ПРАВИТЕЛЬСТВО РОССИЙСКОЙ ФЕДЕРАЦИИ
НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ
«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Факультет компьютерных наук
Департамент программной инженерии

Согласовано

Профессор департамента
программной инженерии факультета
компьютерных наук, канд. техн. наук

_____________ С.М. Авдошин

”____”_________ 2019 г

Утверждаю

Академический руководитель
образовательной программы
«Программная инженерия»
профессор, канд. техн. наук

_____________ В. В. Шилов

”____”_________ 2019 г

Криптографические алгоритмы и протоколы для распределенных реестров

Текст программы

ЛИСТ УТВЕРЖДЕНИЯ

RU.17701729.04.01 12 01-1

Студент группы БПИ 151 НИУ ВШЭ
_____________ Куприянов К. И.
”____”_________ 2019 г

2019

Подп. и дата

Инв. № дубл.

Взам. инв. №

Подп. и дата

Инв. № подл.

УТВЕРЖДЕНО
RU.17701729.04.01 12 01-1

Криптографические алгоритмы и протоколы для распределенных реестров

Текст программы

RU.17701729.04.01 12 01-1

Листов 36

2019

Инв. № подл.	Подп. и дата	Взам. инв. №	Инв. № дубл.	Подп. и дата

Содержание

1. Текст программы

Исходный код на языке Python3.6.5, вспомогательные скрипты автоматизации на языке Bash, и конфигурационные файлы на языке YAML1.2 приведены ниже.

1.1. Исходный код

./example_config.yaml

```
# Example and debug configuration
#
init_dir: /tmp/gsl
```

./setup.py

```
from setuptools import setup, find_packages

__author__ = 'Kirill Kupriyanov'
__author_email__ = 'kupriyanovkirill@gmail.com'


def version():
    with open('VERSION') as _fd_:
        return _fd_.read()


def requirements():
    with open('requirements.txt', 'r') as _fd_:
        return _fd_.read().split('\n')


def readme():
    with open('README.md') as _fd_:
        return _fd_.read()


ENTRY_POINTS = {
    'console_scripts': [
        'gsl=goodsteel_ledger:main'
    ]
}

setup(
    name='gsl',
    version=version(),
    description='Goodsteel Ledger -- a program for building own distributed ledger',
    long_description=readme(),
     classifiers =[
        # 'Development Status :: 1 - Planning',
        # 'Development Status :: 2 - Pre-Alpha',
        # 'Development Status :: 3 - Alpha',
        'Development Status :: 4 - Beta',
        # 'Development Status :: 5 - Production/Stable',
        # 'Development Status :: 6 - Mature',
        # ’Development Status :: 7 - Inactive ’,
        ’Topic :: Security :: Cryptography’,
        ’Programming Language :: Python’,
        ’Programming Language :: Python :: 3.6’,
        ’License :: OSI Approved :: GNU General Public License v3 (GPLv3)’,
        ’Operating System :: POSIX :: Linux ’,
    ],
    author=__author__,
    author_email=__author_email__,
    package_dir={
        ’’: ’src’,
    },
    packages=find_packages(’src’),
    include_package_data=True,
    install_requires=requirements(),
    zip_safe=False,
    entry_points=ENTRY_POINTS
)



./README.md
# gsl

[![Run Status](https://api.shippable.com/projects/5cbc3edfdaf54c0007d7bbd1/badge?branch=master)]()
[![License: GPL v3](https://img.shields.io/badge/License-GPLv3-blue.svg)](https://www.gnu.org/licenses/gpl-3.0)


gsl -- Goodsteel ledger. Krinkle Goodsteel will help you to build your own distributed ledger.

## Installation
Execute following commands linebyline
‘‘‘ bash
git clone https://github.com/Sinopsys/gsl.git
cd gsl
export PYTHONPATH=$PYTHONPATH":$(pwd)/src"
[[ $PATH != *".local/bin"* ]] && export PATH=$PATH":/home/$USER/.local/bin"
echo "mkdir /tmp/gsl" && mkdir /tmp/gsl
echo "sudo mkdir /etc/gsl" && sudo mkdir /etc/gsl
echo "sudo cp ./example_config.yaml /etc/gsl/config.yaml" && sudo cp ./example_config.yaml /etc/gsl/config.yaml

python3.6 -m pip install . --user

## OPTIONAL, if you do not want to set it manually every time
## Just echoes above exports to your $SHELLrc file
rc_file="/home/$USER/.$(echo $SHELL | rev | cut -d / -f 1 | rev)rc"
echo "export PATH=\$PATH\":/home/$USER/.local/bin\"" >> $rc_file
echo "export PYTHONPATH=\$PYTHONPATH\":$(pwd)/src/\"" >> $rc_file
‘‘‘

Then, to successfully launch app, it is needed to have a config with path
‘/etc/gsl/config.yaml‘. During installation, an example config has already been
put to that path.

Example ‘config.yaml‘:
‘‘‘ yaml
# Key init_dir is path where your blockchain will be stored.
#
init_dir: /tmp/gsl
‘‘‘
```

```
If you encounter error like _ModuleNotFoundError: No module named 'goodsteel\_ledger'_ then firstly, ‘cd‘ into gsl
    directory and then run

‘‘‘ bash
export PYTHONPATH=$PYTHONPATH":$(pwd)/src"
‘‘‘


## Usage

‘‘‘ bash
gsl --init --name NAME [--path PATH] # path can also be taken from config.
‘‘‘

## Questions
Write kupriyanovkirill@gmail.com, mephisto@openmail.cc, https://t.me/SsinopsysS, or create an issue
```

```
./src/updater.py
#!/usr/bin/env python3.6

import subprocess
from technologies import _kv_, get
TOINSTALL = _kv_.TOINSTALL
UPDATE_LINKS = _kv_.UPDATE_LINKS

for alg, src in UPDATE_LINKS.items():
    p = subprocess.Popen(['bash', 'pull_single.sh', f'{get("TOINSTALL", alg)}', f'{src}'], stdout=subprocess.PIPE)
    (result, error) = p.communicate()
    print(result.decode())
```

```
./src/utils/output.py
"""
    gsl -- Goodsteel ledger. A program for building an own distributed ledger

    Copyright (C) 2019 Kirill Kupriyanov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program. If not, see <https://www.gnu.org/licenses/>.
"""
"""
    Utils, function and classes for output
"""


class ASCIIColors:
    """
        Class container for save ASCII
    """
    ENDS = '\033[0m'
    LIGHT_CYAN = '\033[96m'
    YELLOW = '\033[33m'
    WHITE = '\033[97m'
    BACK_LIGHT_BLUE = '\033[104m'
    BACK_BLUE = '\033[44m'
    INVERTED = '\033[7m'


class NestedPrint:
    """
        Console nested outputter
    """

    def printf(self, output, space):
        """
            Wrapper function for output all passed param
        :param output:
        :param space: int
        :return:
        """
        if isinstance(output, dict):
            self.pdict(output, space)
        elif isinstance(output, list ):
            self.plist(output, space)
        else:
            print('{0}{1}- {2}{3}'.format(
                ''.rjust(space, ' '),
                ASCIIColors.INVERTED,
                output,
                ASCIIColors.ENDS
            ))

    def pdict(self, output, space):
        for key, val in output.items():
            print('{0}{1}{2}:{3}'.format(
                ''.rjust(space, ' '),
                ASCIIColors.LIGHT_CYAN,
                key,
                ASCIIColors.INVERTED
            ))
            self.printf(val, space=(space + 4))

    def plist(self, output, space):
        for item in output:
            if isinstance(item, dict):
                self.pdict(item, space=(space + 4))
            else:
                print(
                    '{0}{1}- {2}{3}'.format(
                        ''.rjust(space, ' '),
                        ASCIIColors.INVERTED,
                        item,
                        ASCIIColors.ENDS
                    )
                )
```

```
print_nested = NestedPrint().printf
```

```
./src/utils/log.py
"""
    Configs for loggers
"""

LOG_CONFIG = {
    'version': 1,
    'formatters': {
        'default': {
            'format': '[{asctime}] [{module} -> {process} -> {thread}] [{levelname}] >> {message}',
            'datefmt': '%Y-%m-%d %H:%M:%S',
            'style': '{',
        },
    },
    'handlers': {
        'default_console': {
            'level': 'DEBUG',
            'class': 'logging.StreamHandler',
            'formatter': 'default',
        },
    },
    'loggers': {
        'stdout': {
            'handlers': ['default_console'],
            'level': 'DEBUG',
            'propagate': False,
        },
        'manager': {
            'handlers': ['default_console'],
            'level': 'DEBUG',
            'propagate': False
        },
    },
}
```

```
./src/utils/misc.py
def get_version() -> str:
    """
    Get package version
    :return: str
    """
    with open('../VERSION', 'r') as __fd__:
        return __fd__.read_lines()
```

```
./src/config.yaml
# Example and debug configuration
#
init_dir: /home/coldmind/Projects/vkr/proga/result
```

```
./src/pull_single.sh
```

```
#!/usr/bin/env bash

if cd $1;
then
    cd ..
    name1=$(echo $1 | rev | cut -d/ -f1 | rev)
    name2=$(echo $2 | rev | cut -d/ -f1 | rev)
    if [[ $name1 == $name2 ]]
    then
        rm -rf $name1
        git clone $2
        cd $name2
        rm -rf .git*
    fi
fi


# EOF


./src/gsl_profiler.py
import os
import sys
import requests
import itertools
import json
import subprocess
from technologies import _kv_, get
from goodsteel_ledger import Jarquai

TOINSTALL = _kv_.TOINSTALL
OPTIONS = _kv_.OPTIONS


class Profiler(object):
    def __init__(self):
        pass

    def get_all_algs(self, alg):
        res = []
        for item in get('OPTIONS', alg):
            if item in TOINSTALL.keys():
                res.append(item)
        return res

    def create_ledger(self, options, name, path, t):
        jq = Jarquai(options, name, path, t, True)
        jq.build_ledger()

    def measure_all(self):
        port = 5000
        import_path = os.path.join(self.path, self.name)
        os.system('sed -ir "0,/def _portd/{s/_portd = .*/_portd = ' + str(port) + '/}" ' +
                  os.path.join(import_path, 'miner.py'))
        os.system('sed -ir "0,/def _portd/{s/_portd = .*/_portd = ' + str(port) + '/}" ' +
                  os.path.join(import_path, 'wallet.py'))
        # if import_path not in sys.path:
        #     sys.path.append(import_path)
        # import miner
        # import wallet
        # miner.full_run()
        # wallet._profile_timings()
        # reloaded = requests.get('http://localhost :' + str(port) + '/reload')
        # print('\n\n\n\n')
        # print(reloaded.content)
        # print('\n\n\n\n')
        # pass

    def measure_all_times(self, path):
        self.path = path
        self.name = 'gsl_profiling'
        hashes = self.get_all_algs('hashing')
        dss = self.get_all_algs('digital signature')
        num = int(os.environ['ALGS_NUM'])
        pairs = list(itertools.product(hashes, dss))[num]
        options = {'hashing': pairs[0], 'digital signature': pairs[1]}
        self.create_ledger(options, self.name, path, True)
        self.measure_all()
        # for hash_ in hashes:
        #     for ds_ in dss:
        #         options = {'hashing': hash_, 'digital signature': ds_}
        #         self.create_ledger(options, self.name, path, True)
        #         self.measure_all()
        #         pass

    def _pip_install_(self, package):
        subprocess.call([sys.executable, '-m', 'pip', 'install', package])

    def _install_(self, path):
        """
        Installs with `python setup.py install`
        """
        # CRY HAVOC
        if 'ecdsa' in path.lower():
            self._pip_install_('ecdsa')
            return
        elif 'x11' in path.lower():
            self._pip_install_('x11_hash')
            return
        elif 'x17' in path.lower():
            self._pip_install_('x17_hash')
            return
        os.chdir(path)
        subprocess.call([sys.executable, f'{path}/setup.py', 'install'])
```

```
./src/profile.sh
#!/usr/bin/env bash

for i in {1..24}
do
    export num=$i
    python goodsteel_ledger.py --profile-all True --name myledger --path ~/tmp/gsl
done

# EOF
```

```
./src/algoritms/hashing/__interfaces/lyra2re-hash-python/myhashing.py
```

```
import lyra2re2_hash
from binascii import hexlify

name = 'lyra2re2_hash'
bit = '512'

class myhashing:
    def __init__(self):
        self.hasher = lyra2re2_hash

    def update(self, s):
        self.string = s
        self.res = hexlify(self.hasher.getPoWHash(self.string))

    def hexdigest(self):
        return self.res.decode()
```

./src/algoritms/hashing/__interfaces/x11/myhashing.py

```
import x11_hash
from binascii import hexlify

name = 'x11'
bit = '512'

class myhashing:
    def __init__(self):
        self.hasher = x11_hash

    def update(self, s):
        self.string = s

    def hexdigest(self):
        return hexlify(self.hasher.getPoWHash(self.string)).decode()
```

./src/algoritms/hashing/__interfaces/blake2b/myhashing.py

```
from Crypto.Hash import BLAKE2b
from binascii import hexlify

name = 'blake2b'
bit = '256'

class myhashing:
    def __init__(self):
        self.hasher = BLAKE2b.new(digest_bits=256)

    def update(self, s):
        self.hasher.update(s)

    def hexdigest(self):
        return self.hasher.hexdigest()
```

./src/algoritms/hashing/__interfaces/pyscrypt/myhashing.py

```
from pyscrypt import hash
from binascii import hexlify
from uuid import uuid4

name = ’scrypt’
bit  = ’256’

class  myhashing:
    def __init__(self):
        self.hasher = hash

    def update(self, s):
        self.string  = s
        self.random_str = str(uuid4())

    def hexdigest(self):
        return  hexlify(hash(self.string,  self.random_str.encode(’utf-8’), 16,  16,  16,  16)).decode()
```

./src/algoritms/hashing/__interfaces/keccak256/myhashing.py

```
from keccak import keccak

name = ’keccak256’
bit  = ’256’

myhashing = keccak.Keccak256
```

./src/algoritms/hashing/__interfaces/myr-groestl_hash/myhashing.py

```
import groestl_hash
from binascii  import hexlify

name = ’myr-groestl_hash’
bit  = ’512’

class  myhashing:
    def __init__(self):
        self.hasher = groestl_hash

    def update(self, s):
        self.string  = s

    def hexdigest(self):
        return  hexlify(self.hasher.getPoWHash(self.string)).decode()
```

./src/algoritms/hashing/__interfaces/keccak512/myhashing.py

```
from keccak import keccak

name = ’keccak512’
bit  -  ’512’

myhashing = keccak.Keccak512
```

./src/algoritms/hashing/__interfaces/sha256/myhashing.py

```
import hashlib

name = ’sha256’
bit  = ’256’
```

```
myhashing = hashlib.sha256


./src/algoritms/hashing/__interfaces/ethash/myhashing.py
import ethash
from binascii import hexlify

name = 'ethash'
bit = '256'

class myhashing:
    def __init__(self):
        self.hasher = ethash.keccak256

    def update(self, s):
        self.string = s

    def hexdigest(self):
        return hexlify(self.hasher(self.string)).decode()



./src/algoritms/hashing/__interfaces/sha512/myhashing.py
import hashlib

name = 'sha512'
bit = '256'

myhashing = hashlib.sha512


./src/algoritms/hashing/__interfaces/x17/myhashing.py
import x17_hash
from binascii import hexlify

name = 'x17'
bit = '512'

class myhashing:
    def __init__(self):
        self.hasher = x17_hash

    def update(self, s):
        self.string = s

    def hexdigest(self):
        return hexlify(self.hasher.x17_gethash(self.string)).decode()



./src/algoritms/hashing/__interfaces/blake2s/myhashing.py
from Crypto.Hash import BLAKE2s
from binascii import hexlify

name = 'blake2s'
bit = '256'

class myhashing:
    def __init__(self):
        self .hasher = BLAKE2s.new(digest_bits=256)

    def update(self, s):
        self .hasher.update(s)

    def hexdigest( self ):
        return self .hasher.hexdigest()



./src/algoritms/digital_signature/__interfaces/pygost/mydss.py
import base64
from binascii import hexlify, unhexlify
from pygost.gost3410 import CURVE_PARAMS
from pygost.gost3410 import GOST3410Curve
from pygost.gost3410 import bytes2long
curve = GOST3410Curve(*CURVE_PARAMS['GostR3410_2012_TC26_ParamSetA'])
from os import urandom
# prv_raw = urandom(32)
from pygost.gost3410 import prv_unmarshal
# prv = prv_unmarshal(prv_raw)
from pygost.gost3410 import public_key
# pub = public_key(curve, prv)
from pygost.gost3410 import pub_marshal
from pygost. utils import hexenc
from pygost. utils import hexdec
# print('Public key is :', hexenc(pub_marshal(pub)))
from pygost import gost34112012256
# data_for_signing = b'some data'
# dgst = gost34112012256.new(data_for_signing).digest()
from pygost.gost3410 import sign
# signature = sign(curve, prv, dgst, mode=2012)
from pygost.gost3410 import verify
# verify(curve, pub, dgst, signature, mode=2012)

name = 'gost'
bit = '256'

class sk_:
    def __init__(self, prv, pub=0000):
        self .prv = prv
        self .pub = pub

    def to_string( self , pub=False):
        if pub:
            return hexenc(pub_marshal(self.pub))
        return self .prv

    def get_verifying_key(self):
        return self .pub

    def sign( self , bmsg):
        dgst = gost34112012256.new(bmsg).digest()
        signature = sign(curve, int( self .prv), dgst, mode=2012)
        return signature

class vk_:
    def __init__(self, pub):
        self .pub = pub
    def verify(self, signature, b_msg):
        self.dgst = gost34112012256.new(b_msg).digest()
        return verify(curve, self.pub, self.dgst, signature, mode=2012)


class SigningKey:
    def __init__(self):
        pass

    def generate(self):
        prv_raw = urandom(32)
        prv = prv_unmarshal(prv_raw)
        pub = public_key(curve, prv)
        return sk_(prv, pub)

    def from_string(self, prv):
        return sk_(prv)

class VerifyingKey:
    def __init__(self):
        pass

    def hex_to_pub(self, hex_pub):
        hex_pub = hexdec(hex_pub)[::-1]
        l = len(hex_pub) // 2
        first = bytes2long(hex_pub[l:])
        second = bytes2long(hex_pub[:l])
        return first, second

    def from_string(self, b_pub):
        pub = hexlify(b_pub)
        pub = self.hex_to_pub(pub)
        return vk_(pub)
```

./src/algoritms/digital_signature/__interfaces/ecdsa/mydss.py

```
import ecdsa

name = 'ecdsa'
bit = '256'

mydss = ecdsa
```

./src/pusher.sh

```
#!/usr/bin/env bash

curr_date=$(date +%d_%m_%Y)

git add .
git commit -m "Update algorithms: $curr_date."
git push


# EOF
```

./src/prolific_writer.py

```
"""
 Prolific Writer
```

```
"""

import os
import sys
import subprocess
from inspect import getsource
from shutil import copyfile
from technologies import _kv_, get
from target_dummy import wallet
from target_dummy import miner
TOINSTALL = _kv_.TOINSTALL
INTERFACES = _kv_.INTERFACES


class ProlificWriter(object):
    def __init__(self, full_path, opts, timed, profd=False):
        self.path = full_path
        self.opts = opts
        self.timed = timed
        self.profd = profd

    def write(self):
        # WRITE ALGORITHMS ITSELF
        #
        self._write_hashing_()
        self._write_digital_signature_()

        self._write_(wallet)
        self._write_(miner)

    def _write_(self, script_to_write):
        name = f'{script_to_write.__name__.split(".")[-1]}.py'
        src_code = getsource(script_to_write)
        with open(os.path.join(self.path, name), 'w') as __fd__:
            __fd__.write(src_code)
        if self.timed:
            os.system('sed -ir "0,/def _timed/{s/_timed = .*/_timed = True/}" ' + os.path.join(self.path, name))
        if self.profd:
            os.system('sed -ir "0,/def _profd/{s/_profd = .*/_profd = True/}" ' + os.path.join(self.path, name))

    def _get_src_path_(self):
        path = sys.path[::-1]
        for p in path:
            if 'gsl/src' in p:
                return p

    def _pip_install_(self, package):
        subprocess.call([sys.executable, '-m', 'pip', 'install', package, '--user'])

    def _install_(self, path):
        """
        Installs with `python setup.py install`
        """
        if 'ecdsa' in path.lower():
            self._pip_install_('ecdsa')
            return
        elif 'x11' in path.lower():
            self._pip_install_('x11_hash')
            return
        elif 'x17' in path.lower():
            self._pip_install_('x17_hash')
            return
        os.chdir(path)
        subprocess.call([sys.executable, f'{path}/setup.py', 'install', '--user'])

    def _write_hashing_(self):
        # INSTALLING PROCEDURE
        src_path = self._get_src_path_()
        path = os.path.join(src_path, get('TOINSTALL', self.opts['hashing']))
        self._install_(path)

        # WRITING PROCEDURE
        name = 'myhashing.py'
        type_ = self.opts['hashing']
        path = os.path.join(src_path, get('INTERFACES', type_), name)
        if not os.path.exists(self.path):
            os.makedirs(self.path)
        copyfile(path, os.path.join(self.path, name))

    def _write_digital_signature_(self):
        # INSTALLING PROCEDURE
        src_path = self._get_src_path_()
        path = os.path.join(src_path, get('TOINSTALL', self.opts['digital signature']))
        self._install_(path)

        # WRITING PROCEDURE
        name = 'mydss.py'
        type_ = self.opts['digital signature']
        path = os.path.join(src_path, get('INTERFACES', type_), name)
        copyfile(path, os.path.join(self.path, name))


# EOF
```

./src/goodsteel_ledger.py

```
"""
    gsl -- Goodsteel ledger. A program for building an own distributed ledger

    Copyright (C) 2019 Kirill Kupriyanov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program. If not, see <https://www.gnu.org/licenses/>.
"""
"""
    Init script for creating ledger.
"""
import sys
import os
import time
import argparse
import yaml
import logging.config

import config
from prolific_writer import ProlificWriter
from utils.log import LOG_CONFIG
from utils.misc import get_version
from utils.output import print_nested
from utils.output import ASCIIColors
from technologies import _kv_, get


TOINSTALL = _kv_.TOINSTALL
OPTIONS = _kv_.OPTIONS
LINKS = _kv_.LINKS


DEFAULT_CONFIG_PATH = '/etc/gsl/config.yaml'
LOGGING = logging.config.dictConfig(LOG_CONFIG)
__logger__ = logging.getLogger('stdout')


class ChooseError(Exception):
    """
    Error when choosing wrong option
    """
    pass


class Krinkle(object):
    """
    Helps to choose a ledger
    """
    def __init__(self, config_path: str):
        """
        :param config_path: str
            Path to a config file
        """
        self.config_path = config_path
        self.config = self.load_config()
        self.ledger_config = {}

    def load_config(self) -> dict:
        """
        Load and parse config file
        :return: dict
            Loaded and parsed config
        """
        return config.load(self.config_path)

    def print_description(self, name, path):
        print(f'''
            ==================================================
            ===============INITIALIZE LEDGER===============
            ==================================================

            Name: {name}
            Path: {path}
```

```
                ================================================
                ===============MAKE YOUR CHOISES===============
                ================================================
                ''' .format(name=name, path=path))
        print(f'{ASCIIColors.BACK_LIGHT_BLUE}THIS color indicates you will be provided with code or
            documentation for a particular algorithm BUT it will not be included in YOUR ledger
            code!{ASCIIColors.ENDS}')
        print(f'{ASCIIColors.BACK_BLUE}THIS color indicates that GSL will generate a working code for your
            ledger using a particular algorithm{ASCIIColors.ENDS}')

    def prompt(self, name, path=None) -> dict:
        """
        Prompts user to choose algorithms that will be used in a ledger
        :returns : dict
            Chosen options
            Example:
                structure:
                    - Blockchain
                openess:
                    - Public
                consensus:
                    - PoW
                hashing:
                    - SHA-256
                random:
                    - DRBG
        """
        if not os.path.exists(path) or os.path.isfile(path):
            __logger__.error('Path is invalid')
            sys.exit(1)
        self.print_description(name, path)
        print('\nChoose type of concrete algorithm from which your blockchain will consist of:\n')
        for k, v in OPTIONS.items():
            if not ('hash' in k or 'digital' in k):
                continue
            print(f'\nChoose type of {k} of the ledger')
            if isinstance(v, list):
                for num, opt in enumerate(v):
                    if 'hash' in k or 'digital' in k:
                        if opt in TOINSTALL:
                            prefix = ASCIIColors.BACK_BLUE
                        else:
                            prefix = ASCIIColors.BACK_LIGHT_BLUE
                    else:
                        prefix = ASCIIColors.ENDS
                    print(f'{prefix}{num+1}: {opt}{ASCIIColors.ENDS}', end='\n')
                try:
                    n = input(f'Enter num from 1 to {len(v)}, default [1]: ')
                    n = 0 if n == '' else int(n) - 1
                    if n < 0 or n >= len(v):
                        raise ChooseError
                    self.ledger_config[k] = n
                except Exception as e:
                    __logger__.exception(str(e))
                    return
            else:
                print(v)
        print('\n\nNow, choose related themes for which you will be provided with relevant information (links, web
            sites, etc.)\n')
        for k, v in OPTIONS.items():
            if 'hash' in k or 'digital' in k:
                continue
            print(f'\nOption: {k} of the ledger')
            if isinstance(v, list):
                for num, opt in enumerate(v):
                    if 'hash' in k or 'digital' in k:
                        if opt in TOINSTALL:
                            prefix = ASCIIColors.BACK_BLUE
                        else:
                            prefix = ASCIIColors.BACK_LIGHT_BLUE
                    else:
                        prefix = ASCIIColors.ENDS
                    print(f'{prefix}{num+1}: {opt}{ASCIIColors.ENDS}', end='\n')
                try:
                    n = input(f'Enter num from 1 to {len(v)}, default [1]: ')
                    n = 0 if n == '' else int(n) - 1
                    if n < 0 or n >= len(v):
                        raise ChooseError
                    self.ledger_config[k] = n
                except Exception as e:
                    __logger__.exception(str(e))
                    return
            else:
                print(v)
        print('\n\nThe following config is to be set:\n')
        chosen_to_print = {}
        for k, v in OPTIONS.items():
            chosen_to_print[k] = v[self.ledger_config[k]]
        print_nested(chosen_to_print, 1)
        if input('\nProceed with this config? [YES]/NO:').lower() in ['', 'yes', 'y', 'ye']:
            return chosen_to_print
        else:
            self.prompt()


class Jarquai(object):
    """
    This helps in building corresponding to a selected structure ledger
    """
    def __init__(self, options: dict, name, path, timed, profd=False):
        """
            :param options: dict
            Oprions, selected by a user
        """
        self.selected_options = options
        self.name = name
        self.path = path
        self.timed = timed
        self.profd = profd

    def build_ledger(self):
        """
        Alpha version, mathces links
        """
        if not os.path.exists(self.path):
            os.mkdir(os.path.join(self.path, self.name))
        # Write code of ledger to path
        self.pw = ProlificWriter(os.path.join(self.path, self.name),
                                 self.selected_options, self.timed, self.profd)
        self.pw.write()
        return  self .match_links()

    def match_links(self):
        """
        Alpha version, mathces links
        Prints  links  mathced by user's choose
        """
        res  = {}
        for  k,  v  in  self .selected_options.items():
            res [v]  = get('LINKS', v)
        print_nested(res,  1)


def arg_parser() -> object:
    """
        Argument parser
    :return:  object
        argparse namespace object
    """
    parser  = argparse.ArgumentParser(description='GSL execution script')
    parser.add_argument('--init', action='store_true')
    parser.add_argument('--name', action='store', dest='NAME', required=False)
    parser.add_argument('--path', action='store', dest='PATH', required=False)
    parser.add_argument('--time', action='store', dest='TIME', required=False)
    parser.add_argument('--profile-all' ,  action='store' ,  dest='PROFILE', required=False)

    return  parser.parse_args()


def main() -> None:
    """
    Main entry for program
    """
    args  = arg_parser()

    try:
        from  gsl_profiler  import  Profiler
        if  args.PROFILE.lower()  ==  'true':
            profiler   = Profiler ()
            profiler .measure_all_times(args.PATH)
            return
    except AttributeError as  e:
        pass
    except Exception as e:
        __logger__.error(e)
        # raise  (e)
        # return

    if  not args. init :
        __logger__.warning('NOT initializing a ledger since `--init`  argument was not provided.')
        sys. exit (1)
    __logger__.info('Start Goodsteel Ledger: a program for generating distributed ledgers')
    config_path = DEFAULT_CONFIG_PATH
    kr  = Krinkle(config_path)
    if  args.PATH is None:
        path = kr.config.get( 'init_dir' ,  '' )
    options  = kr.prompt(name=args.NAME, path=path)
    __logger__.info('Start getting your ledger\'s algorithms')
    time.sleep (0.5)
    try:
        if args.TIME.lower() == 'true':
            t = True
        else:
            t = False
    except:
        t = False
    jq = Jarquai(options, args.NAME, path, t)
    jq.build_ledger()


if __name__ == '__main__':
    main()



./src/config.py
"""
    gsl -- Goodsteel ledger. A program for building an own distributed ledger

    Copyright (C) 2019 Kirill Kupriyanov

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program. If not, see <https://www.gnu.org/licenses/>.
"""
"""
    Module for working with config
"""
import os
import yaml
import logging.config

from utils.log import LOG_CONFIG

LOGGING = logging.config.dictConfig(LOG_CONFIG)
__logger__ = logging.getLogger('stdout')


# Define Exceptions
class LoadConfigError(Exception):
    """
        Common config exception
    """
    pass


class ConfigYamlError(LoadConfigError):
    """
        Yaml parse config exception
    """
    pass
```

```
class CongifValidateError(LoadConfigError):
    """
        Config validation exception
    """
    pass


def load(path: str) -> dict:
    """
    Load and parse config file
    : param path : str
        Path to a config file
    : return : dict
        Loaded and parsed config
    """
    __logger__.info(f'Loading config from {path}')
    config = _read_(path)
    if not config:
        raise ConfigYamlError('Load configuration form yaml Fail')
    __logger__.info('Configuration loaded')
    return config


def _read_(path: str) -> dict:
    """
    Reads config from path
    : param path : str
        Path to a config file
    : return : dict
        Read yaml config
    """
    config = {}
    if not os.path.exists(path):
        raise OSError(f'No such file or directory {path}, please check if file \
                exists')

    with open(path, 'r') as __fd__:
        try:
            config = yaml.load(__fd__, Loader=yaml.BaseLoader)
        except yaml.YAMLError as err:
            __logger__.exception(
                f'Error {err.__class__.__name__} occurred when parse yaml\
                parse config file, {err}'
            )
        except Exception as err:
            __logger__.exception(
                f'Unknown error {err.__class__.__name__} occurred when parse\
                yaml parse config file, {err}'
            )

    return config


./src/target_dummy/julia.txt
Private key: c59be229e951e2cdad8207048ce4d2b5034613ea96e246c2a152309510a290bc
Wallet address / Public key:
    zPWDn3kWQmnnD9l+q7o0Q3PhiEmxqwJRSa8YfrSXhvBD+NQd+wGLBgqRd49sA+jrjT4mdYHpOAfGJOf1eMNxDA==
./src/target_dummy/kirill.txt
Private key: f43e9aa03fedd64c11000e9c8d76a220beb2fb22c87689f9c2d10c7fe2cc1d22
Wallet address / Public key:
    ytJVVZGRZjQBZJ2RWRVjg7oCaesEKmqNWTlpc8ncUIalVFHRywYhahA36nSDHJ18ZdCRybYgW+kfdt8rHwfyxw==

./src/target_dummy/john.txt
Private key: 1048ab60a7f402afca91311e24b7bc72880cec9a7e1ba608c7657d0b1e140399
Wallet address / Public key:
    YmLjToO9UrGfHe4Knv8wl+umrTvPuE7qzPNR8zjhEDvnfJZymnPblSNzR7EK140kaPIbeHKh7IYftL/+dgnKCQ==

./src/target_dummy/wallet.py
"""This is going to be your wallet. Here you can do several things:
- Generate a new address (public and private key). You are going
to use this address (public key) to send or receive any transactions. You can
have as many addresses as you wish, but keep in mind that if you
lose its credential data, you will not be able to retrieve it.

- Send coins to another address
- Retrieve the entire blockchain and check your balance

If this is your first time using this script don't forget to generate
a new address and edit miner config file with it (only if you are
going to mine).

Timestamp in hashed message. When you send your transaction it will be received
by several nodes. If any node mine a block, your transaction will get added to the
blockchain but other nodes still will have it pending. If any node see that your
transaction with same timestamp was added, they should remove it from the
node_pending_transactions list to avoid it get processed more than 1 time.
"""

import os
import sys
import time
import requests
import time
import base64

try:
    import mydss
    dss = mydss
    if hasattr(dss, 'name') and hasattr(dss, 'bit'):
        alg_name = dss.name
        alg_bit = dss.bit
        try:
            from mydss import mydss
            dss = mydss
        except:
            dss = mydss
except:
    import ecdsa
    dss = ecdsa
    alg_name = 'ecdsa'
    alg_bit = '256'

header_written = False
_timed = False
_profd = False
_port = 5000
```

```
def _write_time(alg, func, bit, etime):
    global header_written
    time_file = '/home/coldmind/tmp/gsl/time_profile_1.csv'
    if not header_written and not os.path.getsize(time_file) > 0:
        with open(time_file, 'a') as __fd__:
            __fd__.write('alg;func;bit;time\n')
            header_written = True
    with open(time_file, 'a') as __fd__:
        __fd__.write(f'{alg};{func};{bit};{etime}\n')


def _profile_timings():
    if _profd:
        keys = {
                    'p1_pub': '',
                    'p1_prv': '',
                    'p2_pub': '',
                    'p2_prv': ''
                    }
        p1_prv, p1_pub = generate_keys(ret=True)
        p2_prv, p2_pub = generate_keys(ret=True)
        print(f'sending from {p1_pub} \n to \n{p2_pub}\nusing\np1_prv')
        # Send for p1 to p2 100 money
        #
        _perform_transaction(p1_pub, p1_prv, p2_pub, 100)
        check_transactions()


def _perform_transaction(from_, prv_key, addr_to, amount):
    try:
        len_prv = len(prv_key)
    except:
        len_prv = len(str(prv_key))

    if dss.name == 'gost' or len_prv == 64:
        signature, message = _sign_msg(prv_key)
        url = f'http://localhost:{_port}/mycoin'
        payload = {'from': from_,
                   'to': addr_to,
                   'amount': amount,
                   'signature': signature.decode(),
                   'message': message}
        headers = {'Content-Type': 'application/json'}

        res = requests.post(url, json=payload, headers=headers)
        print(res.text)
    else:
        print('Wrong address or key length! Verify and try again.')


def check_transactions():
    # Gets whole blockchain
    res = requests.get(f'http://localhost:{_port}/blocks')
    print(res.text)


def generate_keys(ret=False):
    if _timed:
        t1 = time.time()

    try:
        singningkey = dss.SigningKey.generate(curve=dss.SECP256k1)
    except:
        singningkey = dss.SigningKey().generate()

    try:
        private_key = singningkey.to_string().hex()
    except:
        private_key = singningkey.to_string()

    vk = singningkey.get_verifying_key()
    try:
        public_key = vk.to_string().hex()
    except:
        public_key = singningkey.to_string(pub=True)


    if _timed:
        t2 = time.time()
        _write_time(alg_name, 'Key pair generation', alg_bit, t2-t1)

    try:
        public_key = base64.b64encode(bytes.fromhex(public_key.decode()))
    except:
        public_key = base64.b64encode(bytes.fromhex(public_key))

    if ret:
        return private_key, public_key.decode()

    filename = input("Write the name of your new address: ") + ".txt"

    with open(filename, "w") as f:
        f.write("Private key: {0}\nWallet address / Public key: {1}\n".format(private_key, public_key.decode()))
    print("Your new address and private key are now in the file {0}".format(filename))


def _sign_msg(private_key):
    message = str(round(time.time()))
    bmessage = message.encode()

    if _timed:
        t1 = time.time()

    try:
        sk = dss.SigningKey.from_string(bytes.fromhex(private_key), curve=dss.SECP256k1)
    except:
        sk = dss.SigningKey().from_string(str(private_key))
    signed = sk.sign(bmessage)

    if _timed:
        t2 = time.time()
        _write_time(alg_name, 'Signing message', alg_bit, t2-t1)

    signature = base64.b64encode(signed)
    return signature, message


def provide_options():
    response = None
    while response not in ['1', '2', '3', '4']:
        response = input(
            """
            Which action would you like to take?
            1. Generate new wallet
            2. Send coins to another wallet
            3. View transactions
            4. Quit wallet.py\n
            """)
    if response == '1':
        print("""============================================\n
IMPORTANT: save this credentials or you won't be able to recover your wallet\n
============================================\n""")
        generate_keys()
    elif response == '2':
        addr_from = input('From: introduce your wallet address (public key)\n')
        private_key = input('Introduce your private key\n')
        addr_to = input('To: introduce destination wallet address\n')
        amount = input('Amount: number stating how much do you want to send\n')
        try:
            if float(amount) <= 0:
                print('You better send positive amounts! :)')
                sys.exit()
        except:
            print(f'Bad number "{amount}". Could not parse.')
            sys.exit()
        print('===============================================\n\n')
        print('Is everything correct?\n')
        print('From: {0}\nPrivate Key: {1}\nTo: {2}\nAmount: {3}\n'.format(addr_from, private_key, addr_to,
            amount))
        response = input('y/n\n')
        if response.lower() == 'y':
            _perform_transaction(addr_from, private_key, addr_to, amount)
    elif response == '3':
        check_transactions()
    else:
        print('Good bye!')
        sys.exit(0)


if __name__ == '__main__':
    # print("""        ===============================================\n
    #      Build using source code from and so more help at: https://github.com/cosme12/SimpleCoin\n
    #       ===========================================""")
    #
    if _profd:
        _profile_timings()
        sys.exit()
    provide_options()
    torepeat = input('Repeat? Would you like one more action? (Y/[N])')
    while torepeat.lower() in ['y', 'yes', 'da']:
        provide_options()
        torepeat = input('Repeat? Would you like one more action? (Y/[N])')
    print('Exiting..')

# EOF
```

./src/target_dummy/miner.py

```
import os
import time
import hashlib
import json
import requests
import base64
from datetime import datetime
from flask import Flask, request
from multiprocessing import Process, Pipe
from werkzeug.serving import run_simple

try:
    import mydss
    dss = mydss
    if hasattr(dss, 'name') and hasattr(dss, 'bit'):
        alg_name = dss.name
        alg_bit = dss.bit
        try:
            from mydss import mydss
            dss = mydss
        except:
            dss = mydss
except:
    import ecdsa
    dss = ecdsa
    alg_name = 'ecdsa'
    alg_bit = '256'

try:
    import myhashing
    hashing = myhashing
    if hasattr(hashing, 'name') and hasattr(hashing, 'bit'):
        hash_name = hashing.name
        hash_bit = hashing.bit
        try:
            from myhashing import myhashing
            hashing = myhashing
        except:
            hashing = myhashing
except:
    hashing = hashlib.sha256
    hash_name = 'sha256'
    hash_bit = '256'


a, b = Pipe()
header_written = False
to_reload = False
_timed = False
_profd = False
_port = 5000

MINER_ADDRESS = 'k4Odf238gn-random-dkfi3-address-k394rbgfGKe392f'
MINER_NODE_URL = f"http://localhost:{_port}"
PEER_NODES = []


def _write_time(alg, func, bit, etime):
    global header_written
    # time_file = '/home/coldmind/tmp/gsl/time_profile_1.csv'
    time_file = '/tmp/time_profile_1.csv'
    if not os.path.exists(time_file) or (not header_written and not os.path.getsize(time_file) > 0):
        with open(time_file, 'a') as __fd__:
            __fd__.write('alg;func;bit;time\n')
            header_written = True
    with open(time_file, 'a') as __fd__:
        __fd__.write(f'{alg};{func};{bit};{etime}\n')


class Block:
    def __init__(self, index, timestamp, data, previous_hash):
        self.index = index
        self.timestamp = timestamp
        self.data = data
        self.previous_hash = previous_hash
        self.hash = self.hash_block()

    def hash_block(self):
        if _timed:
            t1 = time.time()
        hasher = hashing()
        hasher.update((str(self.index) + str(self.timestamp) + str(self.data) +
            str(self.previous_hash)).encode('utf-8'))
        hexxx = hasher.hexdigest()
        if _timed:
            t2 = time.time()
            _write_time(hash_name, 'Hash value computing', hash_bit, t2-t1)
        return hexxx


def create_genesis_block():
    return Block(0, time.time(), {
        'proof-of-work': 9,
        'transactions': None},
        '0')


BLOCKCHAIN = [create_genesis_block()]

""" Stores the transactions that this node has in a list.
If the node you sent the transaction adds a block
it will get accepted, but there is a chance it gets
discarded and your transaction goes back as if it was never
processed"""
NODE_PENDING_TRANSACTIONS = []


def proof_of_work(last_proof, blockchain):
    if _timed:
        t1 = time.time()
    # For finding new proof of work
    incrementer = last_proof + 1
    # Keep incrementing the incrementer until it's equal to a number divisible by 9
    # and the proof of work of the previous block in the chain
    start_time = time.time()
    while not (incrementer % 7919 == 0 and incrementer % last_proof == 0):
        incrementer += 1
        if int((time.time()-start_time) % 60) == 0:
            # If any other node got the proof, stop searching
            new_blockchain = consensus(blockchain)
            if new_blockchain:
                # (False: another node got proof first , new blockchain)
                return False, new_blockchain
    # Once that number is found, we can return it as a proof of our work
    if _timed:
        t2 = time.time()
        _write_time(hash_name, 'Proof of Work', hash_bit, t2-t1)
    return incrementer, blockchain


def mine(a, blockchain, node_pending_transactions):
    BLOCKCHAIN = blockchain
    NODE_PENDING_TRANSACTIONS = node_pending_transactions
    while True:
        """Mining is the only way that new coins can be created.
        In order to prevent too many coins to be created, the process
        is slowed down by a proof of work algorithm.
        """

        if _timed:
            t1 = time.time()
        # Get the last proof of work
        last_block = BLOCKCHAIN[len(BLOCKCHAIN) - 1]
        last_proof = last_block.data['proof-of-work']
        # Find the proof of work for the current block being mined
        # Note: The program will hang here until a new proof of work is found
        proof = proof_of_work(last_proof, BLOCKCHAIN)
        # If we didn't guess the proof, start mining again
        if not proof [0]:
            # Update blockchain and save it to file
            BLOCKCHAIN = proof[1]
            a.send(BLOCKCHAIN)
            continue
        else :
            # Once we find a valid proof of work, we know we can mine a block so
            # ...we reward the miner by adding a transaction
            # First we load all pending transactions sent to the node server
            NODE_PENDING_TRANSACTIONS = requests.get(MINER_NODE_URL + "/mycoin?update=" +
                  MINER_ADDRESS).content
            NODE_PENDING_TRANSACTIONS = json.loads(NODE_PENDING_TRANSACTIONS)
            # Then we add the mining reward
            NODE_PENDING_TRANSACTIONS.append({
                "from": "network",
                "to": MINER_ADDRESS,
                "amount": 1})
            # Now we can gather the data needed to create the new block
            new_block_data = {
                "proof-of-work": proof [0],
                "transactions": list (NODE_PENDING_TRANSACTIONS)
            }
            new_block_index = last_block.index + 1
            new_block_timestamp = time.time()
            last_block_hash = last_block.hash
            # Empty transaction list
            NODE_PENDING_TRANSACTIONS = []
            # Now create the new block
            mined_block = Block(new_block_index, new_block_timestamp, new_block_data, last_block_hash)
            BLOCKCHAIN.append(mined_block)
            # Let the client know this node mined a block
            try :
                print(json.dumps({
                  "index": new_block_index,
                  "timestamp": str(new_block_timestamp),
                  "data": new_block_data,
                  "hash": last_block_hash.decode()
                }) + "\n")
            except:
                print(json.dumps({
                  "index": new_block_index,
                  "timestamp": str(new_block_timestamp),
                  "data": new_block_data,
                  "hash": last_block_hash
                }) + "\n")
            a.send(BLOCKCHAIN)
            requests.get(MINER_NODE_URL + "/blocks?update=" + MINER_ADDRESS)
            if _timed:
                t2 = time.time()
                _write_time(hash_name, 'Mining one block', hash_bit, t2-t1)


def find_new_chains():
    # Get the blockchains of every other node
    other_chains = []
    for node_url in PEER_NODES:
        # Get their chains using a GET request
        block = requests.get(node_url + "/blocks").content
        # Convert the JSON object to a Python dictionary
        block = json.loads(block)
        # Verify other node block is correct
        validated = validate_blockchain(block)
        if validated:
            # Add it to our list
            other_chains.append(block)
    return other_chains


def consensus(blockchain):
    # Get the blocks from other nodes
    other_chains = find_new_chains()
    # If our chain isn't longest, then we store the longest chain
    BLOCKCHAIN = blockchain
    longest_chain = BLOCKCHAIN
    for chain in other_chains:
        if len(longest_chain) < len(chain):
            longest_chain = chain
    # If the longest chain wasn't ours, then we set our chain to the longest
    if longest_chain == BLOCKCHAIN:
        # Keep searching for proof
        return False
    else:
        # Give up searching proof, update chain and start over again
        BLOCKCHAIN = longest_chain
        return BLOCKCHAIN


def validate_blockchain(block):
    """Validate the submitted chain. If hashes are not correct, return false
    block(str): json
    """
    return True
```

```
def validate_signature(public_key, signature, message):
    """Verifies if the signature is correct. This is used to prove
    it's you (and not someone else) trying to do a transaction with your
    address. Called when a user tries to submit a new transaction.
    """
    if _timed:
        t1 = time.time()
    try:
        if hasattr(dss, 'name'):
            if dss.name == 'gost':
                public_key = base64.b64decode(public_key)
        else:
            public_key = (base64.b64decode(public_key)).hex()
    except:
        pass
    signature = base64.b64decode(signature)
    try:
        print(public_key)
        print(type(public_key))
        vk = dss.VerifyingKey().from_string(public_key)
    except:
        curve = dss.SECP256k1
        vk = dss.VerifyingKey.from_string(bytes.fromhex(public_key), curve=curve)

    # Try changing into an if/else statement as except is too broad.
    try:
        res = vk.verify(signature, message.encode())
        if _timed:
            t2 = time.time()
            _write_time(alg_name, 'Verifying signature', alg_bit, t2-t1)
        return res
    except:
        return False


def welcome_msg():
    # print("""        ==========================================\n
    #      SIMPLE COIN v1.0.0 - BLOCKCHAIN SYSTEM\n
    #     ==========================================\n\n
    #      You can find more help at: https://github.com/cosme12/SimpleCoin\n
    #      Make sure you are using the latest version or you may end in
    #      a parallel chain.\n\n\n""")
    pass


def get_app():
    app = Flask(__name__)
    now = datetime.now()

    @app.route('/')
    def index():
        return f'hello, the app started at %s' % now

    @app.route('/reload')
    def reload():
        global to_reload
        to_reload = True
        return 'reloaded'
    @app.route(’/blocks’, methods=[’GET’])
    def get_blocks():
        # Load current blockchain. Only you should update your blockchain
        if request.args.get("update") == MINER_ADDRESS:
            global BLOCKCHAIN
            BLOCKCHAIN = b.recv()
        chain_to_send = BLOCKCHAIN
        # Converts our blocks into dictionaries so we can send them as json objects later
        chain_to_send_json = []
        for block in chain_to_send:
            try:
                block = {
                    ’index’: str(block.index),
                    ’timestamp’: str(block.timestamp),
                    ’data’: str(block.data),
                    ’hash’: block.hash.decode()
                }
            except:
                block = {
                    ’index’: str(block.index),
                    ’timestamp’: str(block.timestamp),
                    ’data’: str(block.data),
                    ’hash’: block.hash
                }
            chain_to_send_json.append(block)

        # Send our chain to whomever requested it
        chain_to_send = json.dumps(chain_to_send_json)
        return chain_to_send

    @app.route(’/mycoin’, methods=[’GET’, ’POST’])
    def transaction():
        """Each transaction sent to this node gets validated and submitted.
        Then it waits to be added to the blockchain. Transactions only move
        coins, they don’t create it.
        """
        if request.method == ’POST’:
            new_mycoin = request.get_json()
            if validate_signature(new_mycoin[’from’], new_mycoin[’signature’], new_mycoin[’message’]):
                NODE_PENDING_TRANSACTIONS.append(new_mycoin)
                print("New transaction")
                print("FROM: {0}".format(new_mycoin[’from’]))
                print("TO: {0}".format(new_mycoin[’to’]))
                print("AMOUNT: {0}\n".format(new_mycoin[’amount’]))
                return "Transaction submission successful\n"
            else:
                return "Transaction submission failed. Wrong signature\n"
        elif request.method == ’GET’ and request.args.get("update") == MINER_ADDRESS:
            pending = json.dumps(NODE_PENDING_TRANSACTIONS)
            NODE_PENDING_TRANSACTIONS[:] = []
            return pending
    return app


class AppReloader(object):
    def __init__(self, create_app):
        self.create_app = create_app
        self.app = create_app()
    def get_application(self):
        global to_reload
        if to_reload:
            self.app = self.create_app()
            to_reload = False

        return self.app

    def __call__(self, environ, start_response):
        app = self.get_application()
        return app(environ, start_response)


def profd_run():
    welcome_msg()
    # Start mining
    p1 = Process(target=mine, args=(a, BLOCKCHAIN, NODE_PENDING_TRANSACTIONS))
    p1.start()
    # Start server to receive transactions

    kwargs = {
            ’use_reloader’: False,
            ’use_debugger’: True,
            ’use_evalex’: True
            }
    app = AppReloader(get_app)
    p2 = Process(target=run_simple, args=(’localhost’, 5000, app), kwargs=kwargs)
    p2.start()


def full_run():
    if _profd:
        profd_run()
    else:
        welcome_msg()
        node = get_app()
        # Start mining
        p1 = Process(target=mine, args=(a, BLOCKCHAIN, NODE_PENDING_TRANSACTIONS))
        p1.start()
        # Start server to receive transactions
        p2 = Process(target=node.run(), args=b)
        p2.start()


if __name__ == ’__main__’:
    full_run()


./src/technologies.py
class _kv_(object):
    OPTIONS = {
            ’structure’:  [’Blockchain’, ’DAG’, ’Hashgraph’, ’Holochain’, ’Tempo’],
            ’openess’:  [’Public’, ’Private’],
            ’consensus’:  [’PoW’, ’PoS’, ’DPoS’, ’PoA’, ’PoWeight’, ’BFT’],
            ’hashing’:  [’SHA-256’, ’SHA-512’, ’Scrypt’, ’KECCAK-256’,
                        ’KECCAK-512’, ’Ethash’, ’X11’, ’X17’, ’myr-groestl’,
                        ’Lyra2rev2’, ’blake2s’, ’blake2b’],
            ’random’: [’DRBG’, ’CPRNG’],
            ’digital signature’:  [’ECDSA’, ’DSA’, ’GOST R 34.10-2012’],
    }
```

```
LINKS = {
    ’Blockchain’: ’See ‘src/target_dummy/*‘ for a good classic blockchain implementation’,
    ’DAG’: ’https://github.com/thieman/py-dag’,
    ’Hashgraph’: ’https://github.com/Lapin0t/py-swirld’,
    ’Holochain’: ’https://github.com/holochain/holochain-rust’,
    ’Tempo’: ’https://github.com/radixdlt/radnet’,
    ’Public’ : ’Depends on your implementation:
        https://masterthecrypto.com/public-vs-private-blockchain-whats-the-difference/’,
    ’Private’ : ’Depends on your implementation:
        https://masterthecrypto.com/public-vs-private-blockchain-whats-the-difference/’,
    ’PoW’: ’https://github.com/csunny/py-bitcoin/blob/master/consensus/proof_of_work.py’,
    ’PoS’: ’https://github.com/csunny/blockchain’,
    ’DPoS’: ’https://github.com/DEADP0OL/DPoS-Slackbot’,
    ’PoA’: ’https://github.com/poanetwork/wiki/wiki/POA-Network-Whitepaper’,
    ’PoWeight’: ’Read https:// filecoin . io/ filecoin .pdf’,
    ’BFT’: ’https://github.com/practicalbft/BFTList-client/tree/master/client’,
    ’SHA-256’: ’https://github.com/thomdixon/pysha2/blob/master/sha2/sha256.py’,
    ’SHA-512’: ’https://github.com/thomdixon/pysha2/blob/master/sha2/sha512.py’,
    ’Scrypt’: ’https://github.com/ricmoo/pyscrypt’,
    ’KECCAK-256’: ’https://pycryptodome.readthedocs.io/en/latest/src/hash/hash.html’,
    ’KECCAK-512’: ’https://pycryptodome.readthedocs.io/en/latest/src/hash/hash.html’,
    ’Ethash’: ’https://github.com/ethereum/ethash’,
    ’X11’: ’https://pypi.org/project/x11_hash/’,
    ’X17’: ’https://pypi.org/project/x17_hash/’,
    ’Lyra2rev2’: ’https://github.com/straks/lyra2re-hash-python’,
    ’myr-groestl’ : ’https://github.com/vergecurrency/myr-groestl_hash’,
    ’blake2s’ : ’https://pycryptodome.readthedocs.io/en/latest/src/hash/hash.html’,
    ’blake2b’: ’https://pycryptodome.readthedocs.io/en/latest/src/hash/hash.html’,
    ’DRBG’: ’https://github.com/blubber/python-drbg/blob/master/drbg.py’,
    ’CPRNG’: ’https://riptutorial.com/python/example/3857/create-cryptographically-secure-random-numbers’,
    ’ECDSA’: ’https://github.com/warner/python-ecdsa’,
    ’DSA’: ’https://github.com/rrottmann/pydsa’,
    # ’elgamal128’: ’https://github.com/RyanRiddle/elgamal’,
    # ’elgamal256’: ’https://github.com/RyanRiddle/elgamal’,
    ’GOST R 34.10-2012’: ’https://pypi.org/project/pygost/’,
}

UPDATE_LINKS = {
    ’Ethash’: ’https://github.com/ethereum/ethash’,
    ’Lyra2rev2’: ’https://github.com/straks/lyra2re-hash-python’,
    ’myr-groestl’ : ’https://github.com/vergecurrency/myr-groestl_hash’,
    ’blake2b’: ’https://github.com/Legrandin/pycryptodome’,
    ’blake2s’ : ’https://github.com/Legrandin/pycryptodome’,
    ’Scrypt’: ’https://github.com/ricmoo/pyscrypt’,
    ’SHA-256’: ’https://github.com/thomdixon/pysha2’,
    ’SHA-512’: ’https://github.com/thomdixon/pysha2’,
    ’X11’: ’https://github.com/mazaclub/x11_hash’,
    ’ECDSA’: ’https://github.com/warner/python-ecdsa’,
}

TOINSTALL = {
    ’SHA-256’: ’algoritms/hashing/sha256’,
    ’SHA-512’: ’algoritms/hashing/sha512’,
    ’Scrypt’: ’algoritms/hashing/pyscrypt’,
    ’Ethash’: ’algoritms/hashing/ethash’,
    ’KECCAK-256’: ’algoritms/hashing/keccak’,
    ’KECCAK-512’: ’algoritms/hashing/keccak’,
    ’Lyra2rev2’: ’algoritms/hashing/lyra2re-hash-python’,
    ’myr-groestl’ : ’algoritms/hashing/myr-groestl_hash’,
        ’blake2s’ : ’algoritms/hashing/pycryptodome’,
        ’blake2b’ : ’algoritms/hashing/pycryptodome’,
        ’X11’: ’algoritms/hashing/x11_hash’,
        ’X17’: ’.x17_hash’,
        ’ECDSA’: ’algoritms/digital_signature/ecdsa’,
        # ’elgamal128’: ’algoritms/digital_signature/elgamal’,
        # ’elgamal256’: ’algoritms/digital_signature/elgamal’,
        ’GOST R 34.10-2012’: ’algoritms/digital_signature/pygost’,
    }

    INTERFACES = {
        ’SHA-256’: ’algoritms/hashing/__interfaces/sha256’,
        ’SHA-512’: ’algoritms/hashing/__interfaces/sha512’,
        ’Scrypt’: ’algoritms/hashing/__interfaces/pyscrypt’,
        ’Ethash’: ’algoritms/hashing/__interfaces/ethash’,
        ’KECCAK-256’: ’algoritms/hashing/__interfaces/keccak256’,
        ’KECCAK-512’: ’algoritms/hashing/__interfaces/keccak512’,
        ’Lyra2rev2’: ’algoritms/hashing/__interfaces/lyra2re-hash-python’,
        ’myr-groestl’ : ’algoritms/hashing/__interfaces/myr-groestl_hash’,
        ’blake2s’ : ’algoritms/hashing/__interfaces/blake2s’,
        ’blake2b’ : ’algoritms/hashing/__interfaces/blake2b’,
        ’ECDSA’: ’algoritms/digital_signature/__interfaces/ecdsa’,
        # ’elgamal128’: ’algoritms/digital_signature/__interfaces/elgamal128’,
        # ’elgamal256’: ’algoritms/digital_signature/__interfaces/elgamal256’,
        ’GOST R 34.10-2012’: ’algoritms/digital_signature/__interfaces/pygost’,
        ’X11’: ’algoritms/hashing/__interfaces/x11’,
        ’X17’: ’algoritms/hashing/__interfaces/x17’,
    }


def get(name, val, default=None):
    return getattr(_kv_, name).get(val, default)
```

```
./get_sources.sh
#!/usr/bin/env bash

1


# EOF
```

```
./VERSION
0.0.1
```

```
./requirements.txt
 certifi ==2019.3.9
chardet==3.0.4
Click==7.0
Flask==1.0.2
idna==2.8
itsdangerous==1.1.0
Jinja2==2.10.1
MarkupSafe==1.1.1
PyYAML==5.1
requests==2.21.0
 urllib3==1.24.3
```

```
Werkzeug==0.15.2
ecdsa==0.13.2
```

2. Приложение 1. Список используемой литературы

Список литературы

1. *Nakamoto S.* Bitcoin: A Peer-to-Peer Electronic Cash SyNakamoto, S. (2008). Bitcoin: A Peer-to-Peer Electronic Cash System. Consulted, 1–9. // Journal for General Philosophy of Science. — 2008. — № 1. — С. 1—9. — ISSN 09254560. — DOI: 10.1007/s10838-008-9062-0. — arXiv: 43543534534v343453.
2. *Vitalik Buterin*. On Public and Private Blockchains. — 2015. — URL: https://blog.ethereum.org/2015/08/07/on-public-and-private-blockchains/ (дата обр. 22.04.2019).
3. *Документации Е. С. П.* ГОСТ 19.101-77 Виды программ и программных документов. — ИПК Издательство стандартов, 2001.
4. *Документации Е. С. П.* ГОСТ 19.102-77 Стадии разработки. — ИПК Издательство стандартов, 2001.
5. *Документации Е. С. П.* ГОСТ 19.103-77 Обозначения программ и программных документов. — ИПК Издательство стандартов, 2001.
6. *Документации Е. С. П.* ГОСТ 19.104-78 Основные надписи. — ИПК Издательство стандартов, 2001.
7. *Документации Е. С. П.* ГОСТ 19.106-78 Требования к программным документам. — ИПК Издательство стандартов, 2001.
8. *Документации Е. С. П.* ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению. — ИПК Издательство стандартов, 2001.
9. *Документации Е. С. П.* ГОСТ 19.404-79 Пояснительная записка. Требования к содержанию и оформлению. — ИПК Издательство стандартов, 2001.
10. *Документации Е. С. П.* ГОСТ 19.603-78 Общие правила внесения изменений. — ИПК Издательство стандартов, 2001.