한 번에 끝내는 블록체인 개발 A to Z

Chapter 1

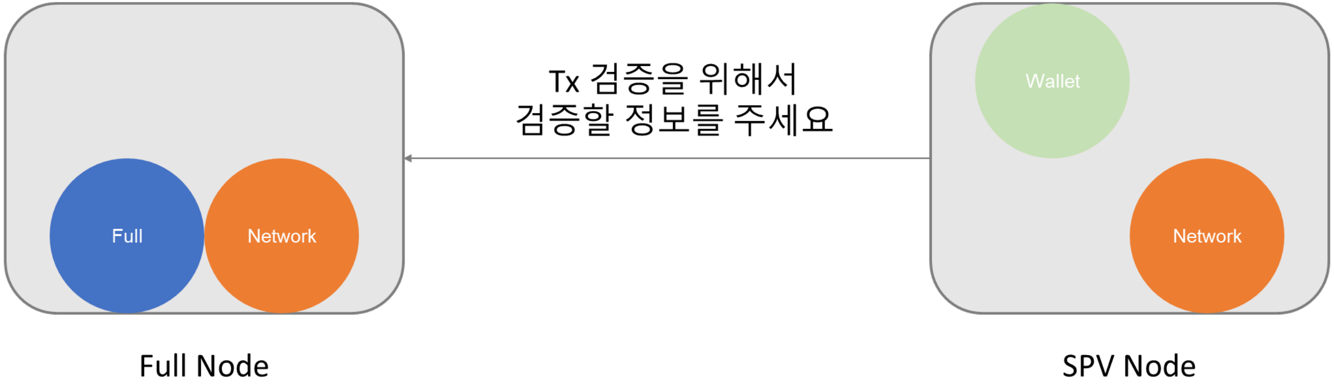
Blockchain 1.0 - Bitcoin

Chapter 1 Blockchain 1.0 - Bitcoin

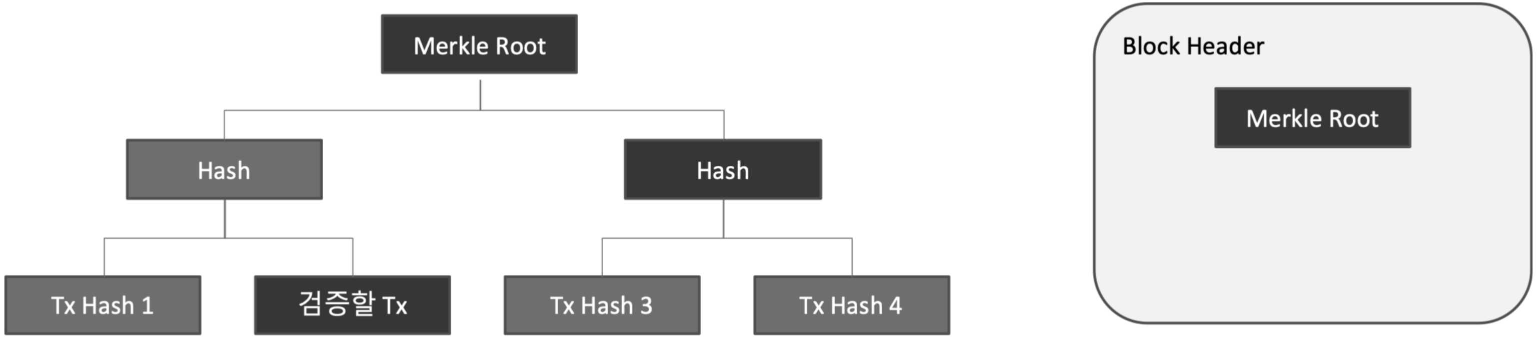
SPV와
Bloom Filter

SPV

- Bitcoin Blockchain Size가 커짐에 따라 이를 저장하기 힘든 Light-Weight 노드 IoT 기기, 스마트폰 등에 Node 설치를 위해 나온 Node 운영 방안
- Full Node로 부터 Merkle Tree와 Block Header만을 전송 받아 Transaction 검증

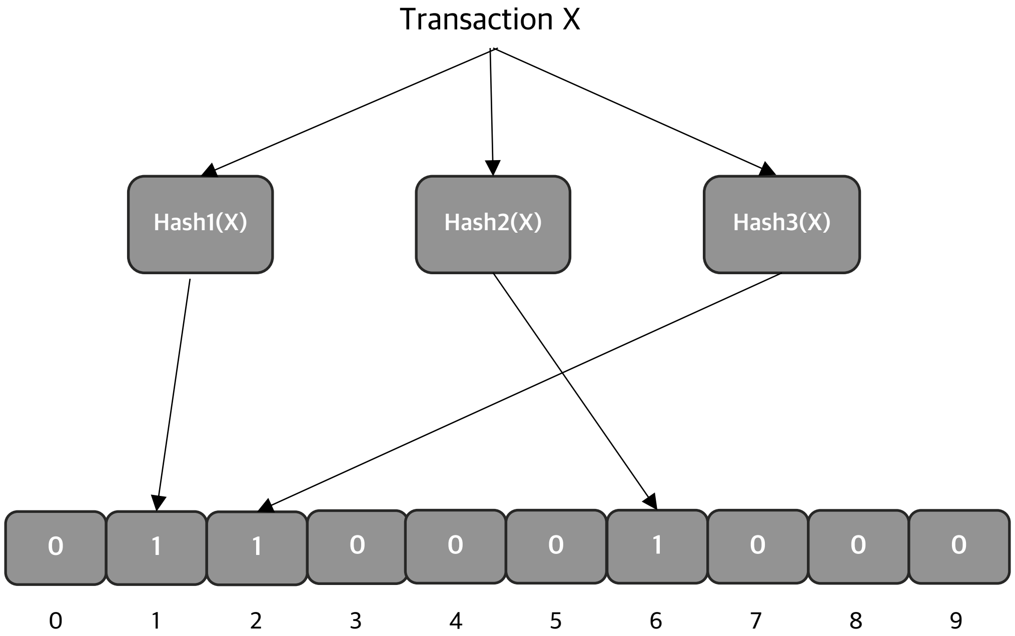


Merkle Pass


Merkle Root
Hash
Hash
Tx Hash 1
검증할 Tx
Tx Hash 3
Tx Hash 4
Block Header
Merkle Root

Bloom Filter

Bloom Filter란 SPV 노드가 Bitcoin Full Node에게 관심있는 Transaction을 전달할 때, 내가 관심있는 Transaction을 숨기고 그 정보를 전달 받는 방법을 위한 기법



Transaction X

Hash1(X)

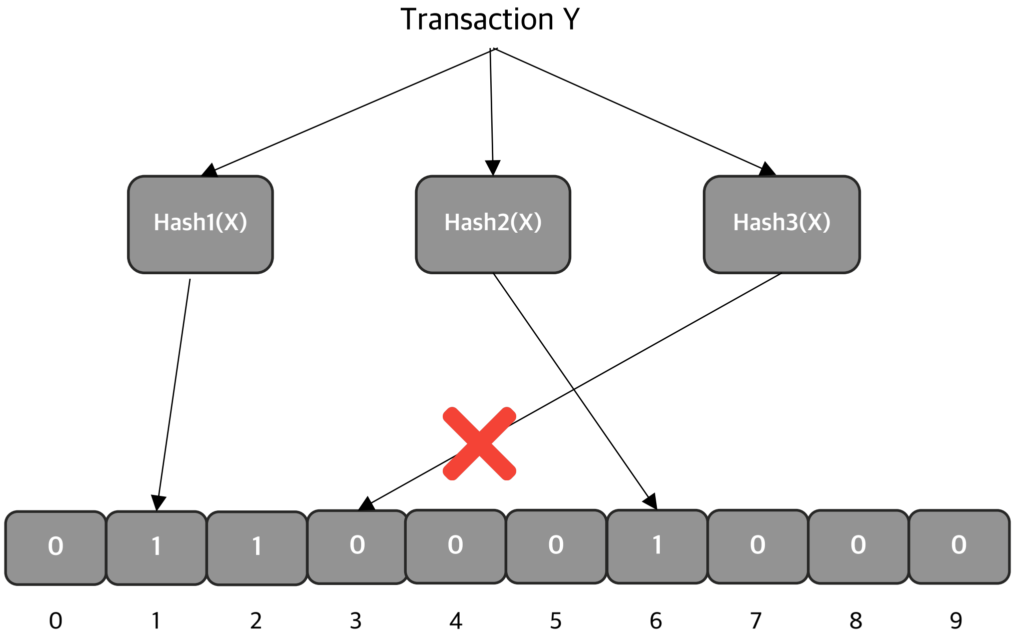
Hash2(X)

Hash3(X)

0	1	2	3	4	5	6	7	8	9
0	1	1	0	0	0	1	0	0	0

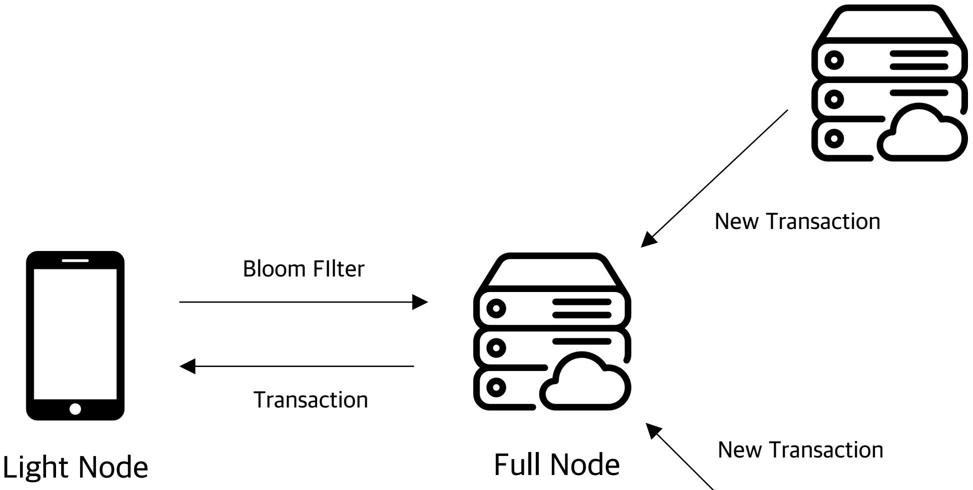
Bloom Filter

Bloom Filter는 False Positive 통해 관심이 없는 Transaction을 Client에서 제외시킬 수 있는 기능이다.

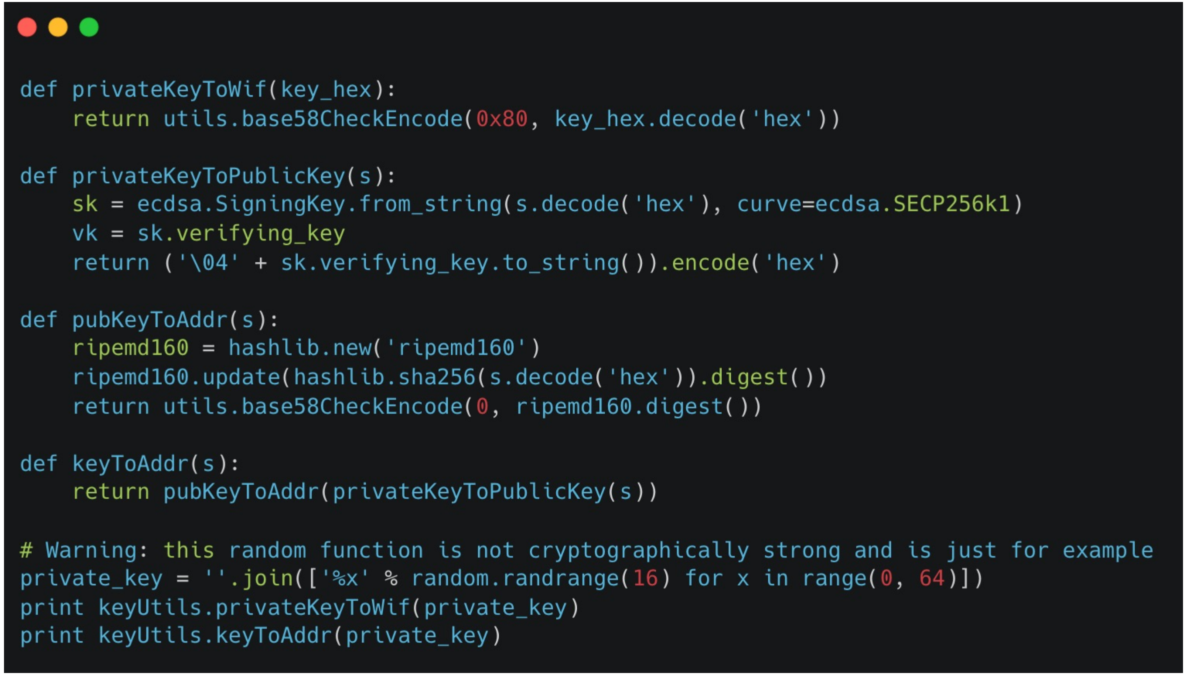


Bloom Filter

Bloom Filter는 Light Node가 Full Node에게 전달하여 Light Node가 관심있는 거래 정보를 실제로 드러내지 않고 Transaction 전달이 가능하도록 한다.

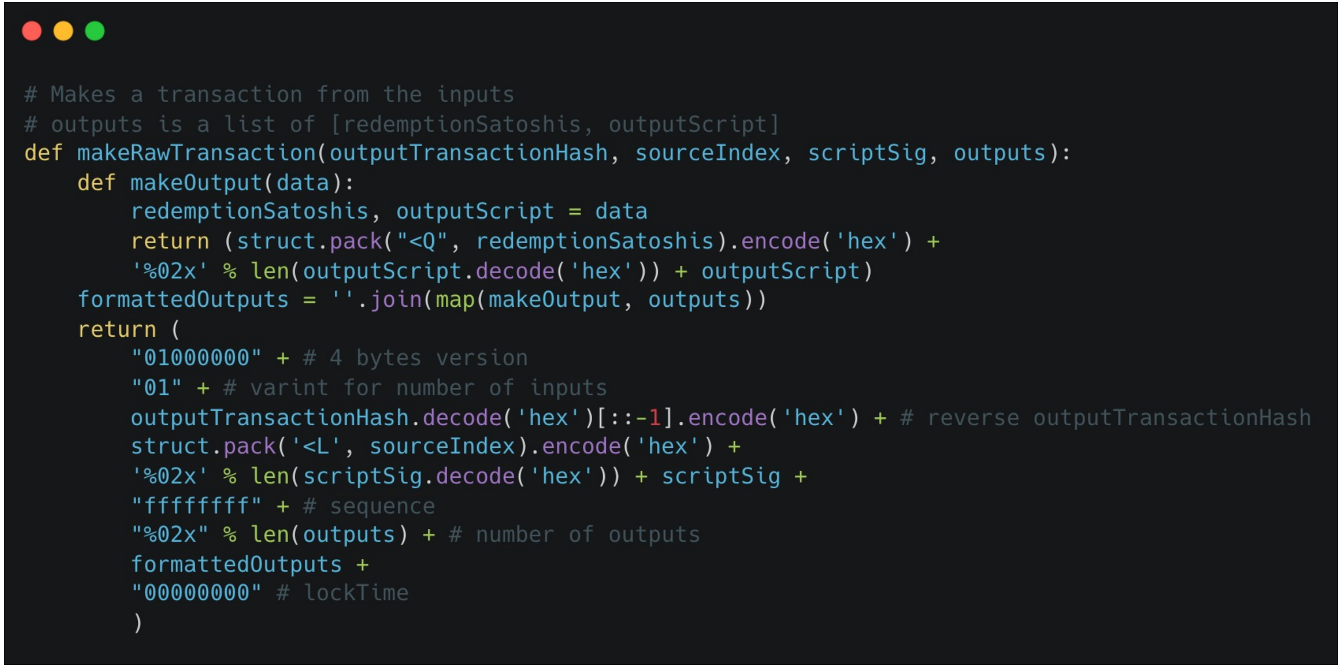


Wallet Source Code



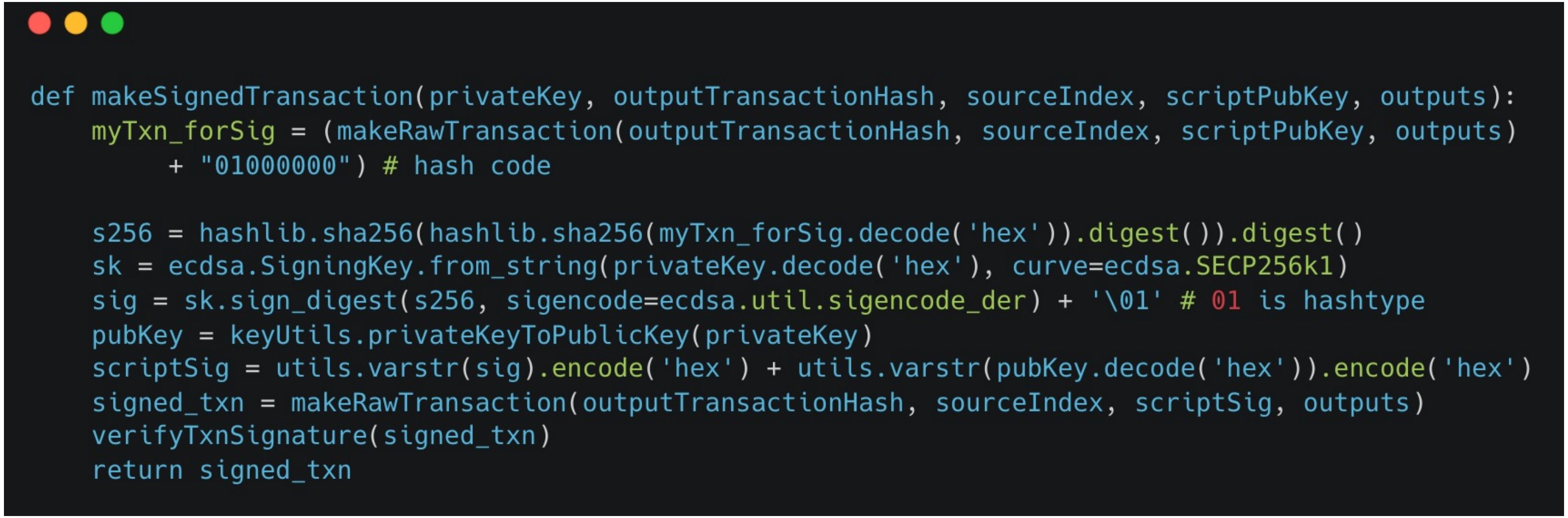
```python
def privateKeyToWif(key_hex):
    return utils.base58CheckEncode(0x80, key_hex.decode('hex'))

def privateKeyToPublicKey(s):
    sk = ecdsa.SigningKey.from_string(s.decode('hex'), curve=ecdsa.SECP256k1)
    vk = sk.verifying_key
    return ('\04' + sk.verifying_key.to_string()).encode('hex')

def pubKeyToAddr(s):
    ripemd160 = hashlib.new('ripemd160')
    ripemd160.update(hashlib.sha256(s.decode('hex')).digest())
    return utils.base58CheckEncode(0, ripemd160.digest())

def keyToAddr(s):
    return pubKeyToAddr(privateKeyToPublicKey(s))

# Warning: this random function is not cryptographically strong and is just for example
private_key = ''.join(['%x' % random.randrange(16) for x in range(0, 64)])
print keyUtils.privateKeyToWif(private_key)
print keyUtils.keyToAddr(private_key)
```

Wallet Source Code



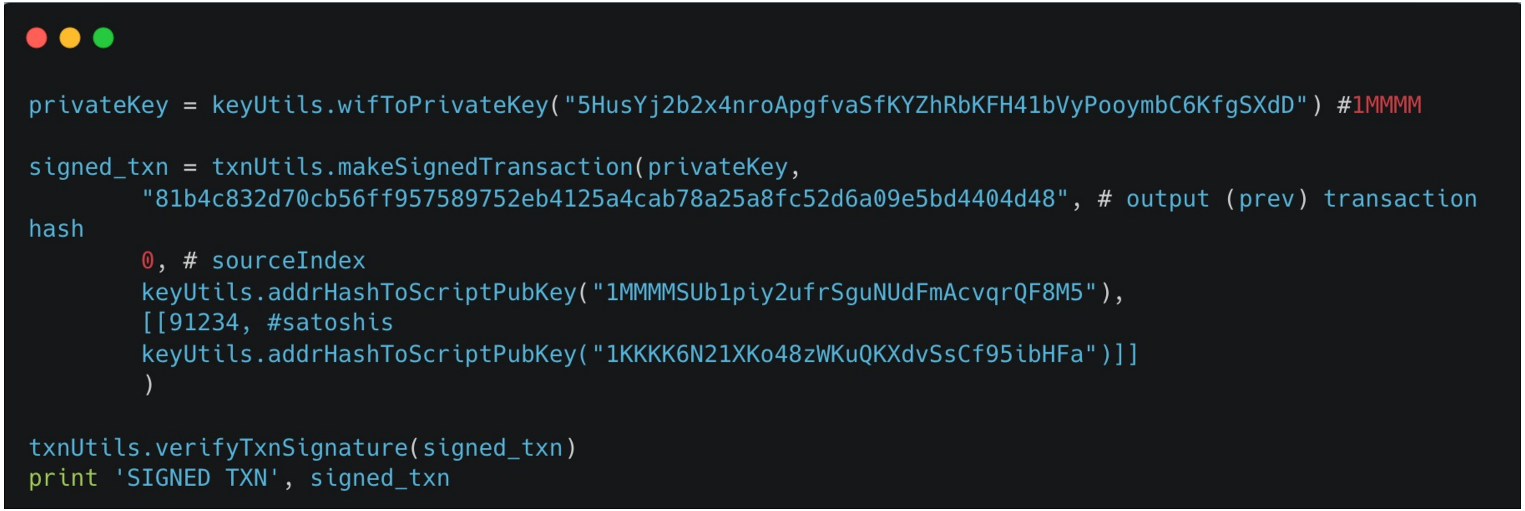
```
# Makes a transaction from the inputs
# outputs is a list of [redemptionSatoshis, outputScript]
def makeRawTransaction(outputTransactionHash, sourceIndex, scriptSig, outputs):
    def makeOutput(data):
        redemptionSatoshis, outputScript = data
        return (struct.pack("<Q", redemptionSatoshis).encode('hex') +
        '%02x' % len(outputScript.decode('hex')) + outputScript)
    formattedOutputs = ''.join(map(makeOutput, outputs))
    return (
        "01000000" + # 4 bytes version
        "01" + # varint for number of inputs
        outputTransactionHash.decode('hex')[::-1].encode('hex') + # reverse outputTransactionHash
        struct.pack('<L', sourceIndex).encode('hex') +
        '%02x' % len(scriptSig.decode('hex')) + scriptSig +
        "ffffffff" + # sequence
        "%02x" % len(outputs) + # number of outputs
        formattedOutputs +
        "00000000" # lockTime
        )
```

Wallet Source Code



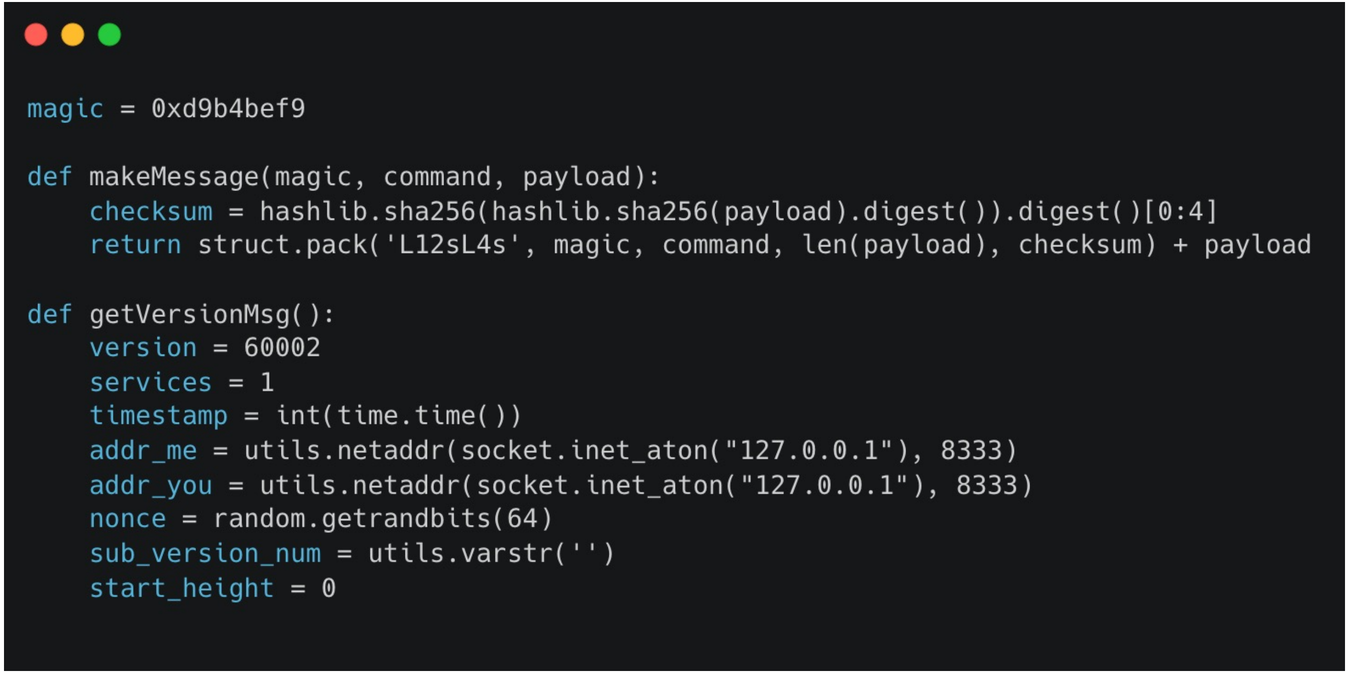
```python
def makeSignedTransaction(privateKey, outputTransactionHash, sourceIndex, scriptPubKey, outputs):
    myTxn_forSig = (makeRawTransaction(outputTransactionHash, sourceIndex, scriptPubKey, outputs)
         + "01000000") # hash code

    s256 = hashlib.sha256(hashlib.sha256(myTxn_forSig.decode('hex')).digest()).digest()
    sk = ecdsa.SigningKey.from_string(privateKey.decode('hex'), curve=ecdsa.SECP256k1)
    sig = sk.sign_digest(s256, sigencode=ecdsa.util.sigencode_der) + '\01' # 01 is hashtype
    pubKey = keyUtils.privateKeyToPublicKey(privateKey)
    scriptSig = utils.varstr(sig).encode('hex') + utils.varstr(pubKey.decode('hex')).encode('hex')
    signed_txn = makeRawTransaction(outputTransactionHash, sourceIndex, scriptSig, outputs)
    verifyTxnSignature(signed_txn)
    return signed_txn
```

Wallet Source Code



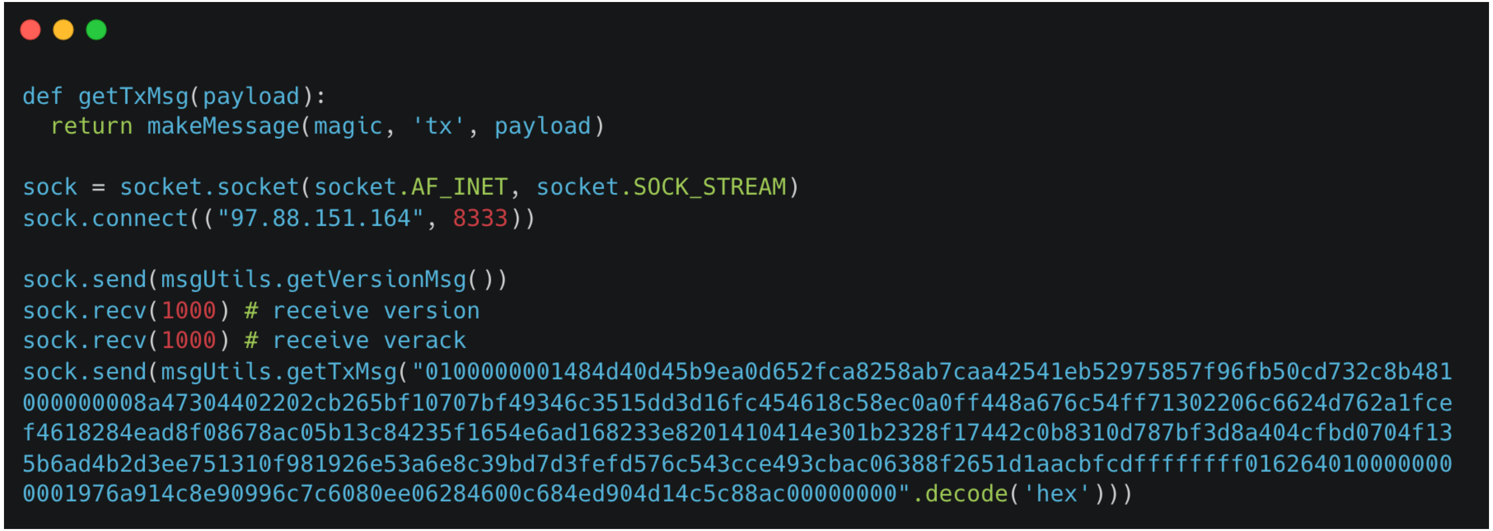
```
privateKey = keyUtils.wifToPrivateKey("5HusYj2b2x4nroApgfvaSfKYZhRbKFH41bVyPooymbC6KfgSXdD") #1MMMM

signed_txn = txnUtils.makeSignedTransaction(privateKey,
        "81b4c832d70cb56ff957589752eb4125a4cab78a25a8fc52d6a09e5bd4404d48", # output (prev) transaction
hash
        0, # sourceIndex
        keyUtils.addrHashToScriptPubKey("1MMMMSUb1piy2ufrSguNUdFmAcvqrQF8M5"),
        [[91234, #satoshis
        keyUtils.addrHashToScriptPubKey("1KKKK6N21XKo48zWKuQKXdvSsCf95ibHFa")]]
        )

txnUtils.verifyTxnSignature(signed_txn)
print 'SIGNED TXN', signed_txn
```

Wallet Source Code



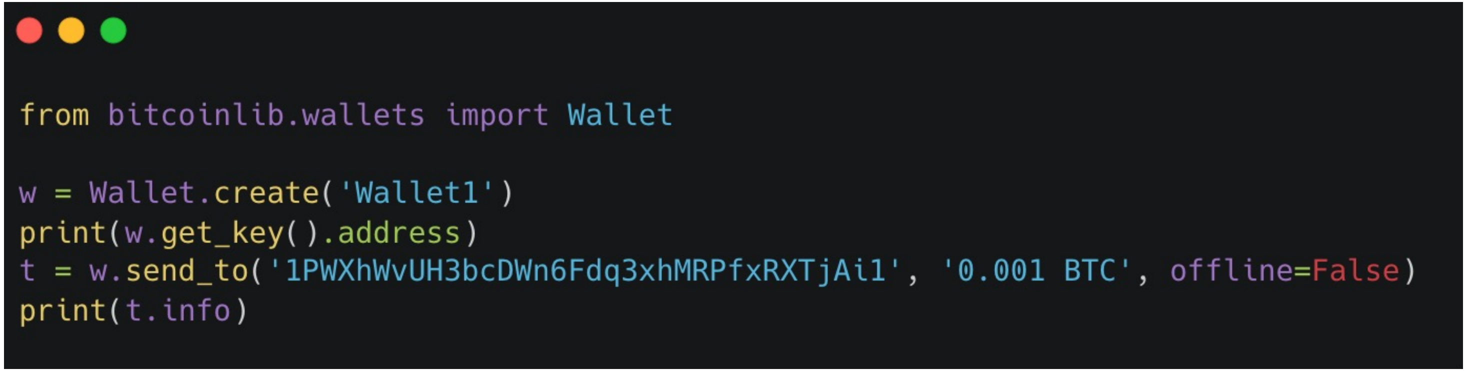
```
magic = 0xd9b4bef9

def makeMessage(magic, command, payload):
    checksum = hashlib.sha256(hashlib.sha256(payload).digest()).digest()[0:4]
    return struct.pack('L12sL4s', magic, command, len(payload), checksum) + payload

def getVersionMsg():
    version = 60002
    services = 1
    timestamp = int(time.time())
    addr_me = utils.netaddr(socket.inet_aton("127.0.0.1"), 8333)
    addr_you = utils.netaddr(socket.inet_aton("127.0.0.1"), 8333)
    nonce = random.getrandbits(64)
    sub_version_num = utils.varstr('')
    start_height = 0
```

Wallet Source Code



```
def getTxMsg(payload):
  return makeMessage(magic, 'tx', payload)

sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
sock.connect(("97.88.151.164", 8333))

sock.send(msgUtils.getVersionMsg())
sock.recv(1000) # receive version
sock.recv(1000) # receive verack
sock.send(msgUtils.getTxMsg("0100000001484d40d45b9ea0d652fca8258ab7caa42541eb52975857f96fb50cd732c8b481
000000008a47304402202cb265bf10707bf49346c3515dd3d16fc454618c58ec0a0ff448a676c54ff71302206c6624d762a1fce
f4618284ead8f08678ac05b13c84235f1654e6ad168233e8201410414e301b2328f17442c0b8310d787bf3d8a404cfbd0704f13
5b6ad4b2d3ee751310f981926e53a6e8c39bd7d3fefd576c543cce493cbac06388f2651d1aacbfcdffffffff016264010000000
0001976a914c8e90996c7c6080ee06284600c684ed904d14c5c88ac00000000".decode('hex')))
```

Wallet Source Code



```
from bitcoinlib.wallets import Wallet

w = Wallet.create('Wallet1')
print(w.get_key().address)
t = w.send_to('1PWXhWvUH3bcDWn6Fdq3xhMRPfxRXTjAi1', '0.001 BTC', offline=False)
print(t.info)
```