به نام خدا

دانشگاه تهران

پردیس دانشکده‌های فنی

دانشکده مهندسی برق و کامپیوتر

تمرین کامپیوتری اول

درس ارزهای رمزگذاری شده

نسا عباسی مقدم ۸۱۰۱۹۹۴۵۷

خرداد ۱۴۰۲

قسمت اول: تولید آدرس

سوال اول.

برای این سوال از کتابخانه های hashlib, secrets, ecdsa و base58 استفاده می‌کنیم. در ابتدا برای تولید Private Key، یک عدد ۲۵۶ بیتی (برابر ۳۲ بایت) رندوم تولید می‌کنیم. پس از اعمال ECDSA بر روی آن، مقدار Public Key را می‌توان بدست آورد.

فرمت WIF: با استفاده از تابع b58encode_check روی کلید (new_key = prefix + private_key) می‌توان آنرا به فرمت WIF تبدیل کرد. در این تابع، هش SHA256 دو مرتبه روی کلید اعمال شده و ۴ بایت اول آن بعنوان check_sum ذخیره می‌شود و در نهایت نتیجه new_key + check_sum را که با استفاده از تابع b58encode، اینکود شده است، ریترن می‌شود.

فرمت PTPKH: تابع درهم ساز SHA256 را روی public_key اعمال کرده و روی نتیجه بدست آمده تابع درهم ساز RIPEMD160 را اعمال می‌کنیم و در نهایت نتیجه بدست آمده را به فرمت WIF تبدیل می‌کنیم.

```
TESTNET_PRIVATE = b'\xef'
TESTNET_PUBLIC = b'\x6f'

def make_private_key():
    private_key = secrets.token_bytes(32)
    return private_key

def wif(key, prefix):
    new_key = prefix + key
    base58_checking = base58.b58encode_check(new_key)
    return base58_checking

def make_public_key(private_key):
    public_key = ecdsa.SigningKey.from_string(private_key, curve=ecdsa.SECP256k1).verifying_key
    return public_key

def pay_to_public_key_hash(public_key):
    hash_byte = hashlib.sha256(public_key).digest()
    new_hash = hashlib.new( "ripemd160", hash_byte).digest()
    print(new_hash)
    return wif(new_hash, TESTNET_PUBLIC)


private_key = make_private_key()
final_private_key = wif(private_key, TESTNET_PRIVATE)
public_key = make_public_key(private_key)
final_public_key = pay_to_public_key_hash(b'\04' + public_key.to_string())

print(f'''Private_key = {final_private_key} \nPublic key = {final_public_key}''')
```

تفاوت آدرس شبکه اصلی و شبکه تست (Testnet)، در Network Byte آن‌ها است. با توجه به لینک مقدار prefix برای شبکه تست در pubic_key برابر عدد هگز 6F و برای private_key برابر عدد هگز EF است. بنابراین وقتی که آدرس به base58 تبدیل می‌شود، اولین کاراکتر برای کلید عمومی همواره ۱ خواهد بود که در شبکه تست این کاراکتر می‌تواند حرف m و یا n باشد؛ همچنین برای کلید خصوصی اولین کاراکتر برابر عدد 9 است.

111	6F	Testnet pubkey hash	m *or* n
239	EF	Testnet Private key (WIF, uncompressed pubkey)	9

خروجی:

```
Private_key WIF = b'92D8GhkunLovENddecVmUhTrWZLM7GxPcGXsfH7oE8PpyWKPyXp'
Public key P2PKH= b'mohzxsM62us9e8vPf18Hg2qC3VysLgbooG'
Private_key = b'R7g\xef\x17\x7f\x94\xfa\x0cy\xed\xb0\xb2\x03\x9d\xbd\xa3\x00-U\xf5\xcdi\xac\xc5s\xf0N\x0e\x80d\xd0'
Public key = b'\xf4\x15\xef#\xc7\x01v\x1f\xd9(\xc4\xa1\xbe\xd7#"nU\x88\xdeiU-h\xf2\n\xb2He\x15\xac\xc0\xd7\xfe\xef\xbe \xccu\xa8\xd5\x82\xa1\xa5Yvm\xa
b`\xde _\x10\xe8v\x94*\xa9\x16n\x051\xa6wt'
```

سوال دوم.

در این قسمت باید vanity address تولید کنیم. به این منظور باید private_key های مختلف را بررسی کنیم تا در نهایت، آدرسی تولید شود که ابتدای آن با پیشوند انتخابی ما یکسان باشد. تیکه کدی که در پایین قرار داده شده است را به کد قسمت قبل اضافه می‌کنیم.

```
def vanity_address(start_chars):
    while True:
        private_key = make_private_key()
        final_private_key = wif(private_key, TESTNET_PRIVATE)
        public_key = make_public_key(private_key)
        hash, final_public_key = pay_to_public_key_hash(b'\04' + public_key.to_string())
        if final_public_key[1:4] == start_chars:
            print(hash)
            return final_public_key, final_private_key, public_key, private_key
```

خروجی:

```
Private_key WIF = b'92hbwcFP1VgHXhunDzcB9UjwgcgnX8EDuM2iyivPc24uvxPxZ8S'
Public key P2PKH= b'mnes1FLi9YAbmA8NmqaDrSDA4REvixfUJz'
Private_key = b'\x92\xe0\x8b\x83\x8d\x13[\x17x\x87;0\xd9\\C\x89\xcf\xe9\xe2<\xbe]\xc8\x19\x01\xcc\x92\xef\xf4\xe7\xc0R'
Public key = b'9\xc2"\x1b\x12\x99Tq/{\x1e\x8ex\xc7\xc1\xe8I\xce\xe3=\x95\x08\xe7X\xc6\xd3\xc8/\xe6\x93\x93;\xd5\x95\x1cu\xb7\xe7e\xaf\xd6\x1c\xe5\xfd\
x88\\\xc3\xbc`\xefNy;D\xae\x07\xd4(\x8a\xb4\x83\xe8\t\xee'
```

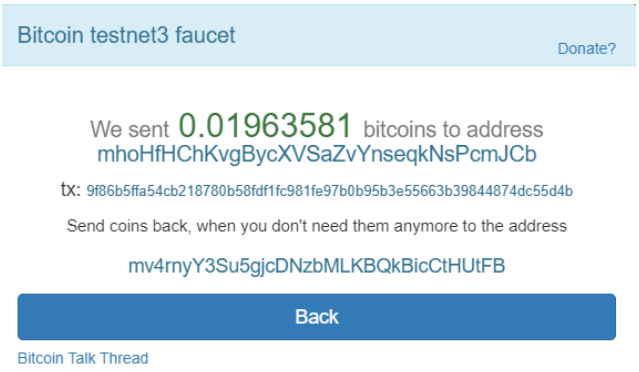
قسمت دوم: انجام تراکنش

۱. سوال اول. خروجی اول غیر قابل خرج و خروجی دوم قابل خرج توسط هرکس

برای خروجی‌ای که توسط هیچ‌کس قابل خرج نباشد، از txout1_scriptPubKey = [OP_TRUE] و برای خروجی ای که توسط هرکس قابل خرج شدن باشد، از txout2_scriptPubKey = [OP_FALSE] استفاده می‌کنیم.
با ران کردن کد قسمت اول، private_key و public_key را بدست می‌آوریم:

Public_key = mhoHfHChKvgBycXVSaZvYnseqkNsPcmJCb
Private_Key = 91gpQs5JtC2XdHEL7WDGjMeyWgmrCJMEZxY7kKYkaGY6LXE1Mw3

سپس public_key را وارد لینک می‌کنیم و مقدار بیت کوین رو به رو را به ما می‌دهد، که از این مقدار 0.0002 را به خروجی غیرقابل خرج و 0.0001 را به خروجی قابل خرج تخصیص می‌دهیم.



در تابع P2PKH_scriptPubKey، ابتدا مقدار سر صف، که در اینجا مقدار public_key است، duplicate می‌شود. سپس از این مقدار hash گرفته شده و در استک قرار می‌گیرد. سپس مقدار Hash160(my_public_key) روی استک قرار گرفته و مقدار آن با هش قبلی مقایسه می‌شود. در صورت برابری، مقدار true و در غیراینصورت مقدار false برگردانده می‌شود. در صورت true بودن، ادامه استک بررسی می‌شود. در ادامه CHECKSIG_OP قرار دارد که امضای ما که در ته استک قرار دارد را با توجه به public_key مقایسه می کند.
کد بخش اول سوال:

```
bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret("91keE45J9TQVVxrwMA4pQwd75Cf9JoRYAfBFXHmUKPSjCE8RwA4")
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, txout1, txout2, txin_scriptPubKey):
    signature = create_OP_CHECKSIG_signature(txin, [txout1, txout2],
txin_scriptPubKey, my_private_key)
    return [signature, my_public_key]

def P2PKH_txin_scriptPubKey():
    return [OP_DUP, OP_HASH160, Hash160(my_public_key),OP_EQUALVERIFY ,OP_CHECKSIG]
```

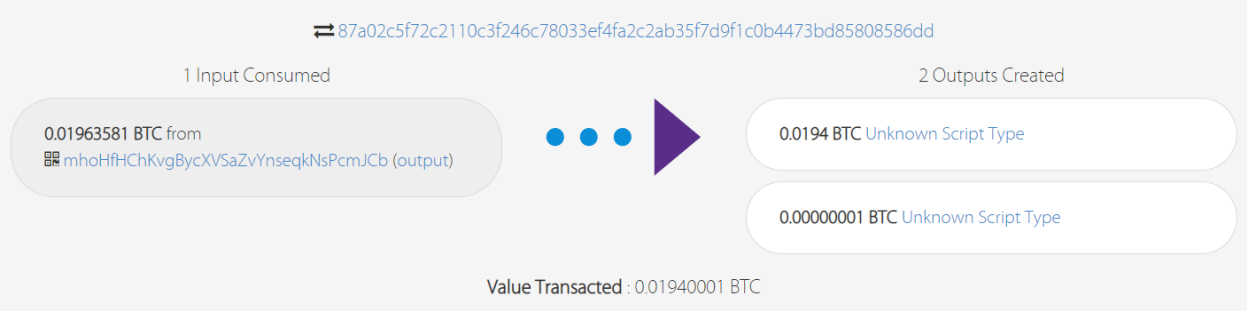
```
def send_from_P2PKH_transaction(amount_to_send1, amount_to_send2, txid_to_spend,
utxo_index):
    txout1_scriptPubKey = [OP_TRUE]
    txout2_scriptPubKey = [OP_FALSE]
    txout1 = create_txout(amount_to_send1, txout1_scriptPubKey)
    txout2 = create_txout(amount_to_send2, txout2_scriptPubKey)
    txin_scriptPubKey = P2PKH_scriptPubKey()
    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__':
    amount_to_send1 = 0.0194
    amount_to_send2 = 0.00000001
    txid_to_spend =
('9f86b5ffa54cb218780b58fdf1fc981fe97b0b95b3e55663b39844874dc55d4b')
    utxo_index = 1
    print("Address = ", my_address)
    print("Public_key = ", my_public_key.hex())
    print("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.reason)
    print(response.text)
```

دو مقدار تعریف می کنیم به طوری که مجموع آنها از مقدار موجودی کلی کمتر باشد و مقداری برای transaction fee باقی بماند تا transaction ما توسط دیگران confirm شود.

هش تراکنش:

```
87a02c5f72c2110c3f246c78033ef4fa2c2ab35f7d9f1c0b4473bd85808586dd
```

اطلاعات این تراکنش در لینک موجود است.



```
root@Nesa:/mnt/d/Crypto/CA/ItC-CA1/starter# python3 Q2.P1.1.py
Address =  mhoHfHChKvgBycXVSaZvYnseqkNsPcmJCb
Public_key =  047208efe5a9d4735c1f6465dbff9bcba68fec0dc894a4e566e264098d798d89509119fd7c584f11ea2a48c5371c0a0dceef2d653e4149d0405095cb72af14e9bf
Private_key =  0d660a204d7856a7958289b95edf4099a0f2058d171c551872c5a0f533177559
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "87a02c5f72c2110c3f246c78033ef4fa2c2ab35f7d9f1c0b4473bd85808586dd",
    "addresses": [
      "mhoHfHChKvgBycXVSaZvYnseqkNsPcmJCb"
    ],
    "total": 1940001,
    "fees": 23580,
    "size": 208,
    "vsize": 208,
    "preference": "high",
1f6465dbff9bcba68fec0dc894a4e566e264098d798d89509119fd7c584f11ea2a48c5371c0a0dceef2d653e4149d0405095cb72af14e9bf",
        "output_value": 1963581,
        "sequence": 4294967295,
        "addresses": [
          "mhoHfHChKvgBycXVSaZvYnseqkNsPcmJCb"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 1940000,
        "script": "51",
        "addresses": null,
        "script_type": "unknown"
      },
      {
        "value": 1,
        "script": "00",
        "addresses": null,
        "script_type": "unknown"
      }
    ]
  }
```

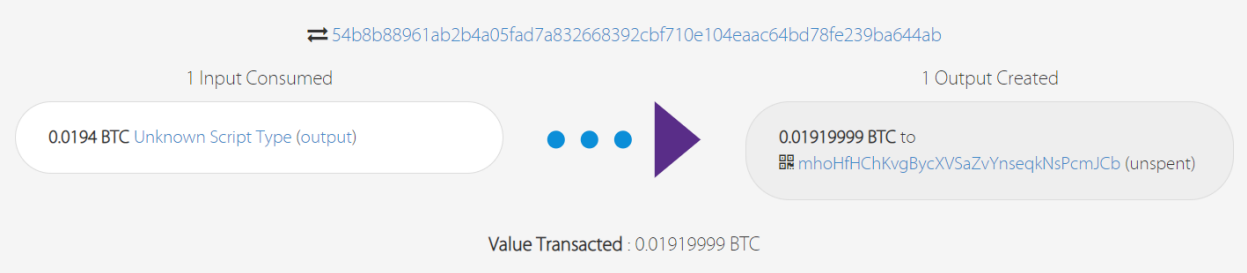
کد بخش دوم سوال:

```
def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = P2PKH_scriptPubKey()
    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)
```

هش تراکنش:

```
54b8b88961ab2b4a05fad7a832668392cbf710e104eaac64bd78fe239ba644ab
```

اطلاعات این تراکنش در لینک موجود است.



```
root@Nesa:/mnt/d/Crypto/CA/ItC-CA1/starter# python3 Q2.P1.2.py
Address =  mhoHfHChKvgBycXVSaZvYnseqkNsPcmJCb
Public_key =  047208efe5a9d4735c1f6465dbff9bcba68fec0dc894a4e566e264098d798d89509119fd7c584f11ea2a48c5371c0a0dceef2d653e4149d0405095cb72af14e9bf
Private_key =  0d660a204d7856a7958289b95edf4099a0f2058d171c551872c5a0f533177559
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "received": "2023-05-26T21:28:12.19562326Z",
    "ver": 1,
    "double_spend": false,
    "vin_sz": 1,
    "vout_sz": 1,
    "confirmations": 0,
    "inputs": [
      {
        "prev_hash": "87a02c5f72c2110c3f246c78033ef4fa2c2ab35f7d9f1c0b4473bd85808586dd",
        "output_index": 0,
        "output_value": 1940000,
        "sequence": 4294967295,
        "script_type": "unknown",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 1919999,
        "script": "76a914190712a66e19ea96149314d3df6a220fad4b324b88ac",
        "addresses": [
          "mhoHfHChKvgBycXVSaZvYnseqkNsPcmJCb"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```

سوال دوم. تولید خروجی Multisig

در ابتدا با توجه به کد سوال اول سه آدرس جدید تولید می‌کنیم که در زیر قابل مشاهده است:

```
Private_key1 = b'92zTJQyVy4QQ1Rv6LFXZVeCcZehhKCBFhpatQZxFgv6X975FFUs'
Public_key1 = b'n3zobnqdiN82q7jkgJBQb3dcdaQC4851y6'
Private_key2 = b'925aN2mT1y2roU6soRAkNYP143vnJFy9UfSsxY4SAcBeb7rVfgK'
Public_key2 = b'mntzHMN1uMsmoBs47yguAiDrfQ3qUNHy3C'
Private_key3= b'91wEoys8V76G1SwJKWmzgwu3o8bWkrfeSXJLjutTJmAMGD3vYSN'
Public_key3 = b'n2vm6GD29oax6gMNnb2Rwo7hCiboee8yUf'
```

با توجه به بخش اول سوال که میخواهد خروجی از نوع Multisig بوده و توسط ۲ نفر از این ۳ آدرس قابل خرج شدن باشد، کد بصورت زیر نوشته می‌شود:

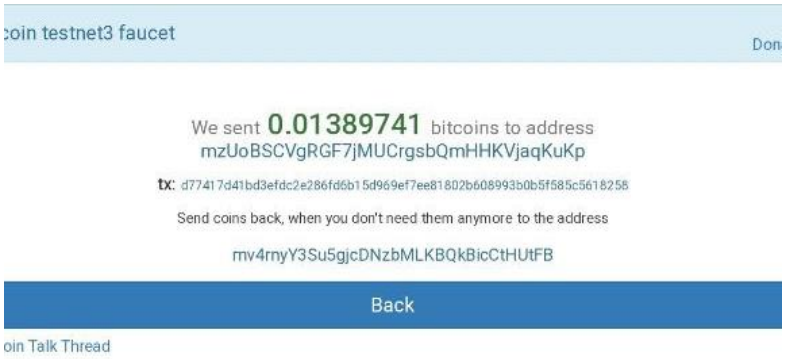
```
[OP_2, public_key1, public_key2, public_key3, OP_3, OP_CHECKMULTISIG]
```

ابتدا CHECKMULTISIG از روی استک خوانده میشود و سپس با دستور OP_3، سه عضو دیگر خوانده می‌شود. این سه عضو همان public_key های سه آدرس ما هستند. با دستور OP_2، دو عضو دیگر نیر از استک خوانده می‌شود.این دو عضو امضاهایی هستند که توسط دو کلید خصوصی انجام شده‌اند و چک میکند که دو مورد از امضاها با دو public_key همخوانی داشته باشند.

با ران کردن کد قسمت اول، private_key و public_key را بدست می‌آوریم:

Public_key = mzUoBSCVgRGF7jMUCrgsbQmHHKVjaqKuKp

Private_Key = 92grWQktDAe6G7NUmJpR6vK1yYRCALkbZ7AaLTQbg2YQtWsTKJk



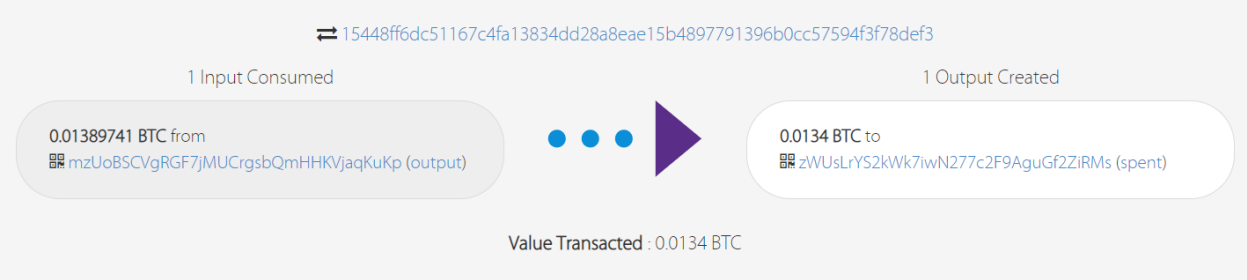
کد بخش اول:

```
bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret("92grWQktDAe6G7NUmJpR6vK1yYRCALkbZ7AaLTQbg2YQtWsTKJk")
my_public_key = my_private_key.pub
my_address = bitcoin.wallet.P2PKHBitcoinAddress.from_pubkey(my_public_key)

private_key1 =
bitcoin.wallet.CBitcoinSecret("92zTJQyVy4QQ1Rv6LFXZVeCcZehhKCBFhpatQZxFgv6X975FFUs")
private_key2 =
bitcoin.wallet.CBitcoinSecret("925aN2mT1y2roU6soRAkNYP143vnJFy9UfSsxY4SAcBeb7rVfgK")
private_key3 =
bitcoin.wallet.CBitcoinSecret("91wEoys8V76G1SwJKWmzgwu3o8bWkrfeSXJLjutTJmAMGD3vYSN")

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_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]
    txin_scriptPubKey = P2PKH_scriptPubKey()
    txout = create_txout(amount_to_send, txout_scriptPubKey)
    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__':
    amount_to_send = 0.0134
    txid_to_spend =
('d77417d41bd3efdc2e286fd6b15d969ef7ee81802b608993b0b5f585c5618258')
    utxo_index = 0
    print("Address = ", my_address)
    print("Public_key = ", my_public_key.hex())
    print("Private_key = ", my_private_key.hex())
    response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index)
    print(response.status_code, response.reason)
    print(response.text)
```

هش تراکنش:

15448ff6dc51167c4fa13834dd28a8eae15b4897791396b0cc57594f3f78def3

اطلاعات این تراکنش در لینک موجود است.



```
root@Nesa:/mnt/d/Crypto/CA/ItC-CA1/starter# python3 Q2.P2.1.py
Address =  mzUoBSCVgRGF7jMUCrgsbQmHHKVjaqKuKp
Public_key =  04e8e5adc7284a5d9b325d9b0a9f2d56a6cf6d14dd633d0d6dc94ab03d759c1c4a91b8e9c09b37c25b23cee0763df9b7f48a6dbdeb7b88aa029fc07cf3f2f753fe
Private_key =  912d3e0fdb0f35eb4409326f7d5994b551960167bb1eb6d025287059d141c0cc
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "15448ff6dc51167c4fa13834dd28a8eae15b4897791396b0cc57594f3f78def3",
    "addresses": [
      "mzUoBSCVgRGF7jMUCrgsbQmHHKVjaqKuKp",
      "zWUsLrYS2kWk7iwN277c2F9AguGf2ZiRMs"
    ],
    "total": 1340000,
    "fees": 49741,
    "size": 400,
    "outputs": [
      {
        "value": 1340000,
        "script": "524104007ca1bcabdc59d3bec569fc1d485e90962763905cdfdaa6f56637eeae4de074a04a00fec235ceb4410aa028960a9becfe01b50769d230c93e82deeef4d32f7441046912c259c49d123492b4f2379889bf329f39dad74242e53075d6bac2a32e9d9ecca36f43edddf1bc3079ac284b9bdcfe7fc7637a5f5df1a4efead26c4785ab6441046afbd467e4ffade6673d3594095373a90d0ee3ddb421c50493c03ee8c56ddeaea672459c2584aac6702a8a6075c04e5fa570b6e79b64fc15cc1028cc9736a99f53ae",
        "addresses": [
          "zWUsLrYS2kWk7iwN277c2F9AguGf2ZiRMs"
        ],
        "script_type": "pay-to-multi-pubkey-hash"
      }
    ]
  }
}
```

با استفاده از signature دو تا از سه آدرس ایجاد شده، این مقدار را به آدرس اصلی برگردانیم.

کد بخش دوم:

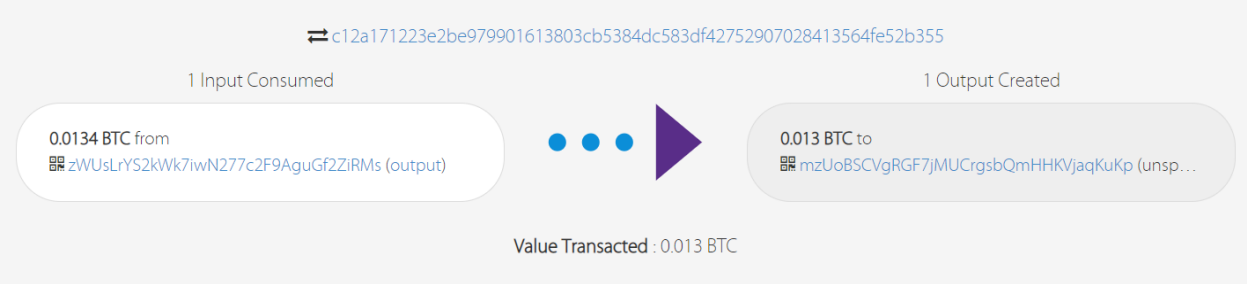
```
def P2PKH_scriptSig(txin, txout, txin_scriptPubKey):
    signature1 = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
private_key1)
    signature2 = create_OP_CHECKSIG_signature(txin, txout, txin_scriptPubKey,
private_key2)
    return [OP_0, signature1, signature2]

def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = P2PKH_scriptPubKey()
    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 = P2PKH_scriptSig(txin, [txout], txin_scriptPubKey)
    new_tx = create_signed_transaction(txin, [txout], txin_scriptPubKey,
txin_scriptSig)
    return broadcast_transaction(new_tx)
```

مقدار txin_scriptSig برابر با [OP_0, signature1, signature2] قرار می‌گیرد. در این حالت دو امضای مشخص شده با دو public_key چک شده و در صورت همخوانی داشتن، بیت کوین را می‌توان خرج کرد. OP_0 به این دلیل در پایین استک قرار دارد که CHECKMULTISIG_OP یک عضو اضافه از استک pop می کند.

هش تراکنش:

```
c12a171223e2be979901613803cb5384dc583df42752907028413564fe52b355
```

اطلاعات این تراکنش در لینک موجود است.




```
root@Nesa:/mnt/d/Crypto/CA/ItC-CA1/starter# python3 Q2.P2.2.py
Public_key =  04e8e54dc7284a5d9b325d9b0a9f2d56a6cf6d14dd633d0d6dc94ab03d759c1c4a91b8e9c09b37c25b23cee0763df9b7f48a6dbdeb7b88aa029fc07cf3f2f753fe
Private_key =  912d3e0fdb0f35eb4409326f7d5994b551960167bb1eb6d025287059d141c0cc
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "c12a171223e2be979901613803cb5384dc583df42752907028413564fe52b355",
    "addresses": [
      "mzUoBSCVgRGF7jMUCrgsbQmHHKVjaqKuKp",
      "zWUsLrYS2kWk7iwN277c2F9AguGf2ZiRMs"
    ],
    "total": 1300000,
        "output_value": 1340000,
        "sequence": 4294967295,
        "addresses": [
          "zWUsLrYS2kWk7iwN277c2F9AguGf2ZiRMs"
        ],
        "script_type": "pay-to-multi-pubkey-hash",
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 1300000,
        "script": "76a914d001f32bca5f00254a9173cbb84cc2940d83ed6788ac",
        "addresses": [
          "mzUoBSCVgRGF7jMUCrgsbQmHHKVjaqKuKp"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```

سوال سوم. اطلاع از دو عدد اول برای تراکنش

در این قسمت جمع و تفریق دو عدد اول را بدست آورده و آنهارا تبدیل به byte می‌کنیم:

```
PRIME1 = 4801
PRIME2 = 3461
SUM = PRIME1 + PRIME2
DIFF = PRIME1 - PRIME2
```

```
SUM_BYTES = SUM.to_bytes(2, byteorder="little")
DIFF_BYTES = DIFF.to_bytes(2, byteorder="little")
```

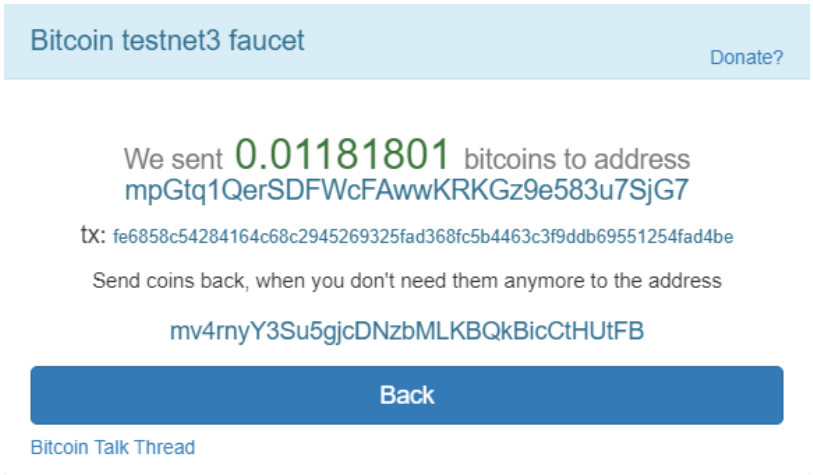
سپس باید هش آنهارا در اسکریپ قرار دهیم، بنابراین کد نوشته شده به شکل زیر خواهد بود:

```
[OP_2DUP, OP_SUB, OP_HASH160, Hash160(DIFF_BYTES), OP_EQUALVERIFY,
OP_ADD, OP_HASH160, Hash160(SUM_BYTES), OP_EQUAL]
```

با ران کردن کد قسمت اول، private_key و public_key را بدست می‌آوریم:

Public_key = mpGtq1QerSDFWcFAwwKRKGz9e583u7SjG7

Private_Key = 92hHBfhRXkQe2zNwu7eYowNu32Wv2CcQmw7ZTas2GUqjSh51PUG



کد بخش اول:

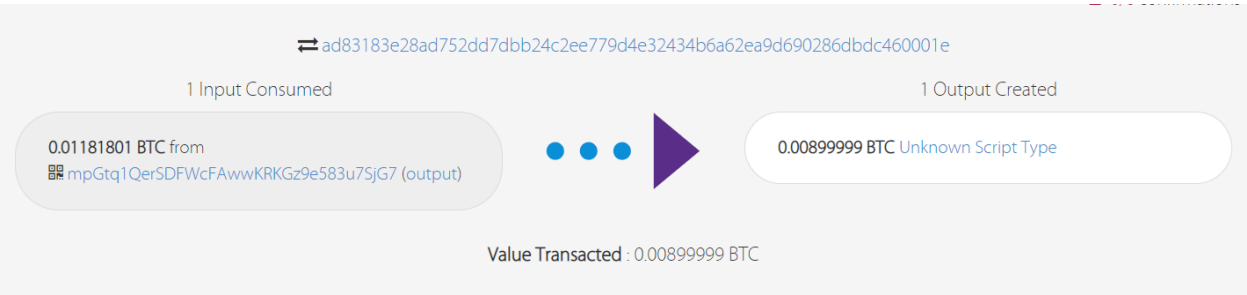
```
PRIME1 = 4801
PRIME2 = 3461
SUM = PRIME1 + PRIME2
DIFF = PRIME1 - PRIME2
SUM_BYTES = SUM.to_bytes(2, byteorder="little")
DIFF_BYTES = DIFF.to_bytes(2, byteorder="little")

bitcoin.SelectParams("testnet")
my_private_key =
bitcoin.wallet.CBitcoinSecret("92hHBfhRXkQe2zNwu7eYowNu32Wv2CcQmw7ZTas2GUqjSh51PUG")
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_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = [OP_2DUP, OP_SUB, OP_HASH160, Hash160(DIFF_BYTES),
OP_EQUALVERIFY, OP_ADD, OP_HASH160, Hash160(SUM_BYTES), OP_EQUAL]
    txout = create_txout(amount_to_send, txout_scriptPubKey)
```

```
    txin_scriptPubKey = P2PKH_scriptPubKey()
    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__':
    amount_to_send = 0.0001
    txid_to_spend =
('fe6858c54284164c68c2945269325fad368fc5b4463c3f9ddb69551254fad4be')
    utxo_index = 0
    print("Address = ", my_address)
    print("Public_key = ", my_public_key.hex())
    print("Private_key = ", my_private_key.hex())
    response = send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index)
    print(response.status_code, response.reason)
    print(response.text)
```

هش تراکنش:

ad83183e28ad752dd7dbb24c2ee779d4e32434b6a62ea9d690286dbdc460001e

اطلاعات این تراکنش در لینک موجود است.




```
root@Nesa:/mnt/d/Crypto/CA/ItC-CA1/starter# python3 Q2.P3.1.py
Address =  mpGtq1QerSDFWcFAwwKRKGz9e583u7SjG7
Public_key =  04876bea3ce2182f3984531d3040ecd1ffe7da628c3bd3f3f93c13f9b00ca7ee9c3ff2d41870a71e602089cd820aac2b7acc03e87d75533cde0e9d2da950c71027
Private_key =  92248e26f5ea2332c6f72ebace5a2a6fec2e1e4e1831a5f7714a5bd99b406908
201 Created
{
  "tx": {
    "block_height": -1,
        "addresses": [
          "mpGtq1QerSDFWcFAwwKRKGz9e583u7SjG7"
        ],
        "script_type": "pay-to-pubkey-hash",
        "age": 2435595
      }
    ],
    "outputs": [
      {
        "value": 899999,
        "script": "6e94a914aff7ff3f525736951432dd496fb409cb76c3e2d78893a9147fc917badf20815d8dd0124fadc85b72f0abb3b487",
        "addresses": null,
        "script_type": "unknown"
      }
    ]
  }
}
```

کد بخش دوم:

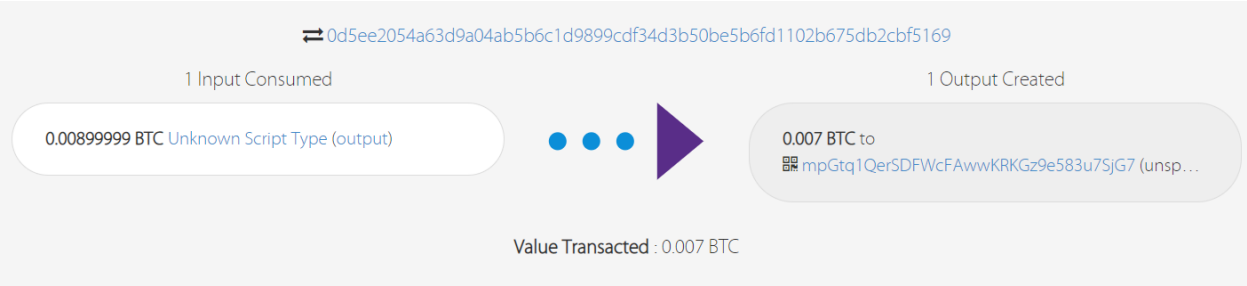
```
def send_from_P2PKH_transaction(amount_to_send, txid_to_spend, utxo_index):
    txout_scriptPubKey = P2PKH_scriptPubKey()
    txout = create_txout(amount_to_send, txout_scriptPubKey)
```

```
    txin_scriptPubKey =  [OP_2DUP, OP_SUB, OP_HASH160, Hash160(DIFF_BYTES),
OP_EQUALVERIFY, OP_ADD, OP_HASH160, Hash160(SUM_BYTES), OP_EQUAL]
    txin = create_txin(txid_to_spend, utxo_index)
    txin_scriptSig = [PRIME1, PRIME2]
    new_tx = create_signed_transaction(txin, [txout], txin_scriptPubKey,
txin_scriptSig)
    return broadcast_transaction(new_tx)
```

هش تراکنش:

```
0d5ee2054a63d9a04ab5b6c1d9899cdf34d3b50be5b6fd1102b675db2cbf5169
```

اطلاعات این تراکنش در لینک موجود است.




```
root@Nesa:/mnt/d/Crypto/CA/ItC-CA1/starter# python3 Q2.P3.2.py
Address =  mpGtq1QerSDFWcFAwwKRKGz9e583u7SjG7
Public_key =  04876bea3ce2182f3984531d3040ecd1ffe7da628c3bd3f3f93c13f9b00ca7ee9c3ff2d41870a71e602089cd820aac2b7acc03e87d75533cde0e9d2da950c71027
Private_key =  92248e26f5ea2332c6f72ebace5a2a6fec2e1e4e1831a5f7714a5bd99b406908
201 Created
{
  "tx": {
    "block_height": -1,
    "block_index": -1,
    "hash": "0d5ee2054a63d9a04ab5b6c1d9899cdf34d3b50be5b6fd1102b675db2cbf5169",
    "addresses": [
      "mpGtq1QerSDFWcFAwwKRKGz9e583u7SjG7"
    ],
        "age": 0
      }
    ],
    "outputs": [
      {
        "value": 700000,
        "script": "76a91460106a412122eac9bd826dd4ba79f66a016883fe88ac",
        "addresses": [
          "mpGtq1QerSDFWcFAwwKRKGz9e583u7SjG7"
        ],
        "script_type": "pay-to-pubkey-hash"
      }
    ]
  }
}
```

قسمت سوم: استخراج بیت کوین

ابتدا یک کلید خصوصی ۳۲ بیتی به فرمت WIF تولید می کنیم. برای mine کردن block ها مرحله به مرحله پیش می‌رویم و هرکدام را توضیح می‌دهیم. در ابتدا باید یک تراکنش ایجاد شود. برای ایجاد تراکنش همانطور که در قسمت های قبل دیدیم، به ایندکس تراکنش(utxo_index)، آیدی تراکنش(txid_to_spend) و مقدار پول (amount_to_send) احتیاج داریم. ایندکس تراکنش را برابر 0xFFFFFFFF، آیدی تراکنش را برابر ۶۴ بیت صفر و مقدار بیت کوین را برابر 6.25 قرار می‌دهیم. مقدار scriptSig را با توجه به اندازه COINBASE_HEX_DATA تعیین می‌شود. از آنجایی که هر ۴بیت یک هگز است و در اینجا میخواهیم مقدار را به بایت تبدیل کنیم، پس همانطور که در کد نیز مشاهده می‌شود، باید یک تقسیم بر دو انجام دهیم.

```
def make_coinbase_transaction(amount_to_send):
    txid_to_spend = (64*'0')
    utxo_index = int('0xffffffff', 16)
    txout_scriptPubKey = P2PKH_scriptPubKey()
    txin = create_txin(txid_to_spend, utxo_index)
    txout = create_txout(amount_to_send, txout_scriptPubKey)
    new_tx = CMutableTransaction([txin], [txout])
    txin.scriptSig =  CScript([int(COINBASE_HEX_DATA,
16).to_bytes(len(COINBASE_HEX_DATA)//2, 'big')])
    return new_tx
```

برای بدست آوردن merkel_root باید دو بار تابع درهم ساز SHA256 را روی coinbase_tx.serialize اعمال کنیم. و سپس برای بدست آوردن block_body، هگز coinbase_tx.serialize را بدست می‌آوریم.

```
def get_merkle_root(coinbase_tx):
    merkle_root =
b2lx(hashlib.sha256(hashlib.sha256(coinbase_tx.serialize()).digest()).digest())
    print("Merkle root: ", merkle_root)
    block_body = (coinbase_tx.serialize()).hex()
    print("Block body: ", block_body)
    return merkle_root, block_body
```

در مرحله بعدی باید partial_header را بدست آوریم. برای ساخت header به اطلاعات ورژن، هش هدر بلوک قبلی، merkle_root و زمان کنونی احتیاج داریم.

```
def get_partial_header(version, prev_hash_block, merkle_root, timestamp, bits):
    partial_header = struct.pack('<L', version) + bytes.fromhex(prev_hash_block)[::-1]
+ bytes.fromhex(merkle_root[::-1]) + struct.pack('<LL', timestamp, int(bits, 16))
    return partial_header
```

کار struct.pack(‘<L’, x) به اینصورت است که x را به فرم little endian و بصورت long برمی‌گرداند.

در قسمت بعد باید درجه سختی را از طریق فرمول بدست آوریم. طبق گفته پروژه، درجه سختی بطوریست که ۴ بیت سمت چپ باید صفر باشند، بنابراین مقدار تغییر bits را برابر 0x1f010000 قرار می‌دهیم.

```
def get_target(bits):
    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_byte = bytes.fromhex(str(target_hex).zfill(64))
    print("Target in hex: ", str(target_hex).zfill(64))
    return target_byte
```

بعد از انجام این مراحل و بدست آوردن داده های مورد نیاز، به قسمت اصلی mine کردن block ها می‌رسیم.
برای استخراج بیت کوین باید مقدار nonce به‌گونه ای باشد که هش حاصل از header partial و nonce از مقدار target کوچکتر باشد.

```
def block_mining(prev_block_hash, bits):
    coinbase_tx = make_coinbase_transaction(BLOCK_REWARD)
    merkle_root, block_body = get_merkle_root(coinbase_tx)
    partial_header = get_partial_header(2, prev_block_hash, merkle_root,
int(time.time()), bits)
    nonce = 0
    target = get_target(bits)
    while nonce <= 0xFFFFFFFF: # = 2**32
        header = partial_header + struct.pack('<L', nonce)
        hash = hashlib.sha256(hashlib.sha256(header).digest()).digest()
        start_time = time.time()
        if hash[::-1] < target:
            print('Hash: ', hash)
            print('Nonce: ', nonce)
            print('Hash rate: ', nonce/(time.time() - start_time))
            return hash, header, block_body
        nonce += 1
```

و درنهایت مقادیر بدست آمده را نمایش می‌دهیم.

```
root@Nesa:/mnt/d/Crypto/CA/ItC-CA1/starter# python3 q3.py
Address =  12X2JGVC1f5DFJcfDgAegWszem1Q7w3Spw
coinbase hexadecimal data:  383130313939393435374e657361416262617369
please enter your block number: 9457
please enter prevoius block hash: 00000000d5dde2b1403c05b8546073aaab6b1c599fa86cf32751dc32d027e712
Merkle root:  711e4f5fbc512359e180d7a2dc01fccd1b4384bf45c2f6afb36f61d563fc5e0d
Block body:  0100000001000000000000000000000000000000000000000000000000000000000000000000ffffffff1413383130313939393435374e6573
61416262617369ffffffff0140be4025000000001976a91410a5b01ac7a29e0645a9d2dbae1ae0dd873b945088ac00000000
Target in hex:  0001000000000000000000000000000000000000000000000000000000000000
Hash:  b'\xc7\xc2G\x15\x9c\x14C\xb5\x9b*@\xd9C\xd8\x10\xda;\x1cU~#\xe6A\xc7\x91\xd3b1/\xf8\x00\x00'
Nonce:  20652
Hash rate:  250349035.28323698
Block hash: 0000f82f3162d391c741e6237e551c3bda10d843d9402a9bb543149c1547c2c7
Block header: 0200000012e727d032dc5127f36ca89f591c6babaa736054b8053c40b1e2ddd5000000000d5efc63d5616fb3aff6c245bf84431bcdfc0
1dca2d780e1592351bc5f4f1e71261771640000011fac500000
Block:  b"\xf9\xbe\xb4\xd9#\x01\x00\x00\x02\x00\x00\x00\x12\xe7'\xd02\xdcQ'\xf3l\xa8\x9fY\x1ck\xab\xaas`T\xb8\x05<@\xb1\xe2
\xdd\xd5\x00\x00\x00\x00\r^\xfcc\xd5ao\xb3\xaf\xf6\xc2E\xbf\x84C\x1b\xcd\xfc\x01\xdc\xa2\xd7\x80\xe1Y#Q\xbc_O\x1eq&\x17qd\x
00\x00\x01\x1f\xacP\x00\x00\x01\x01\x00\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\xff\xff\xff\xff\x14\x13810199457NesaAbbasi\xff\xff\xff\xff\x01@\xb
e@%\x00\x00\x00\x00\x19v\xa9\x14\x10\xa5\xb0\x1a\xc7\xa2\x9e\x06E\xa9\xd2\xdb\xae\x1a\xe0\xdd\x87;\x94P\x88\xac\x00\x00\x00
\x00"
```

اطلاعات و توضیحات بیشتر کدهای این قسمت در لینک قابل مشاهده است.