# قسمت اول: تولید آدرس

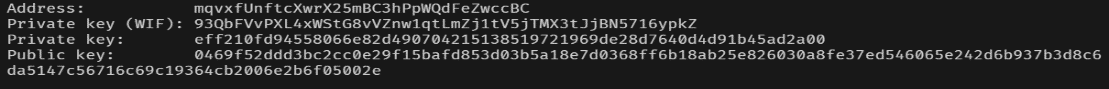
## تولید آدرس عادی

برای این کار ابتدا یک عدد 256 بیتی رندوم را به عنوان Private Key انتخاب می‌کنیم. پس از اعمال ECDSA بر روی آن، مقدار Public Key بدست می‌آید. سپس، پس از یک بار اعمال SHA256 و یک بار اعمال RIPEMD160 پس از آن، اگر نتیجه را به فرمت WIF تبدیل کنیم، آدرس کیف پول مورد نظر بدست می‌آید. در این بخش از کلاس زیر (فایل part1\_q1.py) استفاده شده است:

class Wallet:  
 class Network(enum.Enum):  
 MAINNET = 0  
 TESTNET = 1  
  
 def \_\_init\_\_(self, network: Network = Network.TESTNET):  
 self.network = network  
 self.\_private\_key = b""  
 self.\_public\_key = b""  
 self.\_bitcoin\_address = ""  
  
 def generate(self) -> None:  
 self.\_private\_key = secrets.token\_bytes(32)  
 self.\_generate\_public\_key()  
 self.\_generate\_bitcoin\_address()  
  
 def generate\_from\_wif(self, private\_key\_wif: str) -> None:  
 self.\_private\_key = self.\_from\_wif(private\_key\_wif)  
 self.\_generate\_public\_key()  
 self.\_generate\_bitcoin\_address()  
  
 def \_get\_network\_byte(self, is\_private: bool = True) -> bytes:  
 if is\_private:  
 if self.network == Wallet.Network.MAINNET:  
 return b"\x80"  
 if self.network == Wallet.Network.TESTNET:  
 return b"\xef"  
 raise ValueError("Invalid network")  
 elif self.network == Wallet.Network.MAINNET:  
 return b"\x00"  
 elif self.network == Wallet.Network.TESTNET:  
 return b"\x6f"  
 else:  
 raise ValueError("Invalid network")  
  
 def \_generate\_public\_key(self) -> None:  
 public\_key = ecdsa.SigningKey.from\_string(  
 self.\_private\_key, curve=ecdsa.SECP256k1  
 ).verifying\_key  
  
 if public\_key is None:  
 raise ValueError("Invalid public key")  
 self.\_public\_key = (b"\x04" + public\_key.to\_string()) *# 0x04 is the prefix for uncompressed public keys*  
  
 def \_generate\_bitcoin\_address(self) -> None:  
 sha256 = hashlib.sha256(self.\_public\_key).digest()  
 ripemd160 = hashlib.new("ripemd160")  
 ripemd160.update(sha256)  
  
 self.\_bitcoin\_address = self.\_to\_wif(ripemd160.digest(), is\_private=False)  
  
 def \_to\_wif(self, key: bytes, is\_private: bool = True) -> str:  
 network\_byte = self.\_get\_network\_byte(is\_private)  
 key\_with\_network\_byte = network\_byte + key  
 sha256\_1 = hashlib.sha256(key\_with\_network\_byte).digest()  
 sha256\_2 = hashlib.sha256(sha256\_1).digest()  
 checksum = sha256\_2[:4]  
 binary\_key = key\_with\_network\_byte + checksum  
 wif = base58.b58encode(binary\_key).decode("utf-8")  
 return wif  
  
 def \_from\_wif(self, wif: str) -> bytes:  
 binary\_key = base58.b58decode(wif)  
 key = binary\_key[:-4]  
 checksum = binary\_key[-4:]  
 sha256\_1 = hashlib.sha256(key).digest()  
 sha256\_2 = hashlib.sha256(sha256\_1).digest()  
 if checksum != sha256\_2[:4]:  
 raise ValueError("Invalid WIF")  
 network\_byte = key[0:1]  
 if network\_byte != self.\_get\_network\_byte():  
 raise ValueError("Invalid WIF")  
 return key[1:]

تفاوت آدرس در شبکه اصلی و در شبکه تست، در Network Byte آن‌ها است. همانطور که در کد مشاهده می‌شود، برای شبکه اصلی مقدار 0x00 به ابتدای آدرس اضافه می‌شود و برای شبکه تست، مقدار 0x6F به آن اضافه می‌شود. در نتیجه زمانی که آدرس را به Base58 تبدیل می‌کنیم، در شبکه اصلی، اولین کاراکتر آدرس همواره 1 خواهد بود و در شبکه تست، این کاراکتر می‌تواند m و یا n باشد.

خروجی این کد به صورت زیر است:



موارد خواسته شده به صورت زیر هستند:

Address: mqvxfUnftcXwrX25mBC3hPpWQdFeZwccBC

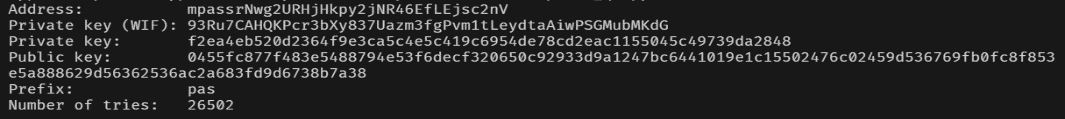
Private Key (WIF): 93QbFVvPXL4xWStG8vVZnw1qtLmZj1tV5jTMX3tJjBN5716ypkZ

## تولید آدرس ویژه

در این حالت باید به تعداد زیادی Private Key مختلف را آزمایش کنیم تا در نهایت آدرس تولید شده، با پیشوند مورد نظر ما آغاز شود. در این بخش من پیشوند pas (3 حرف ابتدای اسمم) را انتخاب کردم. با توجه به اینکه می‌خواهیم 3 حرف ابتدایی آدرس ثابت باشد، به طور متوسط باید حدود کلید را آزمایش کنیم تا به نتیجه برسیم. برای این کار یک کلاس جدید ایجاد می‌کنیم (فایل part1\_q2.py) که از کلاس قبلی ارث می‌برد:

class VanityWallet(Wallet):  
 def \_\_init\_\_(self, prefix: str, network: Wallet.Network = Wallet.Network.TESTNET):  
 super().\_\_init\_\_(network)  
 self.\_prefix = prefix  
 self.\_number\_of\_tries = 0  
  
 *# some properties*  
  
 def generate(self) -> None:  
 while True:  
 super().generate()  
 self.\_number\_of\_tries += 1  
 if self.bitcoin\_address[1:].startswith(self.prefix):  
 break

خروجی این کد به صورت زیر است:



موارد خواسته شده به صورت زیر هستند:

Address: mpassrNwg2URHjHkpy2jNR46EfLEjsc2nV

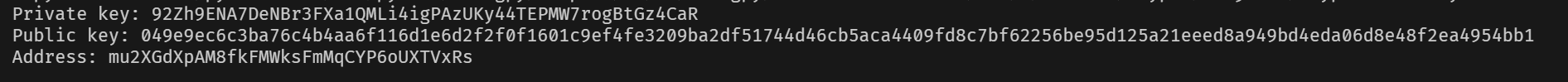
Private Key (WIF): 93Ru7CAHQKPcr3bXy837Uazm3fgPvm1tLeydtaAiwPSGMubMKdG

# قسمت دوم: انجام تراکنش

ابتدا باید از طریق Faucetهای ارائه شده، مقداری پول به آدرس‌مان اتقال دهیم. در این بخش، برای ساخت آدرس از کد زیر استفاده شده و برای مقدار private\_key\_wif، یکی از کلیدهای ساخته شده به فرم WIF توسط سوال اول بخش اول استفاده شده است:

if \_\_name\_\_ == "\_\_main\_\_":  
 private\_key\_wif = "92Zh9ENA7DeNBr3FXa1QMLi4igPAzUKy44TEPMW7rogBtGz4CaR"  
 bitcoin.SelectParams("testnet")  
 private\_key = bitcoin.wallet.CBitcoinSecret(private\_key\_wif)  
 public\_key = private\_key.pub  
 address = bitcoin.wallet.P2PKHBitcoinAddress.from\_pubkey(public\_key)  
 print(f"Private key: {private\_key}")  
 print(f"Public key: {public\_key.hex()}")  
 print(f"Address: {address}")

خروجی به صورت زیر است:



مقادیر خواسته شده به صورت زیر هستند:

Private Key (WIF): 92Zh9ENA7DeNBr3FXa1QMLi4igPAzUKy44TEPMW7rogBtGz4CaR

Address: mu2XGdXpAM8fkFMWksFmMqCYP6oUXTVxRs

از Faucet اول مقدار 0.0137566 بیت‌کوین به آدرس من انتقال داده شد. مشخصات این تراکنش در [این لینک](https://live.blockcypher.com/btc-testnet/tx/12a4ccff3ed9bec0715fcf678b914397e1e7b3fcea239fe551f269c89810f909/) و [این لینک](https://blockstream.info/testnet/tx/12a4ccff3ed9bec0715fcf678b914397e1e7b3fcea239fe551f269c89810f909?expand) قابل مشاهده است. هش تراکنش نیز به صورت زیر است:

12a4ccff3ed9bec0715fcf678b914397e1e7b3fcea239fe551f269c89810f909

تمامی تراکنش‌های این بخش به طور خلاصه از [این لینک](https://live.blockcypher.com/btc-testnet/address/mu2XGdXpAM8fkFMWksFmMqCYP6oUXTVxRs/) قابل مشاهده هستند.

برای ایجاد تراکنش‌های این بخش، ابتدا تعدادی کلاس نوشته شد که در فایل transaction.py قابل دسترس هستند:

def address\_to\_pub\_key\_hash160(address: str) -> bytes:  
 pub\_key\_hash = base58.b58decode\_check(address)[1:]  
 return pub\_key\_hash  
  
  
def P2PKH\_script\_pub\_key(pub\_key\_hash: bytes) -> CScript:  
 return CScript([OP\_DUP, OP\_HASH160, pub\_key\_hash, OP\_EQUALVERIFY, OP\_CHECKSIG]) *# type: ignore*  
  
  
class Destination:  
 def \_\_init\_\_(  
 self, address: str, amount: float, script\_pub\_key: CScript | None = None  
 ):  
 self.\_address = address  
 self.\_script\_pub\_key = (  
 P2PKH\_script\_pub\_key(address\_to\_pub\_key\_hash160(address))  
 if script\_pub\_key is None  
 else script\_pub\_key  
 )  
 self.\_amount = amount  
  
 *# some properties*  
  
 @property  
 def TxOut(self) -> CMutableTxOut:  
 return CMutableTxOut(int(self.amount \* COIN), self.script\_pub\_key)  
  
  
class UnspentTransactionOutput:  
 def \_\_init\_\_(  
 self,  
 tx\_id: str,  
 index: int,  
 script\_pub\_key: CScript,  
 custom\_sig: CScript | None = None,  
 ):  
 self.\_tx\_id = tx\_id  
 self.\_index = index  
 self.\_script\_pub\_key = script\_pub\_key  
 self.\_custom\_sig = custom\_sig  
  
 *# some properties*  
  
 @property  
 def TxIn(self) -> CMutableTxIn:  
 return CMutableTxIn(COutPoint(lx(self.tx\_id), self.index))

class Transaction:  
 class Network(enum.Enum):  
 MAINNET = "mainnet"  
 TESTNET = "testnet"  
  
 def \_\_init\_\_(self, private\_key: str, network: Network = Network.TESTNET):  
 self.\_network = network  
 bitcoin.SelectParams(self.\_network.value)  
  
 self.\_private\_key = bitcoin.wallet.CBitcoinSecret(private\_key)  
 self.\_public\_key = self.\_private\_key.pub  
 self.\_address = bitcoin.wallet.P2PKHBitcoinAddress.from\_pubkey(self.\_public\_key)  
 self.\_destinations = []  
 self.\_utxos = []  
 self.\_tx = CMutableTransaction()  
  
 @property  
 def address(self) -> str:  
 return str(self.\_address)  
  
 def add\_destination(self, destination: Destination) -> None:  
 self.\_destinations.append(destination)  
  
 def add\_utxo(self, utxo: UnspentTransactionOutput) -> None:  
 self.\_utxos.append(utxo)  
  
 def create(self) -> requests.Response:  
 if not self.\_destinations:  
 raise ValueError("No destinations were added to the transaction")  
 if not self.\_utxos:  
 raise ValueError(  
 "No unspent transaction outputs were added to the transaction"  
 )  
 self.\_create\_transaction()  
 self.\_verify()  
 return self.\_broadcast\_transaction()  
  
 def my\_P2PKH\_script\_pub\_key(self) -> CScript:  
 return P2PKH\_script\_pub\_key(Hash160(self.\_public\_key))  
  
 def \_my\_P2PKH\_script\_sig(self, txin\_script\_pub\_key: CScript) -> CScript:  
 signature = self.\_create\_OP\_CHECKSIG\_signature(txin\_script\_pub\_key)  
 return CScript([signature, self.\_public\_key]) *# type: ignore*  
  
 def \_create\_transaction(self) -> None:  
 txins = [utxo.TxIn for utxo in self.\_utxos]  
 txouts = [destination.TxOut for destination in self.\_destinations]  
 self.\_tx = CMutableTransaction(txins, txouts)  
  
 def \_create\_OP\_CHECKSIG\_signature(self, txin\_script\_pub\_key: CScript) -> bytes:  
 sighash = SignatureHash(txin\_script\_pub\_key, self.\_tx, 0, SIGHASH\_ALL)  
 sig = self.\_private\_key.sign(sighash) + bytes([SIGHASH\_ALL]) *# type: ignore*  
 return sig  
  
 def \_verify(self):  
 for i, \_ in enumerate(self.\_utxos):  
 txin\_script\_pub\_key = self.\_utxos[i].script\_pub\_key  
 txin\_script\_sig = self.\_utxos[i].custom\_sig  
 if txin\_script\_sig is None:  
 txin\_script\_sig = self.\_my\_P2PKH\_script\_sig(txin\_script\_pub\_key)  
 self.\_tx.vin[i].scriptSig = txin\_script\_sig  
 VerifyScript(  
 self.\_tx.vin[i].scriptSig,  
 txin\_script\_pub\_key,  
 self.\_tx,  
 i,  
 (SCRIPT\_VERIFY\_P2SH,),  
 )  
  
 def \_broadcast\_transaction(self) -> requests.Response:  
 raw\_transaction = b2x(self.\_tx.serialize())  
 headers = {"content-type": "application/x-www-form-urlencoded"}  
 return requests.post(  
 TRANSACTION\_BROADCAST\_URL,  
 headers=headers,  
 data='{"tx": "%s"}' % raw\_transaction,  
 timeout=60,  
 )

## خروجی اول غیر قابل خرج و خروجی دوم قابل خرج توسط هرکس

برای خروجی‌ای که توسط هیچ‌کس قابل خرج نیست، می‌توانیم از اسکریپت زیر استفاده کنیم:

Script = OP\_RETURN

همچنین برای خروجی‌ای که توسط هرکس قابل خرج شدن است، می‌توان از اسکریپت زیر استفاده کرد:

Script = OP\_CHECKSIG

در این بخش از مقدار 0.0137566 بیت‌کوین موجود، 0.002 آن را به خروجی غیر قابل خرج، 0.008 آن را به خروجی قابل خرج توسط هرکس و باقی آن را به Transaction Fee اختصاص دادم. این کار توسط کد زیر که در فایل part2\_q1\_1.py قابل دسترس است، انجام شده است:

UNSPENDABLE\_SCRIPT\_PUB\_KEY = CScript([OP\_RETURN]) *# type: ignore*  
SPENDABLE\_BY\_ANYONE\_SCRIPT\_PUB\_KEY = CScript([OP\_CHECKSIG]) *# type: ignore*  
  
  
def main():  
 private\_key = "92Zh9ENA7DeNBr3FXa1QMLi4igPAzUKy44TEPMW7rogBtGz4CaR"  
  
 tx = Transaction(private\_key)  
 tx.add\_destination(Destination(tx.address, 0.002, UNSPENDABLE\_SCRIPT\_PUB\_KEY))  
 tx.add\_destination(  
 Destination(tx.address, 0.008, SPENDABLE\_BY\_ANYONE\_SCRIPT\_PUB\_KEY)  
 )  
 tx.add\_utxo(  
 UnspentTransactionOutput(  
 "12a4ccff3ed9bec0715fcf678b914397e1e7b3fcea239fe551f269c89810f909",  
 0,  
 tx.my\_P2PKH\_script\_pub\_key(),  
 )  
 )  
  
 resp = tx.create()  
 print(f"[{resp.status\_code}] {resp.reason}")  
 print(resp.text)

اطلاعات این تراکنش در [این لینک](https://live.blockcypher.com/btc-testnet/tx/76d3ef0f1c733e5b6a15da0233ceca7a5694674cf3f511c6015fdc3d7f52b00a/) و [این لینک](https://blockstream.info/testnet/tx/76d3ef0f1c733e5b6a15da0233ceca7a5694674cf3f511c6015fdc3d7f52b00a?expand) قابل دسترس است. هش تراکنش نیز به صورت زیر است:

76d3ef0f1c733e5b6a15da0233ceca7a5694674cf3f511c6015fdc3d7f52b00a

حال باید مقدار قابل خرج را به حساب خود برگردانیم. از مقدار 0.008 بیت‌کوین این خروجی، 0.0078 را به حساب خودم برگرداندم و باقی آن را به Transaction Fee اختصاص دادم. این کار را با روش P2PKH و با کد زیر که در فایل part2\_q1\_2.py قرار دارد، انجام دادم:

SPENDABLE\_BY\_ANYONE\_SCRIPT\_PUB\_KEY = CScript([OP\_CHECKSIG]) *# type: ignore*  
  
  
def main():  
 private\_key = "92Zh9ENA7DeNBr3FXa1QMLi4igPAzUKy44TEPMW7rogBtGz4CaR"  
  
 tx = Transaction(private\_key)  
 tx.add\_destination(Destination("mu2XGdXpAM8fkFMWksFmMqCYP6oUXTVxRs", 0.0078))  
 tx.add\_utxo(  
 UnspentTransactionOutput(  
 "76d3ef0f1c733e5b6a15da0233ceca7a5694674cf3f511c6015fdc3d7f52b00a",  
 1,  
 SPENDABLE\_BY\_ANYONE\_SCRIPT\_PUB\_KEY,  
 )  
 )  
  
 resp = tx.create()  
 print(f"[{resp.status\_code}] {resp.reason}")  
 print(resp.text)

اطلاعات این تراکنش در [این لینک](https://live.blockcypher.com/btc-testnet/tx/15c794bbb169f272fb1ed45526eef5613a851716a1df574480c12f23126772be/) و [این لینک](https://blockstream.info/testnet/tx/15c794bbb169f272fb1ed45526eef5613a851716a1df574480c12f23126772be?expand) قابل دسترس است. همچنین، هش تراکنش به صورت زیر است:

15c794bbb169f272fb1ed45526eef5613a851716a1df574480c12f23126772be

## خروجی از نوع MultiSig

برای این بخش ابتدا 3 آدرس جدید ایجاد می‌کنیم که به صورت زیر هستند:

Private Key 1: 93RdsGKJn6ExkLEWpVogpC5kRibgoJDWZWUzjonohYhynkScHLX

Public Key 1: 04c9a3dc14a523ad8b6b36f445ce55475e0f716ff0a8e7f92103e36cb344  
a64a2160c48c2e97a8471537da6154acf6d612d2eae18af07884e25eec1b4dd07a59e9

Address 1: n3FpTJHHU17VrYxgAbUmuhYUvRoCTzhEzR

Private Key 2: 938zdHuD6PVuUb26s31xevjTBPY6fggvrxv9fhJz5UGeAfqF61j

Public Key 2: 04844325ab760b86d29b90a59d88d57ed3ef8b9124b804257ec15d7b4e42  
6400cc74e028f73c949e36b1919aaf823dbc35440538e49151e28ce8c80528f0e0dd21

Address 2: mgPaYYnNtAC6V3NNrHjfU43HkszV9vszFu

Private Key 3: 91cxgQtoYrMLUcDi3PsRvQnPQa9hgx9jc3keaq3vNXzmiaBFBrW

Public Key 3: 048f6a3736d960a861bce581abd4e24f87977dc2804cd3eddec08dbe0380  
4aea562ba30396818f3cf3cfdf0eb0ed8841abb2b3b28510ce89922d275f1f58e707b8

Address 3: mmux7bRFj5SVKKQyqGvrJiysSNJw2D9Axw

حال از حالت 2-of-3 در روش MultiSig استفاده می‌کنیم. اسکریپت زیر می‌تواند در این بخش مورد استفاده قرار بگیرد:

Script: 2 <public\_key1> <public\_key2> <public\_key3> 3 OP\_CHECKMULTISIG

در این بخش از کد زیر استفاده کردم که در فایل part2\_q2\_1.py قابل دسترس است:

def multi\_sig\_2\_of\_3(pub1: bytes, pub2: bytes, pub3: bytes) -> CScript:  
 return CScript([2, pub1, pub2, pub3, 3, OP\_CHECKMULTISIG]) *# type: ignore*  
  
  
def main():  
 private\_key = "92Zh9ENA7DeNBr3FXa1QMLi4igPAzUKy44TEPMW7rogBtGz4CaR"  
  
 pub1 = bytes.fromhex( "04c9a3dc14a523ad8b6b36f445ce55475e0f716ff0a8e7f92103e36cb344a64a2160c48c2e97a8471537da6154acf6d612d2eae18af07884e25eec1b4dd07a59e9"  
 )  
 pub2 = bytes.fromhex( "04844325ab760b86d29b90a59d88d57ed3ef8b9124b804257ec15d7b4e426400cc74e028f73c949e36b1919aaf823dbc35440538e49151e28ce8c80528f0e0dd21"  
 )  
 pub3 = bytes.fromhex( "048f6a3736d960a861bce581abd4e24f87977dc2804cd3eddec08dbe03804aea562ba30396818f3cf3cfdf0eb0ed8841abb2b3b28510ce89922d275f1f58e707b8"  
 )  
  
 tx = Transaction(private\_key)  
 tx.add\_destination(  
 Destination(tx.address, 0.0076, multi\_sig\_2\_of\_3(pub1, pub2, pub3))  
 )  
 tx.add\_utxo(  
 UnspentTransactionOutput(  
 "15c794bbb169f272fb1ed45526eef5613a851716a1df574480c12f23126772be",  
 0,  
 tx.my\_P2PKH\_script\_pub\_key(),  
 )  
 )  
  
 resp = tx.create()  
 print(f"[{resp.status\_code}] {resp.reason}")  
 print(resp.text)

اطلاعات این تراکنش (انتقال مقدار 0.0076 بیت‌کوین) در [این لینک](https://live.blockcypher.com/btc-testnet/tx/6c9cea1530ef89838e551bf08481ba4193be9f0ed9d8dcdcceff20a645f9e357/) و [این لینک](https://blockstream.info/testnet/tx/6c9cea1530ef89838e551bf08481ba4193be9f0ed9d8dcdcceff20a645f9e357?expand) قابل دسترس است. همچنین هش تراکنش به صورت زیر است:

6c9cea1530ef89838e551bf08481ba4193be9f0ed9d8dcdcceff20a645f9e357

حال باید با استفاده از signature دو تا از سه آدرس ایجاد شده، این مقدار را به آدرس اصلی برگردانیم. در این بخش 0.0074 بیت‌کوین را به آدرس اصلی بازمی‌گردانم و باقی آن را به Transaction Fee اختصاص می‌دهم. این کار توسط کد زیر که در فایل part2\_q2\_2.py قرار دارد، انجام شده است:

def multi\_sig\_2\_of\_3(pub1: bytes, pub2: bytes, pub3: bytes) -> CScript:  
 return CScript([2, pub1, pub2, pub3, 3, OP\_CHECKMULTISIG]) *# type: ignore*  
  
  
def sig\_script\_2\_of\_3(sig1: bytes, sig2: bytes) -> CScript:  
 return CScript([OP\_0, sig1, sig2]) *# type: ignore*  
  
  
def sign(  
 tx: Transaction, utxo\_index: int, private\_key: bitcoin.wallet.CBitcoinSecret  
) -> bytes:  
 txin\_script\_pub\_key = tx.\_utxos[utxo\_index].script\_pub\_key  
 sighash = SignatureHash(txin\_script\_pub\_key, tx.\_tx, 0, SIGHASH\_ALL)  
 return private\_key.sign(sighash) + bytes([SIGHASH\_ALL]) *# type: ignore*  
  
  
def main():  
 private\_key = "92Zh9ENA7DeNBr3FXa1QMLi4igPAzUKy44TEPMW7rogBtGz4CaR"  
  
 bitcoin.SelectParams("testnet")  
 private\_key1 = bitcoin.wallet.CBitcoinSecret(  
 "93RdsGKJn6ExkLEWpVogpC5kRibgoJDWZWUzjonohYhynkScHLX"  
 )  
 public\_key1 = private\_key1.pub  
 private\_key2 = bitcoin.wallet.CBitcoinSecret(  
 "938zdHuD6PVuUb26s31xevjTBPY6fggvrxv9fhJz5UGeAfqF61j"  
 )  
 public\_key2 = private\_key2.pub  
 private\_key3 = bitcoin.wallet.CBitcoinSecret(  
 "91cxgQtoYrMLUcDi3PsRvQnPQa9hgx9jc3keaq3vNXzmiaBFBrW"  
 )  
 public\_key3 = private\_key3.pub  
  
 tx = Transaction(private\_key)  
  
 tx.add\_destination(Destination(tx.address, 0.0074))  
 tx.add\_utxo(  
 UnspentTransactionOutput(  
 "6c9cea1530ef89838e551bf08481ba4193be9f0ed9d8dcdcceff20a645f9e357",  
 0,  
 multi\_sig\_2\_of\_3(public\_key1, public\_key2, public\_key3),  
 )  
 )  
  
 tx.\_create\_transaction()  
  
 sig1 = sign(tx, 0, private\_key1)  
 sig2 = sign(tx, 0, private\_key2)  
 tx.\_utxos[0].\_custom\_sig = sig\_script\_2\_of\_3(sig1, sig2)  
  
 resp = tx.create()  
 print(f"[{resp.status\_code}] {resp.reason}")  
 print(resp.text)

اطلاعات این تراکنش در [این لینک](https://live.blockcypher.com/btc-testnet/tx/804b02bd9c3db8c2c480cca63342a6758d7c91c221028403df82ae3be483dba1/) و [این لینک](https://blockstream.info/testnet/tx/804b02bd9c3db8c2c480cca63342a6758d7c91c221028403df82ae3be483dba1?expand) و همچنین هش زیر قابل دسترس است:

804b02bd9c3db8c2c480cca63342a6758d7c91c221028403df82ae3be483dba1

## اطلاع از 2 عدد اول برای خرج تراکنش

در این بخش با توجه به اینکه scriptPubKey برای همه افراد نمایان می‌شود، اگر حاصل جمع و تفریق اعداد اول را به صورت مستقیم در این اسکریپت بگذاریم، هر فردی می‌تواند دو عدد اول را بدست آورده و تراکنش را خرج کند. به همین دلیل، از خاصیت Preimage Resistance توابع هش استفاده می‌کنیم و هش جمع و تفریق این دو عدد را در اسکریپت قرار می‌دهیم. در واقع اسکریپت مد نظر می‌تواند به صورت زیر باشد:

Script: OP\_2DUP, OP\_ADD, OP\_HASH160, Hash160(SUM), OP\_EQUALVERIFY, OP\_SUB, OP\_HASH160, Hash160(DIFF), OP\_EQUAL

در این حالت کافیست عدد دوم را در top استک و عدد اول را زیر آن قرار دهیم و نتیجه را به عنوان scriptSig استفاده کنیم. در این بخش اعداد اول به صورت زیر انتخاب شده‌اند:

Num1 = 977

Num2 = 881

Sum = 1858

Diff = 96

**نکته:** به دلیل اشتباهی که در scriptPubKey این بخش داشتم، مجددا از Faucet ذکر شده مقداری بیت‌کوین دریافت کردم که اطلاعات این تراکنش در [این لینک](https://live.blockcypher.com/btc-testnet/tx/6c7b5a4c4551bafd06b8279f4a64c445d4b46245a65a38c31c058468909c36ff/) و [این لینک](https://blockstream.info/testnet/tx/6c7b5a4c4551bafd06b8279f4a64c445d4b46245a65a38c31c058468909c36ff?expand) و هش زیر قابل دسترسی است:

6c7b5a4c4551bafd06b8279f4a64c445d4b46245a65a38c31c058468909c36ff

این کار توسط کد زیر که در فایل part2\_q3\_1.py قابل دسترس است، انجام شده است:

PRIME\_NUM1 = 977  
PRIME\_NUM2 = 881  
  
SUM = PRIME\_NUM1 + PRIME\_NUM2  
SUM\_IN\_BYTES = SUM.to\_bytes(2, byteorder="little")  
DIFF = PRIME\_NUM1 - PRIME\_NUM2  
DIFF\_IN\_BYTES = DIFF.to\_bytes(1, byteorder="little")  
  
SCRIPT\_SUM\_DIFF\_PUB\_KEY = CScript([OP\_2DUP, OP\_ADD, OP\_HASH160, Hash160(SUM\_IN\_BYTES), OP\_EQUALVERIFY, OP\_SUB, OP\_HASH160, Hash160(DIFF\_IN\_BYTES), OP\_EQUAL]) *# type: ignore*  
  
  
def main():  
 private\_key = "92Zh9ENA7DeNBr3FXa1QMLi4igPAzUKy44TEPMW7rogBtGz4CaR"  
  
 tx = Transaction(private\_key)  
 tx.add\_destination(Destination(tx.address, 0.015, SCRIPT\_SUM\_DIFF\_PUB\_KEY))  
 tx.add\_utxo(  
 UnspentTransactionOutput(  
 "6c7b5a4c4551bafd06b8279f4a64c445d4b46245a65a38c31c058468909c36ff",  
 1,  
 tx.my\_P2PKH\_script\_pub\_key(),  
 )  
 )  
  
 resp = tx.create()  
 print(f"[{resp.status\_code}] {resp.reason}")  
 print(resp.text)

لازم به ذکر است که حاصل جمع یک عدد 2 بایتی و حاصل تفریق آن‌ها یک عدد 1 بایتی است.

اطلاعات این تراکنش در [این لینک](https://live.blockcypher.com/btc-testnet/tx/e2b73c7d033a2f672b65f64c68322aced2bccc4db44e7cd7a355dcf7ba0b2955/) و [این لینک](https://blockstream.info/testnet/tx/e2b73c7d033a2f672b65f64c68322aced2bccc4db44e7cd7a355dcf7ba0b2955?expand) و همچنین هش زیر قابل دسترس است:

e2b73c7d033a2f672b65f64c68322aced2bccc4db44e7cd7a355dcf7ba0b2955

حال باید این تراکنش را به حساب خودمان بازگردانیم. در این حالت اسکریپت Sig به صورت زیر است:

Script: 977, 881

این کار توسط کد زیر که در فایل part2\_q3\_2.py قرار دارد انجام شده است:

PRIME\_NUM1 = 977  
PRIME\_NUM1\_IN\_BYTES = PRIME\_NUM1.to\_bytes(2, byteorder="little")  
PRIME\_NUM2 = 881  
PRIME\_NUM2\_IN\_BYTES = PRIME\_NUM2.to\_bytes(2, byteorder="little")  
  
SUM = PRIME\_NUM1 + PRIME\_NUM2  
SUM\_IN\_BYTES = SUM.to\_bytes(2, byteorder="little")  
DIFF = PRIME\_NUM1 - PRIME\_NUM2  
DIFF\_IN\_BYTES = DIFF.to\_bytes(1, byteorder="little")  
  
SCRIPT\_SUM\_DIFF\_PUB\_KEY = CScript([OP\_2DUP, OP\_ADD, OP\_HASH160, Hash160(SUM\_IN\_BYTES), OP\_EQUALVERIFY, OP\_SUB, OP\_HASH160, Hash160(DIFF\_IN\_BYTES), OP\_EQUAL]) *# type: ignore*  
SCRIPT\_PRIME\_NUMS\_SIG = CScript([PRIME\_NUM1\_IN\_BYTES, PRIME\_NUM2\_IN\_BYTES]) *# type: ignore*  
  
  
def main():  
 private\_key = "92Zh9ENA7DeNBr3FXa1QMLi4igPAzUKy44TEPMW7rogBtGz4CaR"  
  
 tx = Transaction(private\_key)  
 tx.add\_destination(Destination(tx.address, 0.014))  
 tx.add\_utxo(  
 UnspentTransactionOutput(  
 "e2b73c7d033a2f672b65f64c68322aced2bccc4db44e7cd7a355dcf7ba0b2955",  
 0,  
 SCRIPT\_SUM\_DIFF\_PUB\_KEY,  
 SCRIPT\_PRIME\_NUMS\_SIG,  
 )  
 )  
  
 resp = tx.create()  
 print(f"[{resp.status\_code}] {resp.reason}")  
 print(resp.text)

اطلاعات این تراکنش در [این لینک](https://live.blockcypher.com/btc-testnet/tx/65f47ef01f3145ed2993f570e9c573e3a7299cd7a27a2bff295d1d7346638965/) و [این لینک](https://blockstream.info/testnet/tx/65f47ef01f3145ed2993f570e9c573e3a7299cd7a27a2bff295d1d7346638965?expand) و همچنین هش زیر قابل دسترس است:

65f47ef01f3145ed2993f570e9c573e3a7299cd7a27a2bff295d1d7346638965

# قسمت سوم: استخراج بلوک

ابتدا باید یک آدرس مخصوص Mainnet داشته باشیم که به صورت زیر است:

Private Key (WIF): 5JWoEUpPb1BCRMTYUqNNq4L7eEAptfiz9FKsBAj7niAJWaQ6uZJ

Public Key: 04bd113d6628b7a3054293dcbfa4d0e98af6cdf1d0f1d519ce7e1033b43a  
cd8e8f609a0191c30b2c614408f59f00ac3f5f028974216b0710d611073dd141909bc8

Address: 1Ny5UQ4B6XRuyuB8BPgUPvCXDdR3WV9xd5

با توجه به شماره دانشجویی‌ام، باید از بلاک 9385 استفاده کنم که هش آن به صورت زیر است:

00000000673405ffe87f801032e901c7f423adddc7b51773e6b108e617e75516

حال برای تولید تراکنش coinbase، از یک کلاس که از کلاس Transaction ارث می‌برد استفاده می‌کنم:

class BaseCoinTransaction(Transaction):  
 def \_\_init\_\_(  
 self,  
 private\_key: str,  
 data: str,  
 network: Transaction.Network = Transaction.Network.MAINNET,  
 ):  
 super().\_\_init\_\_(private\_key, network)  
 self.\_data = data  
  
 self.\_destinations.append(Destination(self.address, BITCOIN\_MINE\_AWARD))  
 self.\_utxos.append(UnspentTransactionOutput("0" \* 64, 0xFFFFFFFF, CScript([]), self.\_get\_coinbase\_sig())) *# type: ignore*  
  
 def \_get\_coinbase\_sig(self) -> CScript:  
 hex\_data = self.\_data.encode("utf-8").hex()  
 return CScript([bytes.fromhex(hex\_data)]) *# type: ignore*  
  
 def create(self) -> CMutableTransaction:  
 self.\_create\_transaction()  
 self.\_tx.vin[0].scriptSig = self.\_utxos[0].custom\_sig  
 return self.\_tx  
  
 def \_broadcast\_transaction(self) -> Response:  
 raise NotImplementedError("BaseCoinTransaction cannot be broadcasted")  
  
 def \_verify(self) -> None:  
 raise NotImplementedError("BaseCoinTransaction cannot be verified")

در اینجا مقدار BITCOIN\_MINE\_AWARD برابر با 6.25 بیت‌کوین در نظر گرفته شده است.

داده مدنظر (810199385PashaBarahimi) در scriptSig قرار می‌گیرد.

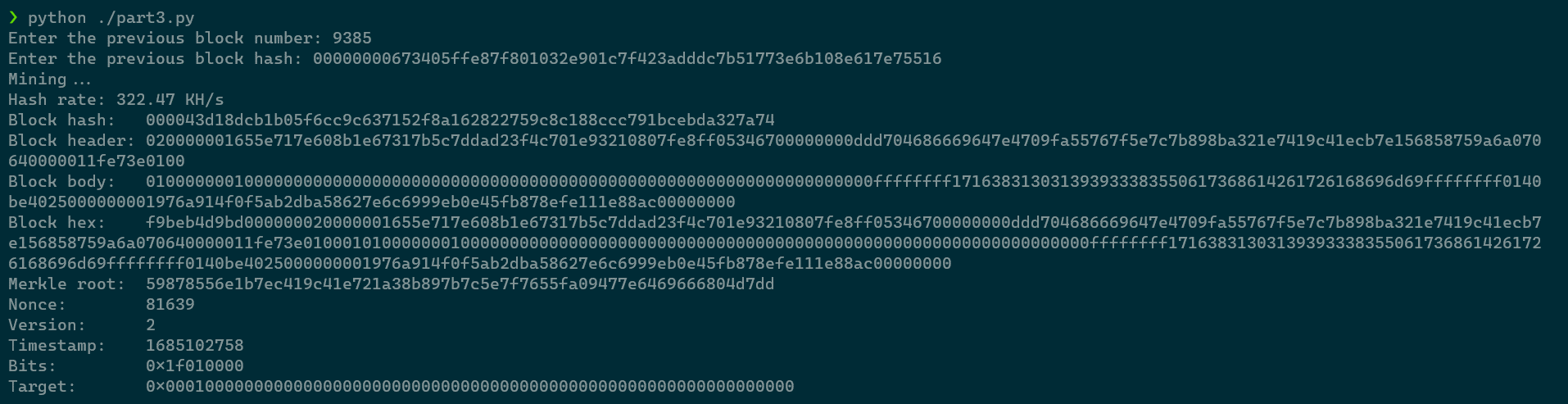
همانطور که در منابع اشاره شده، داده‌ها در بلاک بیت‌کوین به صورت little endian ذخیره می‌شوند و به همین دلیل این مورد در کد نیز رعایت شده است. کد زیر برای mine کردن بلاک نوشته شده است (این کد در فایل part3.py در دسترس است):

def main():  
 data = "810199385PashaBarahimi"  
 private\_key = "5JWoEUpPb1BCRMTYUqNNq4L7eEAptfiz9FKsBAj7niAJWaQ6uZJ"  
 bits = "0x1f010000" *# 16 bits leading 0s*  
 timestamp = int(time.time())  
 \_ = int(input("Enter the previous block number: ")) *# unused*  
 prev\_hash = input("Enter the previous block hash: ")  
  
 basecoin = BaseCoinTransaction(private\_key, data, Transaction.Network.MAINNET)  
 tx = basecoin.create()  
  
 block = BitcoinBlock([tx], prev\_hash, bits, timestamp)  
 print("Mining...")  
 hash\_value = block.mine()  
  
 print(f"Block hash: {b2lx(hash\_value)}")  
 print(f"Block header: {b2x(block.header)}")  
 print(f"Block body: {b2x(block.body)}")  
 print(f"Block hex: {b2x(block.block)}")  
  
 print(f"Merkle root: {block.merkle\_root}")  
 print(f"Nonce: {block.nonce}")  
 print(f"Version: {block.version}")  
 print(f"Timestamp: {timestamp}")  
 print(f"Bits: {bits}")  
 print(f"Target: {block.target}")

پیاده‌سازی کلاس BitcoinBlock نیز به صورت زیر است:

class BitcoinBlock:  
 def \_\_init\_\_(  
 self,  
 transactions: list[CMutableTransaction],  
 prev\_block\_hash: str,  
 bits: str = "0x1f010000",  
 timespamp: int = int(time.time()),  
 ):  
 self.\_transactions = transactions  
 self.\_prev\_block\_hash = prev\_block\_hash  
 self.\_merkle\_root = self.\_calculate\_merkle\_root()  
 self.\_timestamp = timespamp  
 self.\_bits = int(bits, 16)  
 self.\_target = self.\_get\_target(bits)  
 self.\_version = 2  
 self.\_nonce = 0  
 self.\_partial\_header = self.\_get\_partial\_header()  
 self.\_header = self.\_partial\_header  
 self.\_body = self.\_get\_body()  
  
 *# some properties*  
  
 def \_calculate\_merkle\_root(self) -> str:  
 hashes = [Hash(tx.serialize()) for tx in self.\_transactions]  
 while len(hashes) > 1:  
 if len(hashes) % 2 != 0:  
 hashes.append(hashes[-1])  
 hashes = [  
 Hash(hash1 + hash2) for hash1, hash2 in zip(hashes[::2], hashes[1::2])  
 ]  
 return b2lx(hashes[0])  
  
 @staticmethod  
 def \_get\_target(bits: str) -> bytes:  
 exponent = bits[2:4]  
 coefficient = bits[4:]  
 target = int(coefficient, 16) \* 2 \*\* (8 \* (int(exponent, 16) - 3))  
 target\_hex = hex(target)[2:]  
 return bytes.fromhex(target\_hex.zfill(64))  
  
 def \_get\_hash\_value(self) -> bytes:  
 nonce = struct.pack("<L", self.\_nonce)  
 self.\_header = self.\_partial\_header + nonce  
 return Hash(self.\_header)  
  
 def \_get\_partial\_header(self) -> bytes:  
 return (  
 struct.pack("<L", self.\_version)  
 + bytes.fromhex(self.\_prev\_block\_hash)[::-1]  
 + bytes.fromhex(self.\_merkle\_root)[::-1]  
 + struct.pack("<LL", self.\_timestamp, self.\_bits)  
 )  
  
 def \_get\_body(self) -> bytes:  
 return b"".join(tx.serialize() for tx in self.\_transactions)  
  
 def \_print\_hash\_rate(self, start: float, end: bool = False) -> None:  
 elapsed\_time = time.time() - start  
 rate = self.\_nonce / elapsed\_time  
 if rate < 1e3:  
 unit = "H/s"  
 elif rate < 1e6:  
 rate /= 1e3  
 unit = "KH/s"  
 elif rate < 1e9:  
 rate /= 1e6  
 unit = "MH/s"  
 else:  
 rate /= 1e9  
 unit = "GH/s"  
 print(f"\r{' ' \* 20}\rHash rate: {rate:.2f} {unit}", end="")  
 if end:  
 print()  
  
 def mine(self) -> bytes:  
 start = time.time()  
 while self.\_nonce < 2\*\*32:  
 hash\_value = self.\_get\_hash\_value()  
 if hash\_value[::-1] < self.\_target:  
 self.\_print\_hash\_rate(start, end=True)  
 return hash\_value  
 self.\_nonce += 1  
 if self.\_nonce % 1000 == 0:  
 self.\_print\_hash\_rate(start)  
 raise ValueError("Nonce overflow")

نتیجه اجرای کد نیز به صورت زیر است:



Block Header: 020000001655e717e608b1e67317b5c7ddad23f4c701e93210807fe8ff0  
5346700000000ddd704686669647e4709fa55767f5e7c7b898ba321e7419c41ecb7e156858759a6a070640000011fe73e0100

Block Body (Transactions): 0100000001000000000000000000000000000000000000000  
0000000000000000000000000ffffffff171638313031393933383550617368614261726168696d69ffffffff0140be4025000000001976a914f0f5ab2dba58627e6c6999eb0e45fb878efe111e88ac00000000

Hash: 000043d18dcb1b05f6cc9c637152f8a162822759c8c188ccc791bcebda327a74

داده ذخیره شده (810199385PashaBarahimi) به صورت hex در بخش زیر نمایش داده شده است:

810199385PashaBarahimi: 0x38313031393933383550617368614261726168696D69