# قسمت اول: تولید آدرس

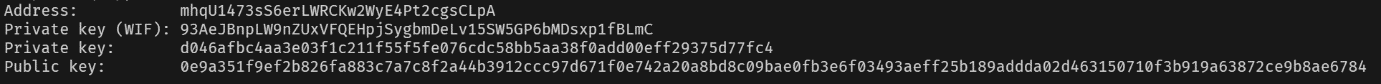
## تولید آدرس عادی

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  
  
 *# some properties*  
  
 def generate(self) -> None:  
 self.\_private\_key = os.urandom(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'  
 elif self.network == Wallet.Network.TESTNET:  
 return b'\xef'  
 else:  
 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 = public\_key.to\_string()  
  
 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: mhqU1473sS6erLWRCKw2WyE4Pt2cgsCLpA

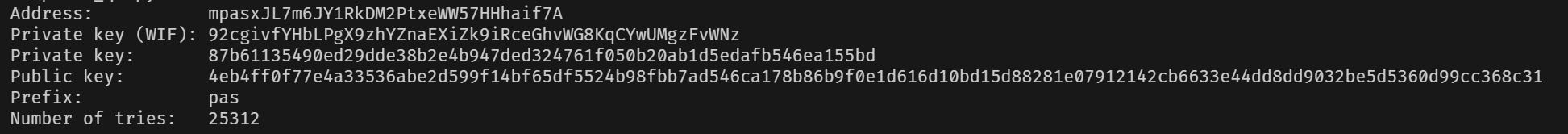
Private Key (WIF): 93AeJBnpLW9nZUxVFQEHpjSygbmDeLv15SW5GP6bMDsxp1fBLmC

## 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: mpasxJL7m6JY1RkDM2PtxeWW57HHhaif7A

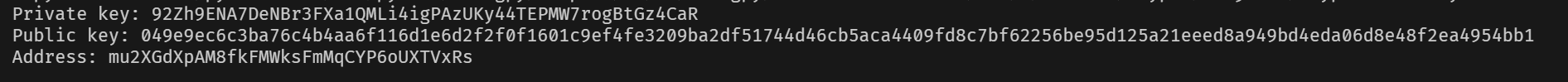
Private Key (WIF): 92cgivfYHbLPgX9zhYZnaEXiZk9iRceGhvWG8KqCYwUMgzFvWNz

# 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

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, destinations: Destination) -> None:  
 self.\_destinations.append(destinations)  
  
 def add\_utxo(self, utxos: UnspentTransactionOutput) -> None:  
 self.\_utxos.append(utxos)  
  
 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