```python
import hashlib
import datetime
from crypto_key import generate_key
from crypto_key import sign
from crypto_key import verify_sig

class Transaction:
    def __init__(self, data):
        #data["Timestamp"] = str(datetime.datetime.now())
        data["Timestamp"] = ""
        data["hash"] = self.json_digest(data)
        self._data = data

    def sign(self, alias, password):
        self._data["signatures"].append(sign(self.digest(), alias, password))

    def json_digest(self, data):
        m = hashlib.sha256()
        if data["type"] == "BaseCoins":
                m.update(
                str(data["type"]).encode('utf-8') +
                str(data["coins_created"]).encode('utf-8') +
                str(data["Timestamp"]).encode('utf-8')
                )
        else:
                m.update(
                str(data["type"]).encode('utf-8') +
                str(data["coins_consumed"]).encode('utf-8') +
                str(data["coins_created"]).encode('utf-8') +
                str(data["Timestamp"]).encode('utf-8')
                )
        return m.hexdigest()

    def digest(self):
        m = hashlib.sha256()
        if self._data["type"] == "BaseCoins":
                m.update (str(self._data["type"]))
                m.update (str(self._data["coins_consumed"]))
                m.update (str(self._data["coins_created"]))
        else:
                m.update (str(self._data["type"]))
                m.update (str(self._data["coins_created"]))
        return bytearray(m.hexdigest(), 'utf-8')

    def get_value_alias(self, coin_id):
        for e in self._data["coins_created"]:
                if e["num"] == coin_id:
                        return [e["value"], e["recipient"]]
        return 0

    def get_sum_coin_create(self):
        sum = 0
        for e in self._data["coins_created"]:
                sum = sum + e["value"]
        return sum

    def verify_sig_trans(self, alias):
        content = self.digest()
        for e in self._data["signatures"]:
                if verify_sig(content, e, alias):
                        return True
        return False

class Block:
        _blockNo = 0
        _prev = None
```

```
        _prev_hash = bytearray(256)
        _nonce = 0
        _trans = []
        _timestamp = datetime.datetime.now()

        def __init__ (self, trans, prev = None):
                self._blockNo = prev._blockNo + 1 if prev else 0
                self._trans = trans
                self._prev = prev
                self._prev_hash = prev.digest() if prev is not None else bytearray
(256)

        def digest(self):
                m = hashlib.sha256()
                m.update(
                str(self._blockNo).encode('utf-8') +
                str(self._nonce).encode('utf-8') +
                str(self._trans).encode('utf-8') +
                str(self._prev_hash).encode('utf-8') +
                str(self._timestamp).encode('utf-8')
                )
                return m.hexdigest()

        def mining_digest(self):
                m = hashlib.sha256()
                m.update(
                str(self._nonce).encode('utf-8') +
                str(self._trans).encode('utf-8') +
                str(self._prev_hash).encode('utf-8')
                )
                return m.hexdigest()

        def search_coin_created(self, trans_hash, coin_id):
                for e in self._trans:
                        if e._data["hash"] == trans_hash:
                                return e.get_value_alias(coin_id)
                return None

        def verify(self, root_hash):
                my_hash = self.digest()
                if (root_hash != my_hash):
                        print ("Hash does not verify for block containing",
self._trans._data)
                return (root_hash == my_hash and (not self._prev or
self._prev.verify (self._prev_hash)))

        def str_trans(self):
                if self._trans == []:
                        return "Generic block"
                s = ""
                n = 0
                for e in  self._trans:
                        s = s + "\ntrs " + str(n) + ": " + str(e._data)
                        n += 1
                return s


        def str(self):
                return "Block Number: " + str(self._blockNo) + "\nHash: " + str
(self.digest()) + "\nTransactions: " + self. str_trans() + "\nTimestamp: " + str
(self._timestamp)

class SBitcoin:
        _diff = 0
        _maxNonce = 2 ** 32
        _target = 2 ** (256 - _diff)
```

```
        _pending_pool = []

        def __init__(self):
                self._head = Block([], None)
                self._root_hash = self._head.digest()

        def add_transaction(self, trans):
                self._pending_pool.append(trans)

        def __add_block(self, block, alias):
                json = {"hash": 0,
                        "type": "BaseCoins",
                        "coins_created": [
                                {"num": 0, "value": 20, "recipient": alias}
                        ],
                        "Timestamp": ""}
                award = Transaction(json)
                block._trans.append(award)
                block._timestamp = datetime.datetime.now()
                self._head = block
                self._root_hash = self._head.digest()
                self._pending_pool = []

        def mine(self, alias):
                valid_transaction = self.collect_transaction(self._pending_pool)
                block = Block(valid_transaction, self._head)
                for n in xrange(self._maxNonce):
                        if int(block.digest(), 16) <= self._target:
                                self.__add_block(block, alias)
                                break
                        else:
                                block._nonce += 1

        def collect_transaction(self, trans_pool):
                valid_trans = []
                for e in trans_pool:
                        if e._data["type"] == "BaseCoins":

                                print str(e._data) + ": only miners can get a
reward as 20 coins"
                        else:
                                if e._data["type"] == "PayCoins":
                                        if self.verify_trans(e):
                                                valid_trans.append(e)
                                        else:
                                                print "The transaction" + str
(e._data) +"is not valid"
                                else:
                                        print "Unknown transaction"
                return valid_trans


        def get_coin_owner(self, trans_hash, coin_id):
                pointer = self._head
                while pointer:
                        owner = pointer.search_coin_created(trans_hash, coin_id)
                        if ( owner != None):
                                return owner

                        else:
                                pointer = pointer._prev
                return None

        def check_double_spending(self, comsumed):
                pointer = self._head
                while pointer:
```

```
                        for e in comsumed:
                                for m in pointer._trans:
                                        if m._data["type"] == "PayCoins":
                                                if (e in m._data
["coins_consumed"]):
                                                        return
True
                        pointer = pointer._prev
                return False


        def verify_trans(self, trans):
                if self.check_double_spending(trans._data["coins_consumed"]):
                        print "Double spending"
                        return False
                sum_coin = 0
                for e in trans._data["coins_consumed"]:
                        p = self.get_coin_owner(e["hash"], int (e["num"]))
                        if p == None:
                                print "The coin is not exsiting"
                                return False
                        else:
                                sum_coin = sum_coin + float (p[0])
                                if not trans.verify_sig_trans(p[1]):
                                        print "the signature is not valid"
                                        return False
                return sum_coin >= trans.get_sum_coin_create()


        def print_pending_pool(self):
                print "\nSBitcoin-Pending pool"
                if self._pending_pool == []:
                        print "\n[]"
                s = ""
                n = 0
                for e in self._pending_pool:
                        s = s + "\ntrs " + str(n) + ": " + str(e._data)
                        n += 1
                print s

        def print_chain(self):
                pointer = self._head
                print "\nSBitcoin\n"
                while pointer:
                        print pointer.str()
                        pointer = pointer._prev

print "\n*******Generic Block********************\n"
sbitcoin = SBitcoin()
sbitcoin.print_pending_pool()
sbitcoin.print_chain()

#1. Generate three new account Alice, Bob, Caleb and marry
Alice = generate_key("alice", "a")
Bob = generate_key("bob", "b")
Caleb = generate_key("caleb", "c")
Marry = generate_key("marry", "m")

#2. Alice mines a new block
print "\n************#2*************************\n"
#sbitcoin.mine("alice")
#sbitcoin.print_pending_pool()
#sbitcoin.print_chain()

#3. Alice transfers 10 coins to Bob and 5 coins to Marry
print "\n************#3*************************\n"
```

```
json_1 = {"hash": "",
"type": "PayCoins",
"coins_consumed": [
        {"hash":
"707e3e923d4e0b6c3c54ae6880659a9bb371cb7dd55689c3a9f153ff26dcf606", "num": 0}],
"coins_created": [
        {"num": 0, "value": 10, "recipient": "bob"},
        {"num": 1, "value": 5, "recipient": "marry"}],
"signatures" : [],
"Timestamp": ""}
trs1 = Transaction(json_1)
trs1.sign("alice","a")
#sbitcoin.add_transaction(trs1)
#sbitcoin.print_pending_pool()
#sbitcoin.print_chain()

#4. Bob mines the next block
print "\n************#4*************************\n"
sbitcoin.mine("bob")
#sbitcoin.print_pending_pool()
#sbitcoin.print_chain()

#5. Bob transfers 25 coins to Caleb and 5 coins to Alice
print "\n************#5*************************\n"
json_2 = {"hash": "",
"type": "PayCoins",
"coins_consumed": [
        {"hash":
"bbf6b700cf24734e2874663bcf52f47485099cb993ceb46e0b42594e20319e56", "num": 0},
        {"hash":
"f1531ef8c83c9d37d68f2505b2a4809d416cfe1b18f2e81645cbb19b5d01c27f", "num": 0}],
"coins_created": [
        {"num": 0, "value": 25, "recipient": "caleb"},
        {"num": 1, "value": 5, "recipient": "alice"}],
"signatures" : [],
"Timestamp": ""}
trs2 = Transaction(json_2)
trs2.sign("bob","b")
#sbitcoin.add_transaction(trs2)
#sbitcoin.print_pending_pool()
#sbitcoin.print_chain()

#6. Alice creates a transaction that Caleb transfers 15 coins to Alice,
#sign this transaction by Alice private key
print "\n************#6*************************\n"
json_3 = {"hash": "",
"type": "PayCoins",
"coins_consumed": [
        {"hash":
"5fcd537615afb5c717ee0e8f9243f903fc131798012468376c715cb3e8334046", "num": 0}],
"coins_created": [
        {"num": 0, "value": 15, "recipient": "alice"}],
"signatures" : [],
"Timestamp": ""}
trs3 = Transaction(json_3)
trs3.sign("alice","a")
#sbitcoin.add_transaction(trs3)
#sbitcoin.print_pending_pool()
#sbitcoin.print_chain()

#7. Marry mines the next block
print "\n************#7*************************\n"
#sbitcoin.mine("marry")

print "\n************#8*************************\n"
#8. Alice and Marry transfer 5 coins to Bob and 5 coins to Caleb
```

```
json_4 = {"hash": "",
"type": "PayCoins",
"coins_consumed": [
        {"hash":
"f50dfd2738d4b923ee3ed54514b8ccf0cfa86048cd5a9d6f2c3d7aadf7b2671d", "num": 0},
        {"hash":
"bbf6b700cf24734e2874663bcf52f47485099cb993ceb46e0b42594e20319e56", "num": 1}],
"coins_created": [
        {"num": 0, "value": 5, "recipient": "bob"},
        {"num": 0, "value": 5, "recipient": "caleb"}],
"signatures" : [],
"Timestamp": ""}
trs4 = Transaction(json_4)
trs4.sign("alice","a")
trs4.sign("marry","m")
#sbitcoin.add_transaction(trs4)
#sbitcoin.print_pending_pool()
#sbitcoin.print_chain()

print "\n***********#9*************************\n"
#9. Caleb transfers 15 coins to Bob and 5 coins to Marry
json_5 = {"hash": "",
"type": "PayCoins",
"coins_consumed": [
        {"hash":
"5fcd537615afb5c717ee0e8f9243f903fc131798012468376c715cb3e8334046", "num": 0}
        ],
"coins_created": [
        {"num": 0, "value": 15, "recipient": "bob"},
        {"num": 1, "value": 5, "recipient": "marry"}],
"signatures" : [],
"Timestamp": ""}
trs5 = Transaction(json_5)
trs5.sign("caleb","c")
#sbitcoin.add_transaction(trs5)
#sbitcoin.print_pending_pool()
#sbitcoin.print_chain()
```