Appendix II: blockchain_utils.py

05/12/19 05:04:07 C:\GitHub\basic_blockchain\blockchain_utils.py

```python
'''
A very, very naive blockchain.

Only used for demo purposes.

Not provably secure. Meant to show that it works in the context of this project.
'''



import hashlib
import os
from stat import S_IREAD, S_IRGRP, S_IROTH
import binascii
import time

HASH_FN = hashlib.sha512
HASH_LENGTH = 512


# a placeholder function that users can customize, if they want a different layout
def format_update(update_file):
    ''' Converts raw update file to a format for the blockchain. '''
    with open(update_file, "rb") as update_file:
        update_body = update_file.read()
    # do whatever formatting here
    return update_body # should be in byte format


# raised when a proof of work check fails
class ProofException(Exception):
    ''' Raised when proof of work fails to verify. '''
    pass


class Block():
    '''
    A single update unit to a blockchain. Block information is formatted in this order.

    self.hash   =   hash of current block (includes everything else)
    proof       =   proof of work
    prev        =   hash of previous block
    body        =   body of update for current block
    append_time =   timestamp

    Everything must be string format.
    '''
    def verify_proof(self, proof, prev, body):
        # return proof[:6] == '000000'
        return True

    def __init__(self, chain_name, proof, prev, body):
        # verify proof
        if self.verify_proof(proof, prev, body):
            # generate hash of block
            append_time = str(time.time())
            try:
                self.hash = HASH_FN(str.encode(proof) + str.encode(prev) + str.encode(body) + str.encode(append_time)).hexdigest()
            except:
                print("All arguments should be passed as string format.")

            # write to record file
            with open(chain_name + "/" + self.hash, "w+") as block_file:
                block_file.write(self.hash + '\n')
                block_file.write(proof + '\n')
                block_file.write(prev + '\n')
                block_file.write(body + '\n')
                block_file.write(append_time + '\n')

            # make file read-only
            os.chmod(chain_name + "/" + self.hash, S_IREAD)
        else:
            # do not create block if proof does not verify
            raise ProofException("Proof of work failed.")


class Chain():
    '''
    Made of many Blocks strung together.
    The first Block is always a bunch of random gibberish, i.e. a standard header.
    This is to provide randomness and security to the rest of the blockchain,
    and to prevent null pointers.

    chain_header stores the tail, aka the hash of the most recently appended block.

    self.name = name of the chain
    self.tail = hash of most recently appended block
        This is just meant to make appending easier for the sake of testing and demonstration.
        This tail file is not meant to be a secure display of the latest appended block. Unlike the record files it is not write protected.
    '''

    # reads from existing chain if available, or creates a new one
    def __init__(self, chain_name, seed_length):
        self.name = chain_name

        if os.path.isfile(chain_name): # existing chain already exists
            print("Loading blockchain '" + chain_name + "'.\n")
            with open(chain_name + '/' + chain_name, "r") as chain_header:
                self.tail = chain_header.readline()

        else: # initialize new blockchain
            print("Blockchain '" + chain_name + "'' does not exist. Initializing new chain.\n")
            os.mkdir(chain_name)

            # create new root block
            with open(chain_name + '/' + chain_name, 'w+') as chain_header:
                proof = os.urandom(seed_length).hex()
                prev = os.urandom(HASH_LENGTH).hex()
                body = os.urandom(seed_length).hex()

                root_block = Block(self.name, proof, prev, body)
                chain_header.write(root_block.hash)

            # initialize blockchain t ail
            with open(chain_name + '/' + chain_name, "r") as chain_header:
                self.tail = chain_header.readline()


    # adds a new block onto the chain
    def append_block(self, proof, prev, body):
        try: # attempt to create new block
            new_block = Block(self.name, proof, prev, body)

            # update header file and tail
            with open(self.name + '/' + self.name, 'w') as chain_header:
                chain_header.write(new_block.hash)
                self.tail = new_block.hash
            print("Successfully appended update " + new_block.hash + "\n")

        except ProofException:
            # do not create new block record or update tail if proof fails
            print("Proof of work failed. Block not appended.")




# test_chain = Chain('newchain_1', 720)
# test_chain.append_block('proof1', test_chain.tail, 'update1')
# test_chain.append_block('proof2', test_chain.tail, 'update2')
# test_chain.append_block('proof3', test_chain.tail, 'update3')
# test_chain.append_block('proof4', test_chain.tail, 'update4')
# test_chain.append_block('proof5', test_chain.tail, 'update5')
```