم

استاد: محمد علی مداح علی

درس: بلاک چین

شرح محتوا: گزارش تمرین عملی سری ۲

نگارنده: امیرحسین رستمی

شماره دانشجویی: ۹۶۱۰۱۶۳۵

دانشگاه صنعتی شریف

پاییز ۹۹

سوال سوم:

برای طراحی اسکرپت برای هندل کردن شرایط مطرحی صورت سوال از دستور OP_IF استفاده می کنیم. از جمله دستورات مهمی که جهت کاهش تعداد خطوط اسکرپت استفاده کردیم دستور OP_DEPTH است که بیانگر تعداد اعضا موجود در استک است و به کمک تعداد آیتم های موجود در استک (push شده توسط unlock script) متوجه می شویم که در کدام condition قرار داریم و طبق آن به بررسی احراز های لازم می پردازیم.

نکته مهمی که در استفاده از OP_CHECKMULTISIG وجود دارد این است که هنگام خواندن از استک با فرض اینکه به تعداد m از n کلید نیاز است، داریم که باید اولاً m تا signature قرار بدهیم سپس عدد m را قرار بدهیم و سپس n تا کلید عمومی قرار بدهیم و روی آن نیز عدد n را بگذاریم، اما این کافی نیست و به خاطر وجود یک باگ در اجرا داریم که یک عدد دیگر از استک pop می شود لذا لازم است که در ابتدا هم به x عه دلخواه در استک تزریق کنیم. یعنی در اصل برای احراز m عدد امضا توسط n عدد کلید عمومی نیاز است تا $m+n+2+1$ عدد آیتم در استک قرار بدهیم.

ساختاری مشابه تصویر زیر: (برای حالتی که $m=2$ و $n=3$ باشد).

3
<Public Key Charlie>
<Public Key Bob>
<Public Key Alice>
2
<Signature Charlie>
<Signature Bob>
0

آن x که پیشتر مطرح کردیم در تصویر فوق توسط OP_0 در ابتدای استک تزریق می شود.

توجه: در این سوال برای هر کدام از سهامداران (عطا فراز و ۵ سهام دار دیگر) جفت کلید های private و public تولید کردیم و همگی در فایل ex3_values قرار دارند که به صورت library در ابتدا import شده اند.

جفت کلید های (عمومی، خصوصی) به صورت زیر است:

```
1 from bitcoin.wallet import CBitcoinSecret, P2PKHBitcoinAddress
2
3
4 def giveMePrivatePublicAddress(pr_key):
5     private_key = CBitcoinSecret(pr_key)
6
7     public_key = private_key.pub
8     address = P2PKHBitcoinAddress.from_pubkey(public_key)
9     return private_key, public_key, address
10
11
12 Ata_private_key, Ata_public_key, Ata_address = giveMePrivatePublicAddress(
13     'cQJ7dKepXM5AtKDqrMbm3hPaNJTcf7ki9iyrrn8Su2r7c8q2GCz3')
14 Faraz_private_key, Faraz_public_key, Faraz_address = giveMePrivatePublicAddress(
15     'cUVgwKxGyds3m2h6v5VPhin7vRWQXp2HAoy92yHXeHzDvQtrAHBo')
16 sh1_private_key, sh1_public_key, sh1_address = giveMePrivatePublicAddress(
17     'cQ4T7p66rM9PdKnAecqkNzhCeMs1CuDRBYqGWgQsaEjVHqinJJQv')
18 sh2_private_key, sh2_public_key, sh2_address = giveMePrivatePublicAddress(
19     'cTgFpkclDXmdguHXMq99855cNvtWpqcMbCMQpXNwYZ5aq8wD92X')
20 sh3_private_key, sh3_public_key, sh3_address = giveMePrivatePublicAddress(
21     'cRHeNQRmHxkAyyHrPM2wvvZkxhXaLFHBAhmSzfla76fWD3G2pkjK')
22 sh4_private_key, sh4_public_key, sh4_address = giveMePrivatePublicAddress(
23     'cVjv2FKr8RTX3CuwYWyAadtFQRnVqx5msn93aYAXH8wMVCSp8J1W')
24 sh5_private_key, sh5_public_key, sh5_address = giveMePrivatePublicAddress(
25     'cQGgG48fg9APGeJ1ABi2CcX8hD6kLpukKZhJheAYHSmgKBNf51b')
26
```

سوال چهارم:

در این سوال جهت قرار دادن String دلخواه داخل بلاکچین از اسکریپت زیر استفاده می کنیم:

```
13 save_message_scriptPubKey = [  
14     OP_RETURN, str.encode("Rostami96101635")  
15 ]
```

توجه: رشته جهت قراردادی روی شبکه نیاز دارد تا encode شود (به صورت byte) و قرار دادن خام آن ارسال را دچار اشکال می کند.

با جست و جو کردن Hash تراکنش حاصل در blockCypher محتوای نوشته شده را ملاحظه می کنیم:

⚠ Data Embedded in Transaction with Unknown Protocol (what's this?)

String: Rostami96101635

Hex: 526f7374616d693936313031363335

سوال پنجم:

در این سوال از دستور CLTV استفاده کردیم (ClockLockTimeVerify) و داریم که خروجی تراکنش تا هنگامی که زمان موردنظر نرسیده است، قابل استفاده نیست.

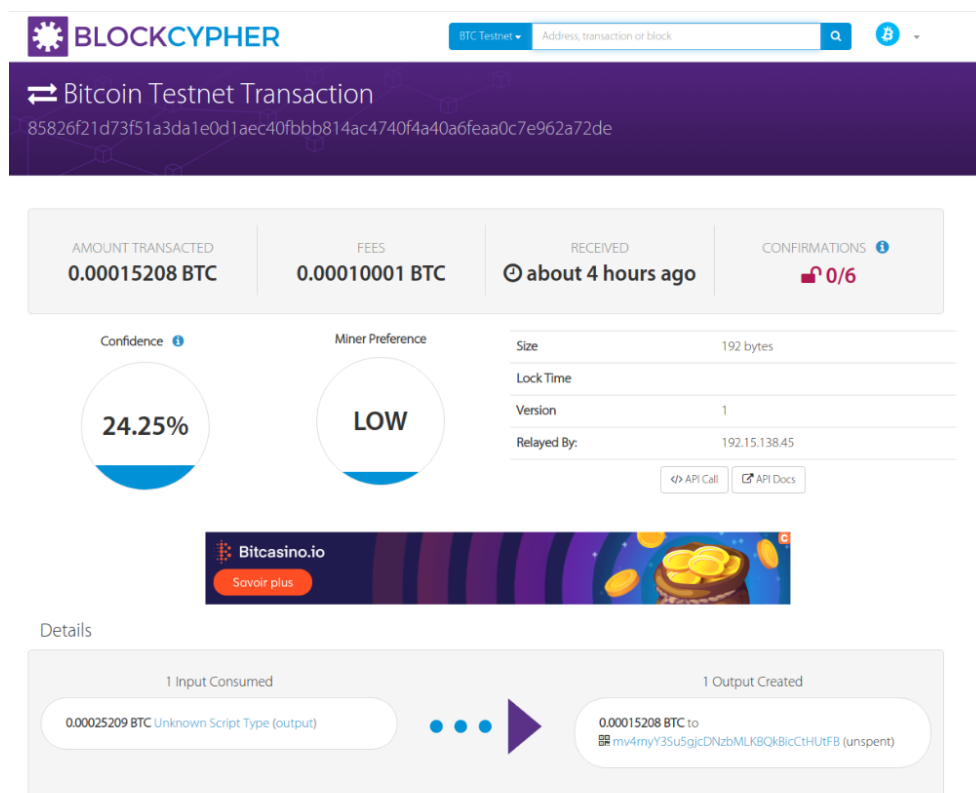
اگر

- زمان زیر 500M باشد -> "بلاک" مدنظر را نشان می دهد.
- زمان بالاتر از 500M باشد -> تاریخ مشخصی را نشان می دهد.

جهت تعیین زمان از سایت <https://www.unixtimestamp.com/index.php> استفاده کرده ام و همانطور که می دانید این عدد برابر است با تعداد ثانیه های گذشته شده از ابتدای سال 1970 میلادی است.

توجه کنید که تراکنش قسمت a در شبکه mine گشته و confirm می گردد اما تراکنش b به صورت unconfirmed باقی می ماند.

وضعیت تراکنش b:



توجه: در این سوال برای Hamed جفت کلید public و private تولید کردیم به همراه متغیر timelock در فایل ex5_values قرار داده ایم که به صورت library در ابتدا import شده اند.

اسکریپت این قسمت از دو تکه تشکیل شده است:

1- احراز اینکه زمان بازگشایی فرارسیده است یا خیر

2- احراز هویت مصرف کننده

```
14 txout_scriptPubKey = [  
15     timelock, # time check(whether it is reached or not).  
16     OP_CHECKLOCKTIMEVERIFY,  
17     OP_DROP,  
18     OP_DUP, # simple Hamed sigAuthen :)  
19     OP_HASH160,  
20     Hamed_address,  
21     OP_EQUALVERIFY,  
22     OP_CHECKSIG  
23 ]  
24  
25  
26
```

سوال ششم:

همانند سوال قبل در این سوال جهت احراز رخ داد expiration، از CLTV استفاده می کنیم. توجه کنید که داریم که اسکریپت این سوال از دو بخش تشکیل شده است:

```
Q6a_txout_scriptPubKey = [  
    2, # sig check  
    Hamed_public_key,  
    my_public_key,  
    2,  
    OP_CHECKMULTISIG,  
  
    OP_IF, OP_TRUE, OP_ELSE, # time expiration check  
    expiration_time,  
    OP_CHECKLOCKTIMEVERIFY,  
    OP_ENDIF  
  
]
```

ابتدا unlock script احراز اصالت می گردد.

- اگر اوکی بود تراکنش قابلیت redeem شدن می یابد.
- اگر اوکی نبود سراغ بررسی رخداد expiration می رویم:
i. اگر rent period is over رخ داده بود:
تراکنش قابلیت redeem شدن می یابد.
ii. اگر rent period isn't over:
تراکنش قابل redeem شدن ندارد.

توجه: در این سوال برای Hamed جفت کلید private و public تولید کردیم و در کنار متغیر expiration_time در فایل ex6_values قرار داده ایم که به صورت library در ابتدا import شده اند.

سوال هفتم:

در این سوال تابع `send_from_P2PKH_transaction()` را با پیاده سازی خود در فایل `ex7.py` قرار داده ایم، در این تابع که خواسته سوال را اجرا می کند مراحل زیر را پله پله انجام می دهیم:

- ساخت `txin` مشابه سوالات قبلی انجام می گیرد.
- یک عدد `txout` با `amount_to_send` ای برابر با $\frac{1}{3}$ کل مبلغ قابل انتقال ایجاد می کنیم.
- به تعداد 3 عدد از `txout` تولیدی مرحله قبل در `tx` قرار می دهیم.

سه مرحله ذکر شده در کد به صورت سه خط کد زیر اجرا می گردد.

```
32 txin = create_txin(txid_to_spend, utxo_index)
33 txout = create_txout(amount_to_send / 3, txout_scriptPubKey)
34 tx = CMutableTransaction([txin], [txout] * 3)
```

- مشابه ایجاد تراکنش با یک عدد ورودی به یک عدد خروجی، به تزریق بخش های `scriptPubKey` و `sigHash` به `txin` می پردازیم.

این مرحله به صورت قطعه کد زیر انجام می گیرد:

```
txin_scriptPubKey = sender_address.to_scriptPubKey()
sighash = SignatureHash(txin_scriptPubKey, tx, 0, SIGHASH_ALL)
txin.scriptSig = CScript([sender_private_key.sign(sighash) + bytes([SIGHASH_ALL]),
                           sender_private_key.pub])
```

- در نهایت صحت تراکنش به وجود آمده شده را با تابع `VerifyScript` بررسی کرده و نتیجه را در شبکه `broadcast` می کنیم.

سوال هشتم:

در این سوال تابع `send_from_P2PKH_transaction()` را با پیاده سازی خود در فایل `ex8.py` قرار داده ایم، پیاده سازی کاملاً مشابه با سوال هفتم است با این تفاوت که کار هایی که در سوال قبلی برای `txout` کردیم را اینجا برای `txin` انجام می دهیم، در این تابع که خواسته سوال را اجرا می کند مراحل زیر را پله پله انجام می دهیم:

- یک آرایه سه تایی از `txin` ایجاد می کنیم.
- یک عدد `txout` با `amount_to_send` ای برابر با 3 مبلغ هر `txin` ایجاد می کنیم.
- به کمک دو مورد بالا `tx` را آماده می کنیم.

سه مرحله ذکر شده در کد به صورت سه خط کد زیر اجرا می گردد.

```
36 txout = create_txout(amount_to_send * 3, txout_scriptPubKey)
37 tx = CMutableTransaction(txin, [txout])
38 txin_scriptPubKey = sender_address.to_scriptPubKey()
```

- برای هر سه `txin` موجود در آرایه `txin` به تزریق بخش های `scriptPubKey` و `sigHash` به `txin` می پردازیم. این مرحله به صورت قطعه کد زیر انجام می گیرد:

```
sighash = []
for i in range(3):
    sighash.append(SignatureHash(txin_scriptPubKey, tx, i, SIGHASH_ALL))

for i in range(3):
    txin[i].scriptSig = CScript(
        [
            sender_private_key.sign(sighash[i]) + bytes([SIGHASH_ALL]),
            sender_private_key.pub
        ]
    )
```

- در نهایت صحت تراکنش به وجود آمده شده را با تابع `VerifyScript` بررسی کرده و نتیجه را در شبکه broadcast می کنیم. (توجه کنید که `Verify` کردن `script` باید برای هر کدام از `txin` ها انجام گیرد).

```
for i in range(3):
    VerifyScript(txin[i].scriptSig, txin_scriptPubKey, tx, i, (SCRIPT_VERIFY_P2SH,))

return broadcast_transaction(tx, network)
```


سوال نهم:

• fileverify.py

در حالت fileverify.py ابتدا به کمک تابع زیر مقدار Hash فایل data.hex موجود را خوانده و Hash آن را محاسبه می کنیم:

```
14 def sha256sum(filename):
15     h = hashlib.sha256()
16     with open(filename, 'rb', buffering=0) as f:
17         for b in iter(lambda: f.read(128 * 1024), b''):
18             h.update(b)
19     return h.digest()
```

حال با در نظر گرفتن این محتوای Hash به دست آمده به عنوان private_key از روی آن کلید عمومی و آدرس کیف پول را محاسبه می کنیم:

```
39 seckey = CBitcoinSecret.from_secret_bytes(sha256sum(filename))
40 address = P2PKHBitcoinAddress.from_pubkey(seckey.pub)
41 ##
42 your_address = address
```

باقی کار کاملاً مشابه با issue کردن یک تراکنش ساده است.

• multifileverify.py

در این قسمت از merkleTree استفاده می کنیم و به root آن برسیم تا بتوانیم هر لحظه احرازهای لازم را انجام بدهیم. (توجه کنید که واضحاً چون سوال اشاره به میزان order زمان/پیغام نکرده است می توان از سناریوهای دیگری استفاده کرد مثلاً concat کردن همه فایل ها و استفاده از Hash آن ...)، همانطور که به خاطر دارید این کار را در تمرین قبل مفصلاً انجام داده ایم و لذا کد تولید merkleTree از تمرین قبل را در فایلی به نام merkle_tree.py آورده ایم و در این حالت از آن استفاده می کنیم.

سوال دهم:

خواسته های سوال را به کمک OP_IF ارضا می کنیم.

- سناریو اول:

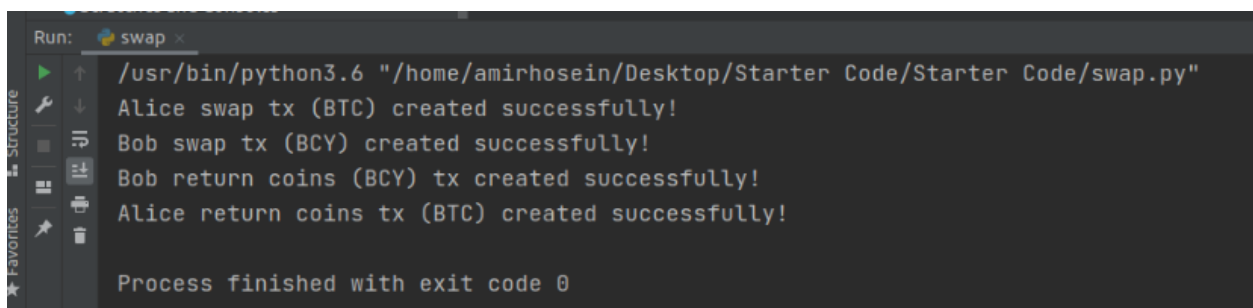
اگر شرط if درست بود، در ابتدا HASH160 مقدار X ورودی با secret مقایسه می شود که اگر درست باشد، امضای نفر دوم با key_public او مقایسه شده و verify می شود.

- سناریو دوم:

اگر شرط if درست نبود، به امضای هردو نفر نیاز است. بنابراین از CHECKMULTISIG استفاده می کنیم. همانطور که پیشتر گفتیم به علت باگ اجرایی CHECKMULTISIG نیاز است تا یک عدد متغیر به دلخواه (که در اینجا OP_0 است) به ابتدای script اضافه شود.

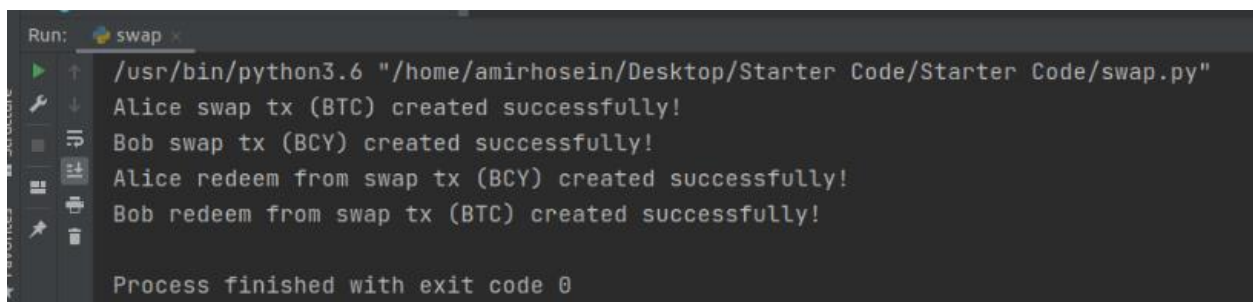
خروجی های در حالت به شرح زیر گردید:

- Alice Redeems = False



```
Run: swap x
/usr/bin/python3.6 "/home/amirhosein/Desktop/Starter Code/Starter Code/swap.py"
Alice swap tx (BTC) created successfully!
Bob swap tx (BCY) created successfully!
Bob return coins (BCY) tx created successfully!
Alice return coins tx (BTC) created successfully!
Process finished with exit code 0
```

- Alice Redeems = True



```
Run: swap x
/usr/bin/python3.6 "/home/amirhosein/Desktop/Starter Code/Starter Code/swap.py"
Alice swap tx (BTC) created successfully!
Bob swap tx (BCY) created successfully!
Alice redeem from swap tx (BCY) created successfully!
Bob redeem from swap tx (BTC) created successfully!
Process finished with exit code 0
```