SECURED RECORDER BOX (SEREBO) VERSION 1.0

Maurice HT Ling*
Colossus Technologies LLP, Singapore
HOHY PTE LTD, Singapore

ABSTRACT

Data authenticity is crucial in many industries. A major aspect of data authenticity is to ensure that a created file is not fraudulently or purposefully edited; for example, changing the data file without affecting the date time stamp. Blockchain technology ensures data authenticity as recorded data is not mutable. This manuscript documents the implementation and the codes of SEREBO and licensed under GNU General Public License version 3. SEREBO codebase is hosted and available for forking at https://github.com/mauriceling/serebo.

Keywords: Data management, Research records, Blockchain, Immutability

Date Submitted: February 20, 2019. **Date Accepted:** March 29, 2019.

PROBLEM SCENARIO AND INTRODUCTION

One of the hallmarks of biological and biomedical research today is data; more specifically, big data (Dolinski and Troyanskaya, 2015; Suwinski et al., 2019). As a bioinformatics researcher, I often generate data files as I go about my work. Take for example, an Excel file, `sampleTimeSeries.xlsx`, that I had generated on 15th May 2015. Three years later, on 10th June 2018, how can I demonstrate that `sampleTimeSeries.xlsx` had not been changed / edited since 15th May 2015? If I had the intention to change the file on 10th December 2016, I can safely set my computer clock back to 15th May 2015, change the file, and the date will

* Corresponding Author: mauriceling@acm.org, mauriceling@colossus-tech.com

still be 15th May 2015. A recent study (Bik et al., 2018) analyzing 960 published papers found 59 (6.1%) papers contained inappropriately duplicated images, resulting in 5 retractions.

What if there is a way for me to log my file into a system, on 15th May 2015, and this record (the log statement) is not editable? To ensure that this record is not editable, it can be put on a blockchain. One of the most useful features of a blockchain is resistance against modification of data (Zheng et al., 2017). This resulted in the implementation of system called SEREBO – SEcured REcorder BOx (Ling, 2018a) – comprising of SEREBO Black Box and SEREBO Notary. This manuscript documents the implementation of SEREBO. SEREBO is available for forking at https://github.com/mauriceling/serebo under GNU General Public License version 3 for non-commercial or academic use only.

SEREBO Black Box is inspired by the black boxes (cockpit voice recorder and flight data recorder) in airliners (Pierce, 2010). The intended purpose is to track and audit research records under the following premise – Given a set of data files, is there a system to log and verify that these files had not been changed or edited since its supposed creation? SEREBO Black Box addresses this issue by three approaches. Firstly, the data files can be used to generate a file hash as an edit in the file will result in a different hash. As such, a file that generates the same hash across two different points in time can be safely assumed unedited during this time span. Secondly, the file hash is securely recorded with amendment protection. SEREBO Black Box records the hash and registers the hash into a blockchain. The main concept of blockchain is that the hash of previous (parent) block is concatenated with the data (file hash in this case) of the current block to generate a hash for the current block. Therefore, any amendments in earlier blocks can be easily detected as the blockchain grows as only amendments to the latest block cannot be detected. This property makes the data in blockchain immutable once it is locked within a chain (Zheng et al., 2017). Lastly, SEREBO Notary is implemented as a web-based notarized by one or more independent notary to notarize SEREBO Black Boxes, which adds another layer of modification restriction to downstream SEREBO Black Boxes. The architecture of SEREBO Notary is based on a previous work, NotaLogger (Ling, 2013).

ARCHITECTURE AND IMPLEMENTATION

SEREBO consists of three components (Figure 1):

- SEREBO Command Line, which is the command-line user interface (CLI) to access both SEREBO Black Box and SEREBO Notary. This is implemented in `serebo.py` (main command-line processor) and `serebo_notary_api.py` (interface to access SEREBO Notary) files.

- SEREBO Black Box, which is the blockchain-based data storage facility. This is implemented in `serebo_api.py` (interface between SEREBO Black Box to the external world) and `sereboDB.py` (talks to SEREBO Black Box database) files.
- SEREBO Notary, which acts as a public and independent notary to sign off (notarize) SEREBO Black Boxes. This is implemented in `services.py` (interface for SEREBO Notary to other SEREBO Black Boxes) and `serebo_notabase.py` (talks to SEREBO Notary database) files.

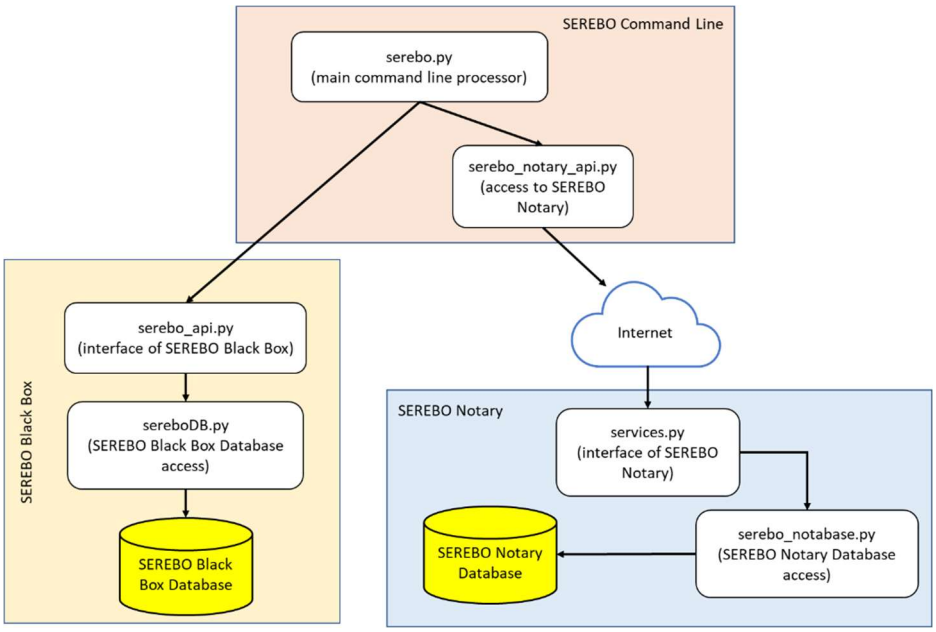

Figure 1. Overview of SEREBO.

The entry point to both SEREBO Black Box and SEREBO Notary is SEREBO Command Line, implemented using Python 3 and Python-Fire module (https://github.com/google/python-fire), which aims to simplify the implementation of command-line interface in Python 3. The means of access is using `serebo.py` file. At the command line level, the general syntax is `python serebo.py [operation] [option(s)]`. The operations / commands in SEREBO Command Line can be classified into three categories, consisting of a total of 32 operations, as follows:

1. SEREBO Black Box Operations
   1.1. `backup`: Backup SEREBO Black Box

1.2. `dump`: Dump out data (text backup) from SEREBO Black Box
1.3. `fhash`: Generate and print out hash of a file
1.4. `init`: Initialize SEREBO Black Box
1.5. `intext`: Insert a text string into SEREBO Black Box
1.6. `localcode`: Generate a random string, and log this generation into SEREBO Black Box
1.7. `localdts`: Get date time string
1.8. l`ogfile`: Log a file into SEREBO Black Box
1.9. `ntpsign`: Self-sign (self-notarization) SEREBO Black Box using NTP (Network Time Protocol) Server
1.10. `searchmsg`: Search SEREBO Black Box data log for a message
1.11. `searchdesc`: Search SEREBO Black Box data log for a description
1.12. `searchfile`: Search SEREBO Black Box data log for a file log
1.13. `selfsign`: Self-sign (self-notarization) SEREBO Black Box
1.14. `shash`: Generate hash for a data string using SEREBO Black Box
1.15. `sysdata`: Print out data and test hashes of current platform
1.16. `sysrecord`: Record data and test hashes of current platform into SEREBO Black Box
1.17. `viewntpnote`: View all self-notarization(s) by NTP time server for this SEREBO Black Box
1.18. `viewselfnote`: View self-notarization(s) for current SEREBO Black Box

2. SEREBO Notary Operations
1.1. `changealias`: Change alias for a specific SEREBO Notary registration
1.2. `notarizebb`: Notarize SEREBO Black Box with SEREBO Notary
1.3. `register`: Register SEREBO Black Box with SEREBO Notary
1.4. `viewsnnote`: View notarization(s) by SEREBO Notary(ies) for current SEREBO Black Box
1.5. `viewreg`: View current SEREBO Notary registrations for current SEREBO Black Box

3. Audit Operations
1.1. `audit_blockchainflow`: Trace the decendency of blockchain records (also known as blocks) within SEREBO Black Box
1.2. `audit_blockchainhash`: Check for accuracy in blockchain hash generation within SEREBO Black Box
1.3. `audit_count`: Check for equal numbers of records in data log and blockchain in SEREBO Black Box
1.4. `audit_data_blockchain`: Check for accuracy in data log and blockchain mapping in SEREBO Black Box

1.5. `audit_datahash`: Check for accuracy of hash generations in data log within SEREBO Black Box
1.6. `audit_notarizebb`: Check for SEREBO Black Box notarization records in SEREBO Notary
1.7. `audit_register`: Check for registration between SEREBO Black Box and SEREBO Notary
1.8. `checkhash`: Compare record hash from SEREBO Black Box with that in a file
1.9. `dumphash`: Dump out record hash from SEREBO Black Box into a file

The black box is implemented as a SQLite database consisting of 7 tables (defined in sereboDB.py):

1. `metadata` table stores information about the SEREBO Black Box. At creation using `init` command, date time stamp of creation and a randomly generated 512-character string to represent the identity of the SEREBO Black Box will be recorded.
2. `notary` table stores registration record(s) between SEREBO Black Box to one or more SEREBO Notary(ies), registered using register command.
3. `systemdata` table stores data and test hashes of current platform using sysrecord command, which is used to provide a data baseline and for future checking for potential differences in processing outcomes due to different platforms.
4. `datalog` table stores the actual data to be logged and its corresponding hash.
5. `blockchain` table is the backbone of SEREBO Black Box, where each record / tuple represents a block in the blockchain.
6. `eventlog` table stores audit trail of the data logging event when a piece of data is stored in datalog table.
7. `eventlog_datamap` table provides the second audit trail of data hash and block chain hashes when a piece of data is stored in datalog table.

Input into SEREBO Black Box, via SEREBO Command Line, can either be a string or a file. When the input is a string, the data is the actual input string and the description can be used to provide additional information about the string. When a file is given, a series of 12 hashes is generated from the file (Ling, 2018b) to reduce the possibility of hash collision (Boneh and Boyen, 2006; Broder and Mitzenmacher, 2001; Rasjid et al., 2017). The series of hashes is then concatenated to form the data. The absolute and relative file path are added to the description. Hence, both file and string input are reduced to a data component and a description component (Figure 2), which is then used to generate a series of 12 hashes, called DataHash, after the inclusion of a date time stamp for the input event. The data, description, date time stamp, and DataHash are recorded in the datalog table.

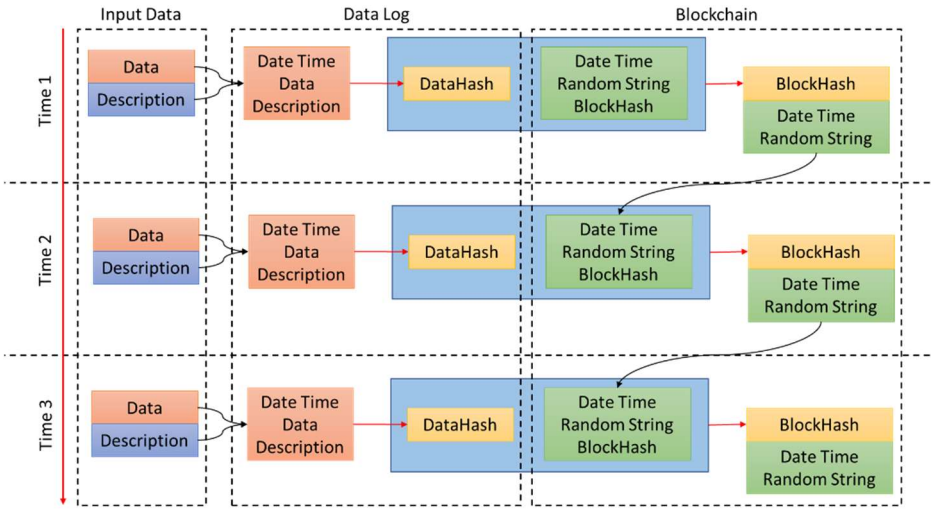

Figure 2. Overview of Operations in SEREBO Black Box [Adapted from (Ling, 2018a)).

The latest block in blockchain table is identified and used as the parental block. Three attributes / values from the parental block are extracted – date time stamp (known as parental Date Time stamp), random string (a 32-character random string, known as parental Random String), and block hash (known as parental BlockHash). These 3 parental attributes will be combined with DataHash to generate a BlockHash (known as current BlockHash). A new 32-character Random String (known as current Random String) will be generated and a new block is generated (and will be used as parental block for the next succeeding block), which consists of the following attributes:

1. Date time stamp of the data entry event (identical to date time stamp in the datalog table)
2. Current Random String
3. Current BlockHash
4. Parental block ID
5. Parental Date Time stamp
6. Parental Random String
7. Parental BlockHash
8. Data (which is DataHash)

Finally, the description and date time stamp are logged in eventlog table. This is followed by logging current BlockHash, parental BlockHash, and DataHash are logged in eventlog_datamap table.

SEREBO Notary is a web application built on Web2Py framework (Pierro, 2008) and exposes a set of XMLRPC web services to provide an independent agent / platform as a notary service. The architecture of SEREBO Notary is based on a

previous work, NotaLogger (Ling, 2013). A SEREBO Black Box can be registered with one or more SEREBO Notaries. An instance SEREBO Notary has been set up for public use and is accessible at https://mauricelab.pythonanywhere.com/serebo_notary/services/call/xmlrpc. After registration, a SEREBO Notary can be called to notarize the SEREBO Black Box. During which, the SEREBO Black Box will generate a local date time stamp (known as Black Box date time stamp) and 32-character random string (known as Black Box code) to be transmitted to the SEREBO Notary, together with identity of the SEREBO Black Box. Upon receiving this set of information, the SEREBO Notary will generate a date time stamp (known as Notary date time stamp) and a 32-character random string (known as Notary code). In addition, the SEREBO Notary will generate a set of hashes (known as common code) from the Black Box code and Notary code. All information will be logged in SEREBO Notary before transmitting Notary date time stamp, Notary code, and common code back to the requesting SEREBO Black Box for logging into datalog table, which will in turn trigger the logging into blockchain.

CODE FILES FOR SEREBO COMMAND LINE[1]

FILE NAME: SEREBO.PY

```
'''!
Secured Recorder Box (SEREBO) Command Line Interface (CLI)

Date created: 17th May 2018

License: GNU General Public License version 3 for academic or
not-for-profit use only


SEREBO 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
ofMERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See
theGNU General Public License for more details.
```

[1] These code files had been formatted for display and printing purposes; hence, different from the original codes in http://github.com/mauriceling/serebo on a line-by-line basis. However, they are logically, syntactically, and semantically identical to the original codes.

```
You should have received a copy of the GNU General Public
Licensealong    with    this    program.    If    not,    see
<http://www.gnu.org/licenses/>.
'''
import random
import os
import sqlite3

import fire

import serebo_blackbox as bb
import serebo_notary_api as notary


def initialize(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to initialize SEREBO blackbox.

    Usage:

        python serebo.py init
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py init \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    try:
        sqlstmt = '''insert into metadata (key, value) values
           ('serebo_blackbox_path', '%s');''' %\
           (str(db.path))
        db.cur.execute(sqlstmt)
        db.conn.commit()
    except sqlite3.IntegrityError: pass
    print('')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path)}

def insertText(message, description='NA',
               bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to insert a text string into SEREBO blackbox.
```

```
    Usage:

        python serebo.py intext
        --message=<text message to be inserted>
        --description=<explanatory description for this
        insertion>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py intext \
        --message="This is a text message for insertion" \
        --description="Texting 1" \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param message String: Text string to be inserted.
    @param description String: Explanation string for this
    entry event. Default = NA.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    rdata = bb.insertText(db, message, description)
    print('')
    print('Insert Text Status ...')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Date Time Stamp': str(rdata['DateTimeStamp']),
            'Message': str(rdata['Data']),
            'Description': str(rdata['UserDescription']),
            'Data Hash': str(rdata['DataHash'])}

def logFile(filepath, description='NA',
            bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to log a file into SEREBO blackbox.

    Usage:

        python serebo.py logfile
        --filepath=<path of file to log>
        --description=<explanatory description for this
        insertion>
--bbpath=<path to SEREBO black box>

    For example:

        python serebo.py logfile \
```

```
        --filepath=doxygen_serebo  \
        --description="Doxygen file for SEREBO" \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param fileapth String: Path of file to log in SEREBO black
    box.
    @param description String: Explanation string for this
    entry event. Default = NA.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    rdata = bb.logFile(db, filepath, description)
    print('')
    print('File Logging Status ...')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Date Time Stamp': str(rdata['DateTimeStamp']),
            'File Hash': str(rdata['Data']),
            'Description': str(rdata['UserDescription']),
            'Data Hash': str(rdata['DataHash'])}

def systemData():
    '''!
    Function to print out data and test hashes of current \
    platform - This does not insert a record into SEREBO Black
    Box.

    Usage:

        python serebo.py sysdata
    '''
    data = bb.systemData()
    print('')
    print('System Data ...')
    return {'architecture': str(data['architecture']),
            'machine': str(data['machine']),
            'node': str(data['node']),
            'platform': str(data['platform']),
            'processor': str(data['processor']),
            'python_build': str(data['python_build']),
            'python_compiler': str(data['python_compiler']),
            'python_implementation': \
                str(data['python_implementation']),
            'python_branch': str(data['python_branch']),
            'python_revision': str(data['python_revision']),
            'python_version': str(data['python_version']),
            'release': str(data['release']),
```

```
            'system': str(data['system']),
            'version': str(data['version']),
            'hashdata': str(data['hashdata']),
            'hash_md5': str(data['hash_md5']),
            'hash_sha1': str(data['hash_sha1']),
            'hash_sha224': str(data['hash_sha224']),
            'hash_sha3_224': str(data['hash_sha3_224']),
            'hash_sha256': str(data['hash_sha256']),
            'hash_sha3_256': str(data['hash_sha3_256']),
            'hash_sha384': str(data['hash_sha384']),
            'hash_sha3_384': str(data['hash_sha3_384']),
            'hash_sha512': str(data['hash_sha512']),
            'hash_sha3_512': str(data['hash_sha3_512']),
            'hash_blake2b': str(data['hash_blake2b']),
            'hash_blake2s': str(data['hash_blake2s'])}

def systemRecord(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to record data and test hashes of current
    platform.

    Usage:

        python serebo.py sysrecord \
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py sysrecord \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    data = bb.systemData()
    dtstamp = bb.dateTime(db)
    sqlstmt = '''insert into systemdata (dtstamp, key, value)
    values ('%s', '%s', '%s');'''
    print('')
    print('System Data ...')
    for k in data:
        if k != 'hashdata':
            db.cur.execute(sqlstmt % (str(dtstamp), str(k),
                                      str(data[k])))
    db.conn.commit()
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
```

```
            'architecture': str(data['architecture']),
            'machine': str(data['machine']),
            'node': str(data['node']),
            'platform': str(data['platform']),
            'processor': str(data['processor']),
            'python_build': str(data['python_build']),
            'python_compiler': str(data['python_compiler']),
            'python_implementation': \
                str(data['python_implementation']),
            'python_branch': str(data['python_branch']),
            'python_revision': str(data['python_revision']),
            'python_version': str(data['python_version']),
            'release': str(data['release']),
            'system': str(data['system']),
            'version': str(data['version']),
            'hashdata': str(data['hashdata']),
            'hash_md5': str(data['hash_md5']),
            'hash_sha1': str(data['hash_sha1']),
            'hash_sha224': str(data['hash_sha224']),
            'hash_sha3_224': str(data['hash_sha3_224']),
            'hash_sha256': str(data['hash_sha256']),
            'hash_sha3_256': str(data['hash_sha3_256']),
            'hash_sha384': str(data['hash_sha384']),
            'hash_sha3_384': str(data['hash_sha3_384']),
            'hash_sha512': str(data['hash_sha512']),
            'hash_sha3_512': str(data['hash_sha3_512']),
            'hash_blake2b': str(data['hash_blake2b']),
            'hash_blake2s': str(data['hash_blake2s'])}

def fileHash(filepath):
    '''!
    Function to generate and print out hash of a file.

    Usage:

        python serebo.py fhash \
        --filepath=<path of file to hash>

    For example:

        python serebo.py fhash \
        --filepath=doxygen_serebo

    @param fileapth String: Path of file to log in SEREBO black
    box.
    '''
    fHash = bb.fileHash(filepath)
    print('')
```

```
    return {'File Path': str(filepath),
            'File Hash': str(fHash)}

def localCode(length, description=None,
              bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to generate a random string, and log this
    generation into SEREBO Black Box.

    Usage:

        python serebo.py localcode \
        --length=<length of random string>
        --description=<explanatory description for this
        insertion>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py localcode \
        --length=10 \
        --description="Notarizing certificate ABC123" \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param length Integer: Length of random string to generate
    @param description String: Explanation string for this
    entry event. Default = None.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    rstring = bb.randomString(db, length)
    description = ['Local random string generation'] + \
        [description]
    description = ' | '.join(description)
    rdata = bb.insertFText(db, rstring, description)
    print('')
    print('Generate Random String (Local) ...')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Date Time Stamp': str(rdata['DateTimeStamp']),
            'Random String': str(rstring)}

def localDTS(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to get date time string. This event is not logged.

    Usage:
```

```
        python serebo.py localdts \
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py localdts \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    dts = bb.dateTime(db)
    print('')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Date Time Stamp': str(dts)}

def stringHash(dstring,
               bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to generate hash for a data string. This event
    is not logged.

    Usage:

        python serebo.py shash
        --dstring=<string to hash>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py shash \
        --dstring="SEREBO is hosted at
        https://github.com/mauriceling/serebo" \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param dstring String: String to generate hash.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    x = bb.stringHash(db, dstring)
    print('')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Data String': str(dstring),
```

```
            'Data Hash': str(x)}

def registerBlackbox(owner, email, alias,
                     notaryURL='https://mauricelab.
                     pythonanywhere.com/serebo_notary/
                     services/call/xmlrpc',
                    bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to register SEREBO Black Box with SEREBO Notary.

    Usage:

        python serebo.py register
        --alias=<alias for this SEREBO Notary>
        --notaryURL="https://mauricelab.pythonanywhere.
        com/serebo_notary/services/call/xmlrpc"
        --owner=<owner's name>
        --email=<owner's email>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py register \
        --alias="NotaryPythonAnywhere" \
        --notaryURL="https://mauricelab.pythonanywhere.
        com/serebo_notary/services/call/xmlrpc" \
        --owner="Maurice HT Ling" \
        --email="mauriceling@acm.org" \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param owner String: Owner's or administrator's name.
    @param email String: Owner's or administrator's email.
    @param alias String: Alias for this SEREBO Notary.
    @param notaryURL String: URL for SEREBO Notary web
    service.  Default="https://mauricelab.pythonanywhere.com/
    serebo_notary/services/call/xmlrpc"
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    owner = str(owner)
    email = str(email)
    sqlstmt = "select value from metadata where
        key='blackboxID'"
    blackboxID = [row
                  for row in db.cur.execute(sqlstmt)][0][0]
    data = bb.systemData()
    architecture = data['architecture']
```

```
    machine = data['machine']
    node = data['node']
    platform = data['platform']
    processor = data['processor']
    try:
        (notaryURL, notaryAuthorization, dtstamp) = \
            notary.registerBlackbox(blackboxID, owner, email,
                                    architecture, machine,
                                    node, platform,
                                    processor, notaryURL)
        sqlstmt = '''insert into notary (dtstamp, alias,
            owner, email, notaryDTS, notaryAuthorization,
            notaryURL) values (?,?,?,?,?,?,?)'''
        sqldata = (db.dtStamp(), alias, owner, email,
                   dtstamp, notaryAuthorization, notaryURL)
        db.cur.execute(sqlstmt, sqldata)
        db.conn.commit()
        rstring = 'Register SEREBO Black Box with SEREBO
            Notary'
        description = ['Notary URL: %s' % str(notaryURL),
                       'Notary Authorization: %s' % \
                            str(notaryAuthorization),
                        'Notary Date Time Stamp %s' %\
                            str(dtstamp)]
        description = ' | '.join(description)
        rdata = bb.insertFText(db, rstring, description)
        print('')
        print('Registering SEREBO Black Box with SEREBO
            Notary...')
        return {'SEREBO Black Box': db,
                'Black Box Path': str(db.path),
                'Black Box ID': str(blackboxID),
                'Notary URL': str(notaryURL),
                'Notary Authorization': \
                    str(notaryAuthorization),
                'Notary Date Time Stamp': str(dtstamp)}
    except:
        print('Registration failed - likely to be SEREBO
            Notary error or XMLRPC error.')
        return {'SEREBO Black Box': db,
                'Black Box Path': str(db.path),
                'Black Box ID': str(blackboxID),
                'Notary URL': str(notaryURL)}

def selfSign(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to self-sign (self notarization) SEREBO Black
    Box.
```

```
    Usage:

        python serebo.py selfsign \
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py selfsign \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    rstring = bb.randomString(db, 32)
    rdata = bb.insertFText(db, rstring, 'Self notarization')
    print('')
    print('Self-Signing / Self-Notarization ...')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Date Time Stamp': str(rdata['DateTimeStamp']),
            'Random String': str(rstring)}

def notarizeBlackbox(alias,
                bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to notarize SEREBO Black Box with SEREBO Notary.

    Usage:

        python serebo.py notarizebb \
        --alias=<alias for SEREBO Notary> \
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py notarizebb \
        --alias="NotaryPythonAnywhere" \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param alias String: Alias for this SEREBO Notary.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    sqlstmt = "select value from metadata where
        key='blackboxID'"
```

```
    blackboxID = [row
                  for row in db.cur.execute(sqlstmt)][0][0]
    try:
        sqlstmt = "select notaryAuthorization, notaryURL from
notary where alias='%s'" % str(alias)
        sqlresult = [row
                     for row in db.cur.execute(sqlstmt)][0]
        notaryAuthorization = sqlresult[0]
        notaryURL = sqlresult[1]
    except IndexError:
        print('Notary authorization or Notary URL not found
           for the given alias')
        return {'SEREBO Black Box': db,
                'Black Box Path': str(db.path),
                'Notary Alias': str(alias)}
    dtstampBB = bb.dateTime(db)
    codeBB = bb.randomString(db, 32)
    try:
        (notaryURL, dtstampNS, codeNS, codeCommon) = \
            notary.notarizeBB(blackboxID,
                              notaryAuthorization,
                              dtstampBB, codeBB, notaryURL)
        description = ['Notarization with SEREBO Notary',
                       'Black Box Code: %s' % codeBB,
                       'Black Box Date Time: %s' % dtstampBB,
                       'Notary Code: %s' % codeNS,
                       'Notary Date Time: %s' % dtstampNS,
                       'Notary URL: %s' % notaryURL]
        description = ' | '.join(description)
        rdata = bb.insertFText(db, codeCommon, description)
        print('')
        print('Notarizing SEREBO Black Box with SEREBO
            Notary...')
        return {'SEREBO Black Box': db,
                'Black Box Path': str(db.path),
                'Notary Alias': str(alias),
                'Notary URL': str(notaryURL),
                'Notary Authorization': \
                    str(notaryAuthorization),
                'Notary Date Time Stamp': str(dtstampNS),
                'Date Time Stamp': str(dtstampBB),
                'Black Box Code': str(codeBB),
                'Notary Code': str(codeNS),
                'Cross-Signing Code': str(codeCommon)}
    except:
        print('Failed in attempt to notarize SEREBO Black Box
            with SEREBO Notary')
        return {'SEREBO Black Box': db,
```

```
                'Black Box Path': str(db.path),
                'Notary Alias': str(alias),
                'Notary URL': str(notaryURL),
                'Notary Authorization': \
                    str(notaryAuthorization)}

def viewRegistration(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to view all SEREBO Notary registration for this
    SEREBO Black Box - This does not insert a record into
    SEREBO Black Box.

    Usage:

        python serebo.py viewreg
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py viewreg \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    print('')
    print('Black Box Path: %s' % str(bbpath))
    sqlstmt = '''select dtstamp, alias, owner, email,
    notaryDTS, notaryAuthorization, notaryURL from notary'''
    print('')
    print('Notary Registration(s) ...')
    for row in db.cur.execute(sqlstmt):
        print('')
        print('Date Time Stamp: %s' % str(row[0]))
        print('Notary Alias: %s' % str(row[1]))
        print('Owner: %s' % str(row[2]))
        print('Email: %s' % str(row[3]))
        print('Notary Date Time Stamp: %s' % str(row[4]))
        print('Notary Authorization: %s' % str(row[5]))
        print('Notary URL: %s' % str(row[6]))

def changeAlias(alias, newalias,
                bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to change alias for a specific SEREBO Notary
    registration.
```

```
    Usage:

        python serebo.py changealias
        --alias=<current alias to be changed>
        --newalias=<new alias to change into>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py changealias \
        --alias="NotaryPythonAnywhere" \
        --newalias="testAlias" \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param alias String: Current alias for the SEREBO Notary
    to change.
    @param newalias String: New alias for the SEREBO Notary.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    alias = str(alias)
    newalias = str(newalias)
    sqlstmt = '''update notary set alias=? where alias=?'''
    db.cur.execute(sqlstmt, (newalias, alias))
    db.conn.commit()
    message = 'Change notary alias from %s to %s' % \
        (alias, newalias)
    rdata = bb.insertFText(db, message, 'NA')
    print('')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Alias': alias,
            'New Alias': newalias}

def searchMessage(term, mode='like',
                  bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to search SEREBO Black Box for a message - This
    does not insert a record into SEREBO Black Box.

    Usage:

        python serebo.py searchmsg
        --mode=<search mode>
        --term=<search term>
        --bbpath=<path to SEREBO black box>
```

```
    For example:

        python serebo.py searchmsg
        --mode='like'
        --term="Change notary alias%"
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param term String: Case sensitive search term.
    @param mode String: Mode of search. Allowable modes are
    'like' and 'exact'. If mode is 'like', wildcards such as
    '_' (matches any single character) and '%' (matches any
    number of characters). Default = 'like'.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    mode = str(mode)
    term = str(term)
    result = bb.searchDatalog(db, term, 'data', mode)
    print('')
    print('Search Result (Search by Message) ...')
    print('')
    for row in result:
        print('Date Time Stamp: %s' % str(row[1]))
        print('Message: %s' % str(row[3]))
        print('Description: %s' % str(row[4]))
        print('')

def searchDescription(term, mode='like',
                      bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to search SEREBO Black Box for a description -
    This does not insert a record into SEREBO Black Box.

    Usage:

        python serebo.py searchdesc
        --mode=<search mode>
        --term=<search term>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py searchdesc
        --mode='like'
        --term="%NA%"
        --bbpath='serebo_blackbox\\blackbox.sdb'
```

```
    @param term String: Case sensitive search term.
    @param mode String: Mode of search. Allowable modes are
    'like' and 'exact'. If mode is 'like', wildcards such as
    '_' (matches any single character) and '%' (matches any
    number of characters). Default = 'like'.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    mode = str(mode)
    term = str(term)
    result = bb.searchDatalog(db, term, 'description', mode)
    print('')
    print('Search Result (Search by Description) ...')
    print('')
    for row in result:
        print('Date Time Stamp: %s' % str(row[1]))
        print('Message: %s' % str(row[3]))
        print('Description: %s' % str(row[4]))
        print('')

def searchFile(filepath,
               bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to search SEREBO Black Box for a file logging
    event - This does not insert a record into SEREBO Black
    Box.

    Usage:

        python serebo.py searchfile
        --filepath=<path to file for searching>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py searchfile
        --filepath=doxygen_serebo
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param fileapth String: Path of file to search in SEREBO
    black box.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    filepath = str(filepath)
    absPath = bb.absolutePath(filepath)
```

```
    fHash = bb.fileHash(absPath)
    result = bb.searchDatalog(db, fHash, 'data', 'exact')
    print('')
    print('Search Result (Search by File) ...')
    print('')
    print('File Path: %s' % filepath)
    print('Absolute File Path: %s' % absPath)
    print('')
    for row in result:
        print('Date Time Stamp: %s' % str(row[1]))
        print('Message: %s' % str(row[3]))
        print('Description: %s' % str(row[4]))
        print('')

def auditCount(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to check for equal numbers of records in data log
    and blockchain in SEREBO Black Box - should have the same
    number of records. This does not insert a record into
    SEREBO Black Box.

    Usage:

        python serebo.py audit_count
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py audit_count \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    sqlstmtA = 'select ID, dtstamp from datalog'
    sqlresultA = {}
    for row in db.cur.execute(sqlstmtA):
        sqlresultA[row[0]] = row[1]
    sqlstmtB = 'select c_ID, c_dtstamp from blockchain'
    sqlresultB = {}
    for row in db.cur.execute(sqlstmtB):
        sqlresultB[row[0]] = row[1]
    print('')
    print('Audit SEREBO Black Box Data Count ...')
    print('')
    if len(sqlresultA) == len(sqlresultB):
        for k in sqlresultA:
```

```
            if sqlresultA[k] != sqlresultB[k]:
                print('Date time stamp mismatch')
                print('Datalog record number %s' % str(k))
                print('Datalog date time stamp: %s' % \
                    str(sqlresultA[k]))
                print('Blockchain date time stamp: %s' % \
                    str(sqlresultB[k]))
            else:
                print('Date time stamp match - Record %s' % \
                      str(k))
        print('Number of records in datalog matches the number
            of records in blockchain')
    else:
        if len(sqlresultA) > len(sqlresultB):
            print('Number of records in datalog MORE than the
                number of records in blockchain')
        elif len(sqlresultA) < len(sqlresultB):
            print('Number of records in datalog LESS than the
                number of records in blockchain')

def auditDatahash(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to check for accuracy of hash generations in data
    log within SEREBO Black Box - recorded hash in data log
    and computed hash should be identical. This does not insert
    a record into SEREBO Black Box.

    Usage:

        python serebo.py audit_datahash
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py audit_datahash \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    sqlstmt = '''select ID, dtstamp, data, description, hash
        from datalog'''
    print('')
    print('Audit SEREBO Black Box Data Log Records ...')
    print('')
    for row in db.cur.execute(sqlstmt):
        ID = str(row[0])
```

```
        dtstamp = str(row[1])
        data = str(row[2])
        description = str(row[3])
        rHash = str(row[4])
        dhash = bytes(dtstamp, 'utf-8') + \
                bytes(data, 'utf-8') + \
                bytes(description, 'utf-8')
        tHash = db.hash(dhash)
        if tHash == rHash:
            print('Verified record %s in data log' % ID)
        else:
            print('ERROR in record %s in data log' % ID)
            print('Hash in record: %s' % rHash)
            print('Computed hash: %s' % tHash)

def dumpHash(outputf,
             bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to write out record hash from SEREBO Black Box
    into a file - This does not insert a record into SEREBO
    Black Box.

    Usage:

        python serebo.py dumphash
        --outputf=<output file path>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py dumphash \
        --outputf=sereboBB_hash \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param outputf String: Output file path. Default =
    sereboBB_hash
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    outputf = str(outputf)
    outputf = bb.absolutePath(outputf)
    outf = open(outputf, 'w')
    sqlstmt = 'select ID, dtstamp, hash from datalog'
    count = 0
    for row in db.cur.execute(sqlstmt):
        data = [str(row[0]), str(row[1]), str(row[2])]
        data = ' | '.join(data)
```

```
        outf.write(data + '\n')
        count = count + 1
    outf.close()
    print('')
    print('Dump SEREBO Black Box Data Log Hashes ...')
    print('')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Output File Path': outputf,
            'Number of Records': str(count)}

def auditDataBlockchain(bbpath='serebo_blackbox\\
                        blackbox.sdb'):
    '''!
    Function to check for accuracy in data log and blockchain
    mapping in SEREBO Black Box - recorded hash in data log
    and data in blockchain should be identical. This does not
    insert a record into SEREBO Black Box.

    Usage:

        python serebo.py audit_data_blockchain
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py audit_data_blockchain \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    sqlstmt = '''select datalog.ID, datalog.dtstamp,
        datalog.hash, blockchain.c_dtstamp, blockchain.data
        from datalog inner join blockchain where
        datalog.ID=blockchain.c_ID and
        datalog.dtstamp=blockchain.c_dtstamp'''
    print('')
    print('Audit SEREBO Black Box - Accuracy in Data Log to
        Blockchain Mapping...')
    print('')
    for row in db.cur.execute(sqlstmt):
        dID = str(row[0])
        ddtstamp = str(row[1])
        dhash = str(row[2])
        bdtstamp = str(row[3])
        bhash = str(row[4])
```

```
        if dhash == bhash:
            print('Verified record %s mapping' % dID)
        else:
            print('ERROR in record %s mapping' % dID)
            print('Hash in Data Log: %s' % dHash)
            print('Data in Blockchain: %s' % bHash)

def auditBlockchainHash(bbpath='serebo_blackbox\\
                        blackbox.sdb'):
    '''!
    Function to check for accuracy in blockchain hash
    Generation within SEREBO Black Box - recorded hash in
    blockchain and computed hash should be identical. This
    does not insert a record into SEREBO Black Box.

    Usage:

        python serebo.py audit_blockchainhash
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py audit_blockchainhash
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    sqlstmt = '''select c_ID, p_dtstamp, p_randomstring,
        p_hash, data, c_hash from blockchain'''
    print('')
    print('Audit SEREBO Black Box Blockchain hashes ...')
    print('')
    for row in db.cur.execute(sqlstmt):
        ID = str(row[0])
        p_dtstamp = str(row[1])
        p_randomstring = str(row[2])
        p_hash = str(row[3])
        data = str(row[4])
        c_hash = str(row[5])
        dhash = ''.join([str(p_dtstamp), str(p_randomstring),
                         str(p_hash), str(data)])
        dhash = bytes(dhash, 'utf-8')
        tHash = db.hash(dhash)
        if tHash == c_hash:
            print('Verified record %s in Blockchain' % ID)
        else:
```

```
            print('ERROR in record %s in Blockchain' % ID)
            print('Hash in record: %s' % c_hash)
            print('Computed hash: %s' % tHash)

def checkHash(hashfile,
              bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to compare record hash from SEREBO Black Box with
    that in a hash file. This does not insert a record into
    SEREBO Black Box.

    Usage:

        python serebo.py checkhash
        --hashfile=<path to hash file>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py checkhash \
        --hashfile=sereboBB_hash \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param hashfile String: File path to hash file.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    hashfile= str(hashfile)
    hashfile = bb.absolutePath(hashfile)
    print('')
    print('Compare record hash from SEREBO Black Box with that
        in a hash file...')
    print('')
    hf = open(hashfile, 'r')
    for record in hf:
        record = [str(d.strip())
                  for d in record[:-1].split('|')]
        ID = record[0]
        dtstamp = record[1]
        thash = record[2]
        sqlstmt = """select hash from datalog where ID='%s'
            and dtstamp='%s'""" % (ID, dtstamp)
        dhash = [row for row in db.cur.execute(sqlstmt)][0][0]
        dhash = str(dhash)
        if thash == dhash:
            print('Verified record %s hash between Data Log
                and Hash file' % ID)
```

```
        else:
            print('ERROR in record %s' % ID)
            print('Hash in Hash File: %s' % thash)
            print('Hash in Data Log: %s' % dhash)

def auditBlockchainFlow(bbpath='serebo_blackbox\\
                        blackbox.sdb'):
    '''!
    Function to trace the decendancy of blockchain records
    (also known as blocks) within SEREBO Black Box - decandency
    from first block should be traceable to the last / latest
    block. This does not insert a record into SEREBO Black
    Box.

    Usage:

        python serebo.py audit_blockchainflow
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py audit_blockchainflow
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    sqlstmt = '''select max(c_ID) from blockchain'''
    print('')
    print("Trace SEREBO Black Box Blockchain's block
        decendancy ...")
    print('')
    maxID = [row for row in db.cur.execute(sqlstmt)][0][0]
    maxID = int(maxID)
    for i in range(1, maxID, 1):
        # Get parent data from parent block
        sqlstmt = """select c_ID, c_dtstamp, c_randomstring,
            c_hash from blockchain where c_ID=%s""" % str(i)
        #print(sqlstmt)
        p_data = [row for row in db.cur.execute(sqlstmt)][0]
        pc_ID = str(p_data[0])
        pc_dtstamp = str(p_data[1])
        pc_randomstring = str(p_data[2])
        pc_hash = str(p_data[3])
        # Get parent data from current / child block
        sqlstmt = """select p_ID, p_dtstamp, p_randomstring,
            p_hash from blockchain where c_ID=%s""" % str(i+1)
```

```
        #print(sqlstmt)
        c_data = [row for row in db.cur.execute(sqlstmt)][0]
        p_ID = str(c_data[0])
        p_dtstamp = str(c_data[1])
        p_randomstring = str(c_data[2])
        p_hash = str(c_data[3])
        # Compare parental block record and parent data in
        # current record
        if (p_ID == pc_ID) and \
            (p_dtstamp == pc_dtstamp) and \
            (p_randomstring == pc_randomstring) and \
            (p_hash == pc_hash):
            print('Verified - Record %s was used as parent
                record in record %s' % (str(i), str(i+1)))
        else:
            print('ERROR in record %s' % str(i+1))
            print('Parent ID in record %s: %s' % \
                (str(i+1), str(i)))
            print('Parent date time stamp in record %s: %s' \
                % (str(i+1), p_dtstamp))
            print('Actual date time stamp in record %s: %s' \
                % (str(i), pc_dtstamp))
            print('Parent random string in record %s: %s' % \
                (str(i+1), p_randomstring))
            print('Actual random string in record %s: %s' % \
                (str(i), pc_randomstring))
            print('Parent hash in record %s: %s' % \
                (str(i+1), p_hash))
            print('Actual hash in record %s: %s' % \
                (str(i), pc_hash))

def NTPSign(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to self-sign (self notarization) SEREBO Black Box
    using NTP (Network Time Protocol) server.

    Usage:

        python serebo.py ntpsign
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py ntpsign
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
```

```
    '''
    db = bb.connectDB(bbpath)
    ntp = bb.ntplib.NTPClient()
    rstring = bb.randomString(db, 32)
    response = ntp.request('pool.ntp.org', version=3)
    dtstamp = bb.gmtime(response.tx_time)
    ntp_ip = bb.ntplib.ref_id_to_text(response.ref_id)
    description = ['NTP server (self) notarization',
                   'Seconds Since Epoch: %s' % \
                       str(response.tx_time),
                   'NTP Date Time: %s' % str(dtstamp),
                   'NTP Server IP: %s' % str(ntp_ip)]
    description = ' | '.join(description)
    rdata = bb.insertFText(db, rstring, description)
    print('')
    print('Self-Signing / Self-Notarization ...')
    print('')
    return {'SEREBO Black Box': db,
            'Black Box Path': str(db.path),
            'Date Time Stamp': str(rdata['DateTimeStamp']),
            'Random String': str(rstring),
            'Seconds Since Epoch': str(response.tx_time),
            'NTP Date Time': str(dtstamp),
            'NTP Server IP': str(ntp_ip)}

def backup(backuppath='blackbox_backup.sdb',
           bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to backup SEREBO Black Box - This does not insert
    A record into SEREBO Black Box.

    Usage:

        python serebo.py backup
        --backuppath=<path for backed-up SEREBO black box>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py backup
        --backuppath='blackbox_backup.sdb'
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param backuppath String: Path for backed-up SEREBO black
    box. Default = 'blackbox_backup.sdb'
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
```

```
    print('')
    print('Backup SEREBO Black Box ...')
    print('')
    if backuppath != bbpath:
        (bbpath, backuppath) = bb.backup(bbpath, backuppath)
        return {'Black Box Path': bbpath,
                'Backup Path': backuppath}
    else:
        print('Backup path cannot be the same as SEREBO Black
            Box path')
        bbpath = bb.absolutePath(bbpath)
        backuppath = bb.absolutePath(backuppath)
        return {'Black Box Path': bbpath,
                'Backup Path': backuppath}

def dump(dumpfolder='.', fileprefix='dumpBB',
         bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to dump individual data tables from SEREBO Black
    Box into text files - This does not insert a record into
    SEREBO Black Box.

    Usage:

        python serebo.py dump
        --dumpfolder=<folder to save dump files>
        --fileprefix=<prefix for individual dump files>
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py dump
        --dumpfolder='.'
        --fileprefix='dumpBB'
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param dumpfolder String: Folder to save dump files.
    Default = '.' (current working directory).
    @param fileprefix String: Prefix for individual dump
    files. Default = 'dumpBB'.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    tableSet = {'metadata': ['key', 'value'],
                'notary': ['dtstamp',
                           'alias',
                           'owner',
```

```
                            'email',
                            'notaryDTS',
                            'notaryAuthorization',
                            'notaryURL'],
                  'systemdata': ['dtstamp', 'key', 'value'],
                  'datalog': ['dtstamp',
                              'hash',
                              'data',
                              'description'],
                  'blockchain': ['c_ID', 'c_dtstamp',
                                 'c_randomstring', 'c_hash',
                                 'p_ID', 'p_dtstamp',
                                 'p_randomstring', 'p_hash',
                                 'data'],
                  'eventlog': ['dtstamp', 'fID',
                               'description'],
                  'eventlog_datamap': ['dtstamp', 'fID',
                                       'key', 'value']}
    print('')
    print('Dump out data (text backup) from SEREBO Black Box
        ...')
    print('')
    for tableName in tableSet:
        outputfile = [dumpfolder,
                      fileprefix + '_' + tableName + '.csv']
        outputfile = os.sep.join(outputfile)
        (outputfile, count) = bb.dumpTable(db, tableName,
                                           tableSet[tableName],
                                           outputfile)
        print('%s table dumped into %s' % \
            (tableName, outputfile))
        print('Number of records dumped: %s' % count)
        print('')

def auditRegister(alias,
                  bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to check for SEREBO Black Box registration with
    SEREBO Notary - This does not insert a record into SEREBO
    Black Box.

    Usage:

        python serebo.py audit_register \
        --alias=<alias for SEREBO Notary> \
        --bbpath=<path to SEREBO black box>

    For example:
```

```
        python serebo.py audit_register \
        --alias="NotaryPythonAnywhere" \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param alias String: Alias for this SEREBO Notary.
    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    sqlstmt = "select value from metadata where
        key='blackboxID'"
    blackboxID = [row
                  for row in db.cur.execute(sqlstmt)][0][0]
    try:
        sqlstmt = "select notaryAuthorization, notaryURL from
notary where alias='%s'" % str(alias)
        sqlresult = [row
                     for row in db.cur.execute(sqlstmt)][0]
        notaryAuthorization = sqlresult[0]
        notaryURL = sqlresult[1]
    except IndexError:
        print('Notary authorization or Notary URL not found
            for the given alias')
        return {'SEREBO Black Box': db,
                'Black Box Path': str(db.path),
                'Notary Alias': str(alias)}
    try:
        presence = notary.checkRegistration(blackboxID,
                       notaryAuthorization, notaryURL)
        if presence:
            message = 'Registration found in SEREBO Notary'
        else:
            message = 'Registration NOT found in SEREBO
                Notary'
        print('')
        print('Checking SEREBO Black Box registration in
            SEREBO Notary...')
        return {'SEREBO Black Box': db,
                'Black Box Path': str(db.path),
                'Notary Alias': str(alias),
                'Notary URL': str(notaryURL),
                'Notary Authorization': \
                    str(notaryAuthorization),
                'Status': message}
    except:
        print('Failed in checking SEREBO Black Box
            registration in SEREBO Notary')
```

```
        return {'SEREBO Black Box': db,
                'Black Box Path': str(db.path),
                'Notary Alias': str(alias),
                'Notary URL': str(notaryURL),
                'Notary Authorization': \
                    str(notaryAuthorization)}

def viewSelfNotarizations(bbpath='serebo_blackbox\\
                          blackbox.sdb'):
    '''!
    Function to view all self notarizations for this SEREBO
    Black Box - This does not insert a record into SEREBO Black
    Box.

    Usage:

        python serebo.py viewselfnote
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py viewselfnote
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    print('')
    print('Black Box Path: %s' % str(bbpath))
    sqlstmt = """select dtstamp, data from datalog where
        description like 'Self notarization'"""
    print('')
    print('Self Notarization(s) ...')
    for row in db.cur.execute(sqlstmt):
        print('')
        print('Date Time Stamp: %s' % str(row[0]))
        print('Hash: %s' % str(row[1]))

def viewNTPNotarizations(bbpath='serebo_blackbox\\
                         blackbox.sdb'):
    '''!
    Function to view all self-notarization(s) by NTP time
    server for this SEREBO Black Box - This does not insert a
    record into SEREBO Black Box.

    Usage:
```

```
        python serebo.py viewntpnote
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py viewntpnote
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    print('')
    print('Black Box Path: %s' % str(bbpath))
    sqlstmt = """select dtstamp, data, description from
        datalog where description like 'NTP server (self)
        notarization%'"""
    print('')
    print('Self-Notarization(s) by NTP Time Server(s) ...')
    for row in db.cur.execute(sqlstmt):
        description = [x.strip()
                       for x in str(row[2]).split('|')]
        print('')
        print('Date Time Stamp: %s' % str(row[0]))
        print('Random Code: %s' % str(row[1]))
        print('NTP Seconds Since Epoch: %s' % description[1])
        print('NTP Date Time: %s' % description[2])
        print('NTP Server IP: %s' % description[3])

def viewNotaryNotarizations(bbpath='serebo_blackbox\\
                            blackbox.sdb'):
    '''!
    Function to view all notarizations by SEREBO Notary for
    this SEREBO Black Box - This does not insert a record into
    SEREBO Black Box.

    Usage:

        python serebo.py viewsnnote \
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py viewsnnote \
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
```

```
    '''
    db = bb.connectDB(bbpath)
    print('')
    print('Black Box Path: %s' % str(bbpath))
    sqlstmt = """select dtstamp, data, description from
        datalog where description like 'Notarization with
        SEREBO Notary%'"""
    print('')
    print('Notarization(s) by SEREBO Notary(ies) ...')
    for row in db.cur.execute(sqlstmt):
        description = [x.strip()
                       for x in str(row[2]).split('|')]
        print('')
        print('Date Time Stamp: %s' % str(row[0]))
        print('Common Code: %s' % str(row[1]))
        print(description[1]) # Black Box Code
        print(description[2]) # Black Box Date Time
        print(description[3]) # Notary Code
        print(description[4]) # Notary Date Time
        print(description[5]) # Notary URL

def _auditSingleNotarizeBB(blackboxID, notaryAuthorization,
                           notaryURL, BBCode, NCode,
                           CommonCode):
    '''!
    Private function - communicate with SEREBO Notary to check
    for SEREBO Black Box notarization record.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param notaryAuthorization String: Notary authorization
    code of SEREBO black box (generated during black box
    registration - found in metadata table in SEREBO black box
    database.
    @param notaryURL String: URL for SEREBO Notary web
    service.
    @param BBCode String: Notarization code from SEREBO Black
    Box.
    @param NCode String: Notarization code from SEREBO Notary.
    @param CommonCode String: Cross-Signing code from SEREBO
    Notary.
    @returns: 'True' if SEREBO Black Box notarization is found
    in SEREBO Notary. 'False' if SEREBO Black Box notarization
    is not found in SEREBO Notary. 'Failed' if there is any
    errors, such as network error.
    '''
    try:
        presence = notary.checkNotarization(blackboxID,
```

```
                                            notaryAuthorization,
                                                BBCode,
                                                NCode,
                                                CommonCode,
                                                notaryURL)
            return presence
        except:
            return 'Failed'

def auditNotarizeBB(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to view all notarizations by SEREBO Notary for
    this SEREBO Black Box - This does not insert a record into
    SEREBO Black Box.

    Usage:

        python serebo.py audit_notarizebb
        --bbpath=<path to SEREBO black box>

    For example:

        python serebo.py audit_notarizebb
        --bbpath='serebo_blackbox\\blackbox.sdb'

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    '''
    db = bb.connectDB(bbpath)
    sqlstmt = "select value from metadata where
        key='blackboxID'"
    blackboxID = [row
                  for row in db.cur.execute(sqlstmt)][0][0]
    print('')
    print('Black Box Path: %s' % str(bbpath))
    sqlstmtA = """select dtstamp, data, description from
        datalog where description like 'Notarization with
        SEREBO Notary%'"""
    dataA = [row for row in db.cur.execute(sqlstmtA)]
    print('')
    print('Notarization(s) by SEREBO Notary(ies) ...')
    for row in dataA:
        description = [x.strip()
                       for x in str(row[2]).split('|')]
        try:
            notaryURL = description[5].split(': ')[1].strip()
            sqlstmt = "select notaryAuthorization from notary
                where notaryURL='%s'" % str(notaryURL)
```

```
            sqlresult = [row
                         for row in db.cur.execute(sqlstmt)][0]
            notaryAuthorization = sqlresult[0]
        except IndexError:
            print('Notary authorization not found for the
                given Notary URL')
            return {'SEREBO Black Box': db,
                    'Black Box Path': str(db.path),
                    'Notary URL': str(notaryURL)}
        presence = _auditSingleNotarizeBB(blackboxID,
                       notaryAuthorization, notaryURL,
                       description[1].split(': ')[1].strip(),
                       description[3].split(': ')[1].strip(),
                       str(row[1]))
        if presence == 'True':
            message = 'Notarization record is found in SEREBO
                Notary'
        elif presence == 'False':
            message = 'Notarization record is NOT found in
                SEREBO Notary'
        elif presence == 'Failed':
            message = 'Unspecified error - does not mean that
                notarization record is not found. It may mean
                network error.'
        print('')
        print('Date Time Stamp: %s' % str(row[0]))
        print('Common Code: %s' % str(row[1]))
        print(description[1]) # Black Box Code
        print(description[2]) # Black Box Date Time
        print(description[3]) # Notary Code
        print(description[4]) # Notary Date Time
        print(description[5]) # Notary URL
        print('Status: %s' % message)


if __name__ == '__main__':
    exposed_functions = {\
          'audit_blockchainflow': auditBlockchainFlow,
          'audit_blockchainhash': auditBlockchainHash,
          'audit_count': auditCount,
          'audit_data_blockchain': auditDataBlockchain,
          'audit_datahash': auditDatahash,
          'audit_notarizebb': auditNotarizeBB,
          'audit_register': auditRegister,
          'backup': backup,
          'changealias': changeAlias,
          'checkhash': checkHash,
          'dump': dump,
```

```
        'dumphash': dumpHash,
        'fhash': fileHash,
        'init': initialize,
        'intext': insertText,
        'localcode': localCode,
        'localdts': localDTS,
        'logfile': logFile,
        'notarizebb': notarizeBlackbox,
        'ntpsign': NTPSign,
        'register': registerBlackbox,
        'searchmsg': searchMessage,
        'searchdesc': searchDescription,
        'searchfile': searchFile,
        'selfsign': selfSign,
        'shash': stringHash,
        'sysdata': systemData,
        'sysrecord': systemRecord,
        'viewntpnote': viewNTPNotarizations,
        'viewselfnote': viewSelfNotarizations,
        'viewsnnote': viewNotaryNotarizations,
        'viewreg': viewRegistration}
  fire.Fire(exposed_functions)
```

FILE NAME: SEREBO_NOTARY_API.PY

```
'''!
Secured Recorder Box (SEREBO) Notary Communicator

Date created: 19th May 2018

License: GNU General Public License version 3 for academic or
not-for-profit use only


SEREBO 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
<http://www.gnu.org/licenses/>.
'''
```

```
from xmlrpc.client import ServerProxy

def registerBlackbox(blackboxID, owner, email,
                     architecture, machine, node,
                     platform, processor,
                     notaryURL='https://mauricelab.
                     pythonanywhere.com/serebo_notary/
                     services/call/xmlrpc'):
    '''!
    Function to communicate with SEREBO Notary to register
    SEREBO Black Box with SEREBO Notary.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param owner String: Owner's or administrator's name.
    @param email String: Owner's or administrator's email.
    @param architecture String: Architecture of this machine
    - from platform library in Python Standard Library.
    @param machine String: This machine description - from
    Platform library in Python Standard Library.
    @param node String: This machine's node description - from
    platform library in Python Standard Library.
    @param platform String: This platform description - from
    platform library in Python Standard Library.
    @param processor String: Machine's processor description
    - from platform library in Python Standard Library
    @param notaryURL String: URL for SEREBO Notary web
    service. Default="https://mauricelab.pythonanywhere.com/
    serebo_notary/services/call/xmlrpc"
    @returns: (URL of SEREBO Notary, Notary authorization
    code, Date time stamp from SEREBO Notary)
    '''
    serv = ServerProxy(notaryURL)
    (notaryAuthorization, dtstamp) = \
        serv.register_blackbox(blackboxID, owner, email,
                               architecture, machine, node,
                               platform, processor)
    return (notaryURL,
            str(notaryAuthorization),
            str(dtstamp))

def notarizeBB(blackboxID, notaryAuthorization, dtstampBB,
               codeBB, notaryURL='https://mauricelab.
               pythonanywhere.com/serebo_notary/
               services/call/xmlrpc'):
    '''!
    Function to communicate with SEREBO Notary to notarize
```

```
    SEREBO Black Box with SEREBO Notary.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param notaryAuthorization String: Notary authorization
    code of SEREBO black box (generated during black box
    registration - found in metadata table in SEREBO black box
    database.
    @param dtstampBB String: Date time stamp from SEREBO black
    box.
    @param codeBB String: Notarization code from SEREBO black
    box.
    @param notaryURL String: URL for SEREBO Notary web
    service. Default="https://mauricelab.pythonanywhere.com/
    serebo_notary/services/call/xmlrpc"
    @returns: (URL of SEREBO Notary, Date time stamp from
    SEREBO Notary, Notarization code from SEREBO Notary,
    Cross-Signing code from SEREBO Notary)
    '''
    serv = ServerProxy(notaryURL)
    (dtstampNS, codeNS, codeCommon) = \
        serv.notarizeSereboBB(blackboxID,
        notaryAuthorization, dtstampBB, codeBB)
    return (notaryURL, str(dtstampNS),
            str(codeNS), str(codeCommon))

def checkRegistration(blackboxID, notaryAuthorization,
    notaryURL='https://mauricelab.pythonanywhere.com/
    serebo_notary/services/call/xmlrpc'):
    '''!
    Function to communicate with SEREBO Notary to check for
    SEREBO Black Box registration record.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param notaryAuthorization String: Notary authorization
    code of SEREBO black box (generated during black box
    registration – found in metadata table in SEREBO black box
    database.
    @param notaryURL String: URL for SEREBO Notary web
    service. Default="https://mauricelab.pythonanywhere.com/
    serebo_notary/services/call/xmlrpc"
    @returns: Boolean flag - True if SEREBO Black Box
    registration is found in SEREBO Notary. False if SEREBO
    Black Box registration is not found in SEREBO Notary.
    '''
    serv = ServerProxy(notaryURL)
    value = serv.checkBlackBoxRegistration(blackboxID,
```

```
                notaryAuthorization)
    if value or value == 'True':
        return True
    elif not value or value == 'False':
        return False

def checkNotarization(blackboxID, notaryAuthorization,
    BBCode, NCode, CommonCode,
    notaryURL='https://mauricelab.pythonanywhere.com/
    serebo_notary/services/call/xmlrpc'):
    '''!
    Function to communicate with SEREBO Notary to check for
    SEREBO Black Box notarization record.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param notaryAuthorization String: Notary authorization
    code of SEREBO black box (generated during black box
    registration - found in metadata table in SEREBO black box
    database.
    @param BBCode String: Notarization code from SEREBO Black
    Box.
    @param NCode String: Notarization code from SEREBO Notary.
    @param CommonCode String: Cross-Signing code from SEREBO
    Notary.
    @param notaryURL String: URL for SEREBO Notary web
    service. Default="https://mauricelab.pythonanywhere.com/
    serebo_notary/services/call/xmlrpc"
    @returns: Boolean flag - True if SEREBO Black Box
    notarization is found in SEREBO Notary. False if SEREBO
    Black Box notarization is not found in SEREBO Notary.
    '''
    serv = ServerProxy(notaryURL)
    value = serv.checkNotarizeSereboBB(blackboxID,
                                       notaryAuthorization,
                                       BBCode, NCode,
                                       CommonCode)
    if value or value == 'True':
        return 'True'
    elif not value or value == 'False':
        return 'False'
```

CODE FILES FOR SEREBO BLACKBOX

FILE NAME: __INIT__.PY

```
'''!
Secured Recorder Box (SEREBO) Black Box

Date created: 17th May 2018

License: GNU General Public License version 3 for academic or
not-for-profit use only


SEREBO 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
<http://www.gnu.org/licenses/>.
'''

from datetime import datetime

# Metadata
__version__ = '1.0'
__author__ = 'Maurice H.T. Ling <mauriceling@acm.org>'
__maintainer__ = 'Maurice H.T. Ling <mauriceling@acm.org>'
__email__ = 'mauriceling@acm.org'
__copyright__ = '(c) 2018-%s, Maurice H.T. Ling.' %
(datetime.now().year)
__description__ = '''
SEREBO (SEcured REcorder BOx) Black Box is inspired by the
black boxes (cockpit voice recorder and flight data recorder)
in airliners. The intended purpose is to track and audit
research records under the following premise - Given a set of
data files, is there a system to log and verify that these
files had not been changed or edited since its supposed
creation?

SEREBO Black Box aims to address this issue using several
approaches. Firstly, the data files can be used to generate a
```

```
file hash. It is very likely that an edit in the file will
result in a different hash. Hence, if a file generates the
same hash across two different points in time, it can be safely
assumed that the file had not been edited during this time
span. Secondly, the file hash has to be securely recorded with
amendment protected. SEREBO records the hash and registers the
hash into a blockchain. The main concept of blockchain is that
the hash of previous (parent) block is concatenated with the
data (file hash in this case) of the current block to generate
a hash for the current block. Hence, as the blockchain grows,
any amendments in earlier blocks can be easily detected - only
amendments to the latest block cannot be detected. Therefore,
the value of SEREBO lies in its use.'''

from . import ntplib
from . import serebo_api
from .serebo_api import absolutePath
from .serebo_api import backup
from .serebo_api import connectDB
from .serebo_api import dateTime
from .serebo_api import dumpTable
from .serebo_api import fileHash
from .serebo_api import gmtime
from .serebo_api import insertFText
from .serebo_api import insertText
from .serebo_api import logFile
from .serebo_api import randomString
from .serebo_api import searchDatalog
from .serebo_api import stringHash
from .serebo_api import systemData
```

FILE NAME: SEREBO_API.PY

```
'''!
Secured Recorder Box (SEREBO) Application Programming
Interface (API)

Date created: 17th May 2018

License: GNU General Public License version 3 for academic or
not-for-profit use only
```

```
SEREBO 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
<http://www.gnu.org/licenses/>.
'''
import hashlib
import random
import secrets
import os.path
import time

from . import sereboDB
from .sereboDB import SereboDB

def connectDB(bbpath='serebo_blackbox\\blackbox.sdb'):
    '''!
    Function to connect to SEREBO database - the recorder box.

    @param bbpath String: Path to SEREBO black box. Default =
    'serebo_blackbox\\blackbox.sdb'.
    @return: SEREBO database object
    '''
    bbpath = os.path.abspath(bbpath)
    db = SereboDB(bbpath)
    return db

def systemData():
    '''!
    Function to extract data and test hashes of current
    platform.

    @return: Dictionary of system data and test hashs
    '''
    import platform
    data = {\
```

```
        'architecture': ':'.join(platform.architecture()),
        'machine': platform.machine(),
        'node': platform.node(),
        'platform': platform.platform(),
        'processor': platform.processor(),
        'python_build': ' > '.join(platform.python_build()),
        'python_compiler': platform.python_compiler(),
        'python_implementation': \
            platform.python_implementation(),
        'python_branch': platform.python_branch(),
        'python_revision': platform.python_revision(),
        'python_version': platform.python_version(),
        'release': platform.release(),
        'system': platform.system(),
        'version': platform.version()
        }
    data['hashdata'] = \
        bytes('''yd6jwAYeqHmzSyxkNOVXTGtDr8dgZIE9LoL9jxRUbq
        OEuODCysfeJkLJHy3LuQX3Rp4f1Ms5HcfTDAyjdLSpNVJx2vbks
        BKAAi5VVkhW7MJ9CtlfZBlBvCYbX8Qk8Jw27fsglmaPmbR9BZQo
        FpuSQxCDF77dmCcbqw5WiKfuTQiUl9PeyHemnMVtsRGKfN2c0x0
        BA54HjOyN30Dy86fJhitrhLsW3wIY9PtzFEcXd1rq36cFKfrNp7
        lRjJzDJ4W8ZCuQY6P3HUM8Eu4fsGytH9WlVmJ1aJGiyPVf1ZAa4
        2yKUnfBUwhFNU1aEtplVeHrQqQvO7tLxyE5Oc8TjRF7sAzQozjV
        bNyhVlxOmhI45pX4qtBA9y9XrHfYJP9RJaprTsnR24g1pOjxVyp
        zEjSGVEh7EKYWXk7fLllwWRkAb7rG5HSEH5gmcsvbpTNNEsXfcm
        myrvvh6i7cfQGPap2XmxjO6VRZg1hkf7yUarltZ1kTdD3pMJRBo
        PpPijuqB1uA''', 'utf-8')
    data['hash_md5'] = \
        hashlib.md5(data['hashdata']).hexdigest()
    data['hash_sha1'] = \
        hashlib.sha1(data['hashdata']).hexdigest()
    data['hash_sha224'] = \
        hashlib.sha224(data['hashdata']).hexdigest()
    data['hash_sha3_224'] = \
        hashlib.sha3_224(data['hashdata']).hexdigest()
    data['hash_sha256'] = \
        hashlib.sha256(data['hashdata']).hexdigest()
    data['hash_sha3_256'] = \
        hashlib.sha3_256(data['hashdata']).hexdigest()
    data['hash_sha384'] = \
        hashlib.sha384(data['hashdata']).hexdigest()
    data['hash_sha3_384'] = \
        hashlib.sha3_384(data['hashdata']).hexdigest()
```

```
    data['hash_sha512'] = \
        hashlib.sha512(data['hashdata']).hexdigest()
    data['hash_sha3_512'] = \
        hashlib.sha3_512(data['hashdata']).hexdigest()
    data['hash_blake2b'] = \
        hashlib.blake2b(data['hashdata']).hexdigest()
    data['hash_blake2s'] = \
        hashlib.blake2s(data['hashdata']).hexdigest()
    return data

def insertText(sdb_object, text, description='NA'):
    '''!
    Function to insert text string into SEREBO database, with
    10-random character string suffixing the description.

    A dictionary of items generated will be returned with the
    following keys: (1) DateTimeStamp is the UTC date time
    stamp of this event, (2) Data is the given data string to
    be inserted, (3) UserDescription is the user given
    explanation string for this event suffixed with a 10-
    character random string, (4) DataHash is the hash string
    of Data, (5) ParentBlockID is the ID of the parent block
    in blockchain, (6) ParentDateTimeStamp is the UTC date
    time stamp of the parent block in blockchain (which is
    also the parent insertion event), (7) ParentRandomString
    is the random string generated in parent block in
    blockchain, (8) ParentHash is the hash of parent block in
    blockchain, (9) BlockRandomString is the random string
    generated for current insertion event, and (10) BlockHash
    is the block hash of current insertion event in blockchain.

    @param sdb_object Object: SEREBO database object.
    @param text String: Text string to be inserted.
    @param description String: Explanation string for this
    Entry event. Default = NA.
    @return: Dictionary of data generated from this event.
    '''
    rdata = sdb_object.insertData(text, description, 'text')
    return rdata

def insertFText(sdb_object, text, description='NA'):
    '''!
    Function to insert text string into SEREBO database,
    without 10-random character string suffixing the
```

```
    description.

    A dictionary of items generated will be returned with the
    following keys: (1) DateTimeStamp is the UTC date time
    stamp of this event, (2) Data is the given data string to
    be inserted, (3) UserDescription is the user given
    explanation string for this event, (4) DataHash is the
    hash string of Data, (5) ParentBlockID is the ID of the
    parent block in blockchain, (6) ParentDateTimeStamp is the
    UTC date time stamp of the parent block in blockchain
    (which is also the parent insertion event), (7)
    ParentRandomString is the random string generated in
    parent block in blockchain, (8) ParentHash is the hash of
    parent block in blockchain, (9) BlockRandomString is the
    random string generated for current insertion event, and
    (10) BlockHash is the block hash of current insertion event
    in blockchain.

    @param sdb_object Object: SEREBO database object.
    @param text String: Text string to be inserted.
    @param description String: Explanation string for this
    entry event. Default = NA.
    @return: Dictionary of data generated from this event.
    '''
    rdata = sdb_object.insertData(text, description, 'ftext')
    return rdata

def absolutePath(filepath):
    '''!
    Function to convert file path (absolute or relative file
    path) into absolute file path.

    @param filepath String: File path to be converted.
    @return: Absolute file path.
    '''
    return os.path.abspath(filepath)

def fileHash(filepath):
    '''!
    Function to generate a series of 12 hashes for a given
    file, in the format of <MD5>:<SHA1>:<SHA224>:<SHA3
    244>:<SHA256>:<SHA3 256>:<SHA384>:<SHA3 384>:<SHA512>:
    <SHA3 215>:<Blake 2b>:<Blake 2s>.
```

```
    @param filepath String: Path of file for hash generation.
    @return: Hash
    '''
    absPath = absolutePath(filepath)
    md5 = hashlib.md5()
    sha1 = hashlib.sha1()
    sha224 = hashlib.sha224()
    sha3_224 = hashlib.sha3_224()
    sha256 = hashlib.sha256()
    sha3_256 = hashlib.sha3_256()
    sha384 = hashlib.sha384()
    sha3_384 = hashlib.sha3_384()
    sha512 = hashlib.sha512()
    sha3_512 =  hashlib.sha3_512()
    blake2b = hashlib.blake2b()
    blake2s = hashlib.blake2s()
    with open(absPath, 'rb') as f:
        while True:
            data = f.read(65536)
            if not data:
                break
            md5.update(data)
            sha1.update(data)
            sha224.update(data)
            sha3_224.update(data)
            sha256.update(data)
            sha3_256.update(data)
            sha384.update(data)
            sha3_384.update(data)
            sha512.update(data)
            sha3_512.update(data)
            blake2b.update(data)
            blake2s.update(data)
    x = [md5.hexdigest(),
         sha1.hexdigest(),
         sha224.hexdigest(),
         sha3_224.hexdigest(),
         sha256.hexdigest(),
         sha3_256.hexdigest(),
         sha384.hexdigest(),
         sha3_384.hexdigest(),
         sha512.hexdigest(),
         sha3_512.hexdigest(),
         blake2b.hexdigest(),
```

```
            blake2s.hexdigest()]
    return ':'.join(x)

def logFile(sdb_object, filepath, description='NA'):
    '''!
    Function to logging a file into SEREBO database.

    A dictionary of items generated will be returned with the
    following keys: (1) DateTimeStamp is the UTC date time
    stamp of this event, (2) Data is the given data string to
    be inserted, (3) UserDescription is the user given
    explanation string for this event, (4) DataHash is the
    hash string of Data, (5) ParentBlockID is the ID of the
    parent block in blockchain, (6) ParentDateTimeStamp is the
    UTC date time stamp of the parent block in blockchain
    (which is also the parent insertion event), (7)
    ParentRandomString is the random string generated in
    parent block in blockchain, (8) ParentHash is the hash of
    parent block in blockchain, (9) BlockRandomString is the
    random string generated for current insertion event, and
    (10) BlockHash is the block hash of current insertion event
    in blockchain.

    @param sdb_object Object: SEREBO database object.
    @param fileapth String: Path of file to log in SEREBO black
    box.
    @param description String: Explanation string for this
    entry event. Default = NA.
    @return: Dictionary of data generated from this event.
    '''
    absPath = absolutePath(filepath)
    if description == 'NA':
        description = ['UserGivenPath:>%s' % str(filepath),
                       'AbsolutePath:>%s' % str(absPath)]
    else:
        description = ['UserGivenPath :> %s' % str(filepath),
                       'AbsolutePath :> %s' % str(absPath),
                       'UserDescription :> %s' % \
                           str(description)]
    description = ' >> '.join(description)
    fHash = fileHash(absPath)
    rdata = sdb_object.insertData(fHash, description, 'file')
    return rdata
```

```
def searchDatalog(sdb_object, term, field, mode='like'):
    '''!
    Function to search datalog table.

    @param sdb_object Object: SEREBO database object.
    @param term String: Case sensitive search term.
    @param field String: Field name to search.
    @param mode String: Mode of search. Allowable modes are
    'like' and 'exact'. If mode is 'like', wildcards such as
    '_' (matches any single character) and '%' (matches any
    number of characters). Default = 'like'.
    @return: List of datalog rows: [ID, dtstamp, hash, data,
    description]
    '''
    term = str(term)
    field = str(field)
    if mode.lower() == 'exact':
        sqlstmt = """select ID, dtstamp, hash, data,
            description from datalog where %s='%s'""" % \
            (field, term)
    if mode.lower() == 'like':
        sqlstmt = """select ID, dtstamp, hash, data,
            description from datalog where %s like '%s'""" %\
            (field, term)
    result = [row for row in sdb_object.cur.execute(sqlstmt)]
    return result

def dateTime(sdb_object):
    '''!
    Function to get a date time string.

    @param sdb_object Object: SEREBO database object.
    @return: Date time string
    '''
    return sdb_object.dtStamp()

def randomString(sdb_object, length):
    '''!
    Function to get a random string.

    @param sdb_object Object: SEREBO database object.
    @param length Integer: Length of random string to
    generate.
    @return: Random string
```

```
    '''
    length = int(length)
    return sdb_object.randomString(length)

def stringHash(sdb_object, dstring):
    '''!
    Function to generate hash for a data string.

    @param dstring String: Data string for hash generation.
    @param sdb_object Object: SEREBO database object.
    @return: Hash
    '''
    return sdb_object.hash(str(dstring))

def gmtime(seconds_since_epoch):
    '''!
    Function to generate a UTC date time stamp string in the
    format of <year>:<month>:<day>:<hour>:<minute>:<second>:
    <microsecond> from seconds since epoch. However,
    microseconds cannot be converted; hence, it is given as
    00000.

    @param seconds_since_epoch Float: Seconds since epoch.
    @return: UTC date time stamp string
    '''
    seconds_since_epoch = float(seconds_since_epoch)
    now = time.gmtime(seconds_since_epoch)
    now = [str(now.tm_year), str(now.tm_mon),
           str(now.tm_mday), str(now.tm_hour),
           str(now.tm_min), str(now.tm_sec),
           '00000']
    return ':'.join(now)

def backup(bbpath, backuppath):
    '''!
    Function to backup SEREBO Black Box.

    @param backuppath String: Path for backed-up SEREBO black
    box.
    @param bbpath String: Path to SEREBO black box.
    @return: (absolute bbpath, absolute backuppath)
    '''
    import shutil
    bbpath = absolutePath(bbpath)
```

```
    backuppath = absolutePath(backuppath)
    db = connectDB(bbpath)
    db.cur.execute('begin immediate')
    shutil.copyfile(bbpath, backuppath)
    db.conn.rollback()
    return (str(bbpath), str(backuppath))

def dumpTable(sdb_object, tableName, fieldNames, outputfile):
    '''!
    Function to dump table from SEREBO Black Box to CSV file.

    @param sdb_object Object: SEREBO database object.
    @param tableName String: Name of table.
    @param fieldNames List: List of fields to dump.
    @param outputfile String: Path of file to write data dump.
    @return: (absolute output file path, number of records
    dumped)
    '''
    tableName = str(tableName)
    fieldNames = [str(x) for x in fieldNames]
    fieldNames = ','.join(fieldNames)
    sqlstmt = 'select %s from %s' % (fieldNames, tableName)
    outputfile = absolutePath(outputfile)
    ofile = open(outputfile, 'w')
    count = 0
    for row in sdb_object.cur.execute(sqlstmt):
        row = [str(d) for d in row]
        row = ','.join(row)
        ofile.write(row + '\n')
        count = count + 1
    ofile.close()
  return (outputfile, str(count))
```

FILE NAME: SEREBODB.PY

```
'''!
Secured Recorder Box (SEREBO) Black Box Interface

Date created: 17th May 2018

License: GNU General Public License version 3 for academic
or not-for-profit use only
```

```
SEREBO 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
<http://www.gnu.org/licenses/>.
'''
from datetime import datetime
import hashlib
import random
import os
import secrets
import sqlite3
import string
import time

class SereboDB(object):
    '''!
    Class representing SEREBO database - the recorder black
    box.
    '''
    def __init__(self, dbpath):
        '''!
        Initiation method - connects to SEREBO database. If
        SEREBO database does not exist, this function will
        create the database with the necessary data tables.

        @param bbpath String: Path to SEREBO black box.
        '''
        self.path = dbpath
        self.conn = sqlite3.connect(self.path)
        self.cur = self.conn.cursor()
        self._createTables()

    def dtStamp(self):
        '''!
        Method to generate a UTC date time stamp string in
        the format of <year>:<month>:<day>:<hour>:<minute>:
        <second>:<microsecond>
```

```
        @return: UTC date time stamp string
        '''
        now = datetime.utcnow()
        now = [str(now.year), str(now.month),
               str(now.day), str(now.hour),
               str(now.minute), str(now.second),
               str(now.microsecond)]
        now = ':'.join(now)
        return now

    def randomString(self, length=64):
        '''!
        Method to generate a random string, which can
        contain 80 possible characters - abcdefghijklm
        nopqrstuvwxyzABCDEFGHIJKLMNOPQRSTUVWXYZ0123456
        789~!@#$%^&*()<>=+[]?. Hence, the possible  number
        of strings is 80**length.

        @param length Integer: Length of random string to
        generate. Default = 64.
        @return: Random string
        '''
        choices = string.ascii_letters + \
                  string.digits + \
                  '~!@#$%^&*()<>=+[]?'
        x = random.choices(choices, k=int(length))
        return ''.join(x)

    def hash(self, data):
        '''!
        Method to generate a series of 12 hashes for a given
        data string, in the format of <MD5>:<SHA1>:<SHA224>:
        <SHA3 244>:<SHA256>:<SHA3 256>:<SHA384>:<SHA3 384>:
        <SHA512>:<SHA3 215>:<Blake 2b>:<Blake 2s>.

        @param data String: Data string to generate hash.
        @return: Hash
        '''
        data = str(data)
        data = bytes(data, 'utf-8')
        x = [hashlib.md5(data).hexdigest(),
             hashlib.sha1(data).hexdigest(),
             hashlib.sha224(data).hexdigest(),
             hashlib.sha3_224(data).hexdigest(),
             hashlib.sha256(data).hexdigest(),
             hashlib.sha3_256(data).hexdigest(),
             hashlib.sha384(data).hexdigest(),
             hashlib.sha3_384(data).hexdigest(),
```

```
            hashlib.sha512(data).hexdigest(),
            hashlib.sha3_512(data).hexdigest(),
            hashlib.blake2b(data).hexdigest(),
            hashlib.blake2s(data).hexdigest()]
        return ':'.join(x)

    def _createTables(self):
        '''!
        Private method - used by initialization method to
        generate data tables.
        '''
        now = self.dtStamp()
        # Metadata table
        sql_metadata_create = '''
        create table if not exists metadata (
            key text primary key,
            value text not null);'''
        sql_metadata_insert1 = '''
        insert into metadata (key, value) values
            ('creation_datetimestamp', '%s');''' % (now)
        sql_metadata_insert2 = '''
        insert into metadata (key, value) values
            ('creation_secondstamp', '%s');''' % \
             str(time.time())
        sql_metadata_insert3 = '''
        insert into metadata (key, value) values
            ('blackboxID', '%s');''' % \
            (self.randomString(512))
        sql_notary_create = '''
        create table if not exists notary (
            ID integer primary key autoincrement,
            dtstamp text not null,
            alias text not null,
            owner text not null,
            email text not null,
            notaryDTS text not null,
            notaryAuthorization text not null,
            notaryURL text not null);'''
        # System data table
        sql_systemdata_create = '''
        create table if not exists systemdata (
            ID integer primary key autoincrement,
            dtstamp text not null,
            key text not null,
            value text not null);'''
        # Data log table
        sql_datalog_create = '''
        create table if not exists datalog (
```

```
    ID integer primary key autoincrement,
    dtstamp text not null,
    hash text not null,
    data blob,
    description blob not null);'''
sql_datalog_unique = '''
    create unique index if not exists datalog_unique
    on datalog (dtstamp, hash);'''
# Blockchain table
sql_blockchain_create = '''
create table if not exists blockchain (
    c_ID integer primary key autoincrement,
    c_dtstamp text not null,
    c_randomstring text not null,
    c_hash text not null,
    p_ID integer not null,
    p_dtstamp text not null,
    p_randomstring text not null,
    p_hash text not null,
    data text not null);'''
# Event log table
sql_eventlog_create1 = '''
create table if not exists eventlog (
    ID integer primary key autoincrement,
    dtstamp text not null,
    fID text not null,
    description text not null);'''
sql_eventlog_create2 = '''
create table if not exists eventlog_datamap (
    dtstamp text not null,
    fID text not null,
    key text not null,
    value text not null);'''
# SQL execution
sqlstmt = [sql_metadata_create,
           sql_metadata_insert1,
           sql_metadata_insert2,
           sql_metadata_insert3,
           sql_notary_create,
           sql_systemdata_create,
           sql_datalog_create,
           sql_datalog_unique,
           sql_blockchain_create,
           sql_eventlog_create1,
           sql_eventlog_create2]
for statement in sqlstmt:
    try:
        self.cur.execute(statement)
```

```
            self.conn.commit()
        except sqlite3.IntegrityError:
            pass

def _insertData1A(self, data, description):
    '''!
    Private method - Step 1 of insert data into SEREBO
    black box. Called by insertData method. Step 1 (1)
    gets a UTC date time stamp; (2) formats the
    description by suffixing the description with a 10-
    character random string (80**10 = 1e19 possibility);
    and (3) generates a hash using the UTC date time
    stamp, data and formatted description.

    The difference between _insertData1A() and
    _insertData1B() methods is that _insertData1A()
    method will suffix the description with a 10-
    character random string before using it to generate
    the hash whereas _insertData1B() method uses the
    original (un-suffixed) description in hash
    generation.
    '''
    dtstamp = self.dtStamp()
    DL_data = str(data)
    if description == 'NA' or description == None:
        description = 'NA:' + self.randomString(10)
    else:
        description = str(description) + ':' + \
                      self.randomString(10)
    DL_hash = self.hash(bytes(dtstamp, 'utf-8') + \
                        bytes(DL_data, 'utf-8') + \
                        bytes(description, 'utf-8'))
    return (dtstamp, DL_data, description, DL_hash)

def _insertData1B(self, data, description):
    '''!
    Private method - Step 1 of insert data into SEREBO
    black box. Called by insertData method. Step 1 (1)
    gets a UTC date time stamp; and (2) generates a hash
    using the UTC date time stamp, data and description
    containing the absolute and relative path to the
    file.

    The difference between _insertData1A() and
    _insertData1B() methods is that _insertData1A()
    method will suffix the description with a 10-
    character random string before using it to generate
    the hash whereas _insertData1B() method uses the
```

```
        original (un-suffixed) description in hash
        generation.
        '''
        dtstamp = self.dtStamp()
        DL_data = str(data)
        description = str(description)
        DL_hash = self.hash(bytes(dtstamp, 'utf-8') + \
                            bytes(DL_data, 'utf-8') + \
                            bytes(description, 'utf-8'))
        return (dtstamp, DL_data, description, DL_hash)

    def _insertData2(self, dtstamp, DL_data, description,
                     DL_hash, debug):
        '''!
        Private method - Step 2 of insert data into SEREBO
        black box.
        Called by insertData method. Step 2 inserts the
        results from Step 1 into datalog table.
        '''
        sqlstmt = '''insert into datalog (dtstamp, hash,
            data, description) values (?,?,?,?)'''
        sqldata = (str(dtstamp), str(DL_hash), str(DL_data),
                   str(description))
        self.cur.execute(sqlstmt, sqldata)
        if debug:
            print('Step 1&2: Inserted Data into Data Log
                ...')
            print('Date Time Stamp: %s' % dtstamp)
            print('Inserted Data: %s' % data)
            print('Generated Hash: %s' % DL_hash)

    def _insertData3(self, debug):
        '''!
        Private method - Step 3 of insert data into SEREBO
        black box. Called by insertData method. Step 3 gets
        data (ID, dtstamp, randomstring, and hash) the
        latest pre-existing block in blockchain table, to be
        used as parent in the next block.
        '''
        sqlstmt = '''select max(c_ID) from blockchain'''
        max_cID = [row
            for row in self.cur.execute(sqlstmt)][0][0]
        if max_cID == None:
            p_ID = 0
            p_dtstamp = '0'
            p_randomstring = \
               'GenesisBlock:SEREBO_MauriceHTLing'
            p_hash = 'TheWord:OmAhHum'
```

```
        else:
            sqlstmt = '''select c_ID, c_dtstamp,
                c_randomstring,
                c_hash from blockchain where c_ID = %s''' %\
                str(max_cID)
            data3 = [row for row in
                     self.cur.execute(sqlstmt)]
            p_ID = data3[0][0]
            p_dtstamp = data3[0][1]
            p_randomstring = data3[0][2]
            p_hash = data3[0][3]
        if debug:
            print('Step 3: Getting Latest Block from
                Blockchain ...')
            print('Parent ID: %s' % p_ID)
            print('Parent Date Time Stamp: %s' % p_dtstamp)
            print('Parent Random String: %s' % \
                p_randomstring)
            print('Parent Hash: %s' % p_hash)
        return (p_ID, p_dtstamp, p_randomstring, p_hash)

    def _insertData4(self, p_dtstamp, p_randomstring,
                     p_hash, DL_hash):
        '''!
        Private method - Step 4 of insert data into SEREBO
        black box. Called by insertData method. Step 4 (1)
        generates a hash from the parent date time stamp,
        parent random string, parent hash, and current data
        hash (from Step 1) as current block hash; (2) and
        generates a 32-character random string (80**32 =
        8e60 possibilities) for the current block.
        '''
        BC_rstr = self.randomString(32)
        hashdata = ''.join([str(p_dtstamp),
                            str(p_randomstring),
                            str(p_hash), str(DL_hash)])
        BC_hash = self.hash(bytes(hashdata, 'utf-8'))
        return (BC_rstr, BC_hash)

    def _insertData5(self, dtstamp, BC_rstr, BC_hash, p_ID,
                     p_dtstamp, p_randomstring, p_hash,
                     DL_hash, debug):
        '''!
        Private method - Step 5 of insert data into SEREBO
        black box. Called by insertData method. Step 5
        inserts data of the current block into blockchain
        table.
        '''
```

```
        sqldata = (str(dtstamp), str(BC_rstr), str(BC_hash),
                   str(p_ID), str(p_dtstamp),
                   str(p_randomstring), str(p_hash),
                   str(DL_hash))
        sqlstmt = '''insert into blockchain (c_dtstamp,
            c_randomstring, c_hash, p_ID, p_dtstamp,
            p_randomstring, p_hash, data) values
           (?,?,?,?,?,?,?,?)'''
        self.cur.execute(sqlstmt, sqldata)
        if debug:
            print('Step 5: Insert Data into Blockchain (New
                Block) ...')
            print('Random String: %s' % BC_rstr)
            print('New Block Hash: %s' % BC_hash)
            print('')

    def _insertData6(self, dtstamp, description,
                     DL_hash, p_hash, BC_hash):
        '''!
        Private method - Step 6 of insert data into SEREBO
        black box. Called by insertData method. Step 6
        records the current data insertion event into
        eventlog tables by recording the date time stamp,
        parent block hash, data hash, and current block
        hash.
        '''
        fID = self.randomString(10)
        sqlstmt = '''insert into eventlog (dtstamp, fID,
            description) values (?,?,?)'''
        sqldata = (str(dtstamp), str(fID), str(description))
        self.cur.execute(sqlstmt, sqldata)
        sqlstmt = '''insert into eventlog_datamap (dtstamp,
            fID, key, value) values (?,?,?,?)'''
        sqldata = [(str(dtstamp), str(fID), 'DataHash',
                   str(DL_hash)),
                   (str(dtstamp), str(fID), 'ParentHash',
                    str(p_hash)),
                   (str(dtstamp), str(fID), 'BlockHash',
                    str(BC_hash))]
        self.cur.executemany(sqlstmt, sqldata)

    def insertData(self, data, description='NA',
                   mode='text', debug=False):
        '''!
        Method to insert data into SEREBO database. Data
        will be recorded in datalog table together with the
        hash of the data. The hash of the data will be
        logged into blockchain table. This data insertion
```

```
event will be logged into eventlog tables.

A dictionary of items generated will be returned
with the following keys: (1) DateTimeStamp is the
UTC date time stamp of this event, (2) Data is the
given data string to be inserted, (3)
UserDescription is the user given explanation string
for this event suffixed with a 64-character random
string, (4) DataHash is the hash string of Data, (5)
ParentBlockID is the ID of the parent block in
blockchain, (6) ParentDateTimeStamp is the UTC date
time stamp of the parent block in blockchain (which
is also the parent insertion event), (7)
ParentRandomString is the random string generated
in parent block in blockchain, (8) ParentHash is the
hash of parent block in blockchain, (9)
BlockRandomString is the random string generated for
current insertion event, and (10) BlockHash is the
block hash of current insertion event in blockchain.

@param data String: Data to be inserted.
@param description String: Explanation string for
this entry event. Default = NA.
@param mode String: Type of data to insert.
Allowable modes are 'text' (description text is
suffixed with a 10-character random string), 'ftext'
(description text is not suffixed with a 10-
character random string) and 'file' (for file hash
logging). Default = 'text'.
@param debug Boolean: Flag to print out debugging
statements.
@return: Dictionary of data generated from this
event.
'''
# Step 1: Preparing data
if mode.lower() == 'text':
    (dtstamp, DL_data, description, DL_hash) = \
        self._insertData1A(data, description)
elif mode.lower() == 'file':
    (dtstamp, DL_data, description, DL_hash) = \
        self._insertData1B(data, description)
elif mode.lower() == 'ftext':
    (dtstamp, DL_data, description, DL_hash) = \
        self._insertData1B(data, description)
# Step 2: Insert data into datalog
self._insertData2(dtstamp, DL_data, description,
                  DL_hash, debug)
# Step 3: Get latest block in blockchain
```

```
        (p_ID, p_dtstamp, p_randomstring, p_hash) = \
            self._insertData3(debug)
        # Step 4: Prepare data for blockchain insertion
        (BC_rstr, BC_hash) = self._insertData4(p_dtstamp,
                                               p_randomstring,
                                                 p_hash,
                                                 DL_hash)
        # Step 5: Insert data into blockchain
        self._insertData5(dtstamp, BC_rstr, BC_hash, p_ID,
                          p_dtstamp, p_randomstring, p_hash,
                          DL_hash, debug)
        # Step 6: Insert event into eventlog
        self._insertData6(dtstamp, description,
                          DL_hash, p_hash, BC_hash)
        # Step 7: Commit
        self.conn.commit()
        # Step 8: Return data
        return {'DateTimeStamp': dtstamp,
                'Data': data,
                'UserDescription': description,
                'DataHash': DL_hash,
                'ParentBlockID': p_ID,
                'ParentDateTimeStamp': p_dtstamp,
                'ParentRandomString': p_randomstring,
                'ParentHash': p_hash,
                'BlockRandomString': BC_rstr,
                'BlockHash': BC_hash}
```

CODE FILES FOR SEREBO NOTARY

FILE NAME: SERVICES.PY

```
'''!
Secured Recorder Box (SEREBO) Notary Services

Date created: 19th May 2018

License: GNU General Public License version 3 for academic or
not-for-profit use only


SEREBO 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
<http://www.gnu.org/licenses/>.
'''

from datetime import datetime
import hashlib as h
import random
import string

from gluon.tools import Service

service = Service()

def call():
    '''!
    Function to enable web services in Web2Py.
    '''
    session.forget()
    return service()

@service.xmlrpc
def now():
    '''!
    Function to generate a UTC date time stamp string in the
    Format of <year>:<month>:<day>:<hour>:<minute>:<second>:
    <microsecond>

    @return: UTC date time stamp string
    '''
    dt = datetime.utcnow()
    x = [str(dt.year), str(dt.month),
         str(dt.day), str(dt.hour),
         str(dt.minute), str(dt.second),
         str(dt.microsecond)]
    return ':'.join(x)

@service.xmlrpc
```

```
def randomString(length=16):
    '''!
    Function to generate a random string, which can contain
    80 possible characters - abcdefghijklmnopqrstuvwxyz
    ABCDEFGHIJKLMNOPQRSTUVWXYZ0123456789~!@#$%^&*()<>=+[]?.
    Hence, the possible number of strings is 80**length.

    @param length Integer: Length of random string to
    generate. Default = 16.
    @return: Random string
    '''
    choices = string.ascii_letters + \
              string.digits + \
              '~!@#$%^&*()<>=+[]?'
    x = [random.choice(choices)
         for i in range(int(length))]
    return ''.join(x)

@service.xmlrpc
def register_blackbox(blackboxID, owner, email,
              architecture, machine, node,
              platform, processor):
    '''!
    Function to register SEREBO Black Box with SEREBO Notary.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param owner String: Owner's or administrator's name.
    @param email String: Owner's or administrator's email.
    @param architecture String: Architecture of machine - from
    platform library in Python Standard Library.
    @param machine String: Machine description - from platform
    library in Python Standard Library.
    @param node String: Machine's node description - from
    platform library in Python Standard Library.
    @param platform String: Platform description - from
    platform library in Python Standard Library.
    @param processor String: Machine's processor description
    - from platform library in Python Standard Library.
    @returns: (Notary authorization code, Date time stamp from
    SEREBO Notary)
    '''
    dtstamp = now()
    notaryAuthorization = str(randomString(256))
```

```
notabase.registered_blackbox.insert(datetimestamp=dtstamp,
                                    blackboxID=str(blackboxID),
                                             owner=str(owner),
                                             email=str(email),
                                architecture=str(architecture),
                                          machine=str(machine),
                                                node=str(node),
                                       platform=str(platform),
                                     processor=str(processor),
                         notaryAuthorization=notaryAuthorization)
    eventText = ['SEREBO Black Box Registration',
                 'Date Time Stamp: %s' % dtstamp,
                 'Black Box ID: %s' % blackboxID,
                 'Owner: %s' % owner,
                 'Email: %s' % email,
                 'Notary Authorization: %s' % \
                     notaryAuthorization]
    eventText = ' | '.join(eventText)
    notabase.eventlog.insert(datetimestamp=dtstamp,
                             event=eventText)
    return (notaryAuthorization, dtstamp)

@service.xmlrpc
def hash(dstring):
    '''!
    Function to generate a series of 6 hashes for a given data
    string, in the format of <MD5>:<SHA1>:<SHA224>:<SHA256>:
    <SHA384>:<SHA512>.

    @param dstring String: String to generate hash.
    @return: Hash
    '''
    dstring = str(dstring)
    x = [h.md5(dstring).hexdigest(),
         h.sha1(dstring).hexdigest(),
         h.sha224(dstring).hexdigest(),
         h.sha256(dstring).hexdigest(),
         h.sha384(dstring).hexdigest(),
         h.sha512(dstring).hexdigest()]
    return ':'.join(x)

@service.xmlrpc
def checkBlackBoxRegistration(blackboxID,
```

```
                                notaryAuthorization):
    '''!
    Function to check for SEREBO Black Box registration.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param notaryAuthorization String: Notary authorization
    code of SEREBO black box (generated during black box
    registration - found in metadata table in SEREBO black box
    database.
    @return: True if SEREBO Black Box is registered; False if
    SEREBO Black Box is not registered
    '''
    if notabase(notabase.registered_blackbox.blackboxID == \
        blackboxID) \
    (notabase.registered_blackbox.notaryAuthorization == \
        notaryAuthorization).count():
        return True
    else:
        return False

@service.xmlrpc
def notarizeSereboBB(blackboxID, notaryAuthorization,
                     dtstampBB, codeBB):
    '''!
    Function to notarize SEREBO Black Box with SEREBO Notary.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param notaryAuthorization String: Notary authorization
    code of SEREBO black box (generated during black box
    registration - found in metadata table in SEREBO black box
    database.
    @param dtstampBB String: Date time stamp from SEREBO black
    box.
    @param codeBB String: Notarization code from SEREBO black
    box.
    @returns: (Date time stamp from SEREBO Notary,
    Notarization code from SEREBO Notary, Cross-Signing code
    from SEREBO Notary)
    '''
    blackboxID = str(blackboxID)
    notaryAuthorization = str(notaryAuthorization)
    dtstampBB = str(dtstampBB)
```

```
    codeBB = str(codeBB)
    dtstampNS = now()
    codeNS = str(randomString(32))
    codeCommon = hash(codeBB + codeNS)
    notabase.notarize_blackbox.insert(blackboxID=blackboxID,

notaryAuthorization=notaryAuthorization,
                                      dtstampBB=dtstampBB,
                                      dtstampNS=dtstampNS,
                                      codeBB=codeBB,
                                      codeNS=codeNS,
                                      codeCommon=codeCommon)
    eventText = ['SEREBO Black Box Notarization',
                 'Success',
                 'Date Time Stamp: %s' % dtstampNS,
                 'Black Box ID: %s' % blackboxID,
                 'Notary Authorization: %s' % \
                     notaryAuthorization,
                 'Black Box Code: %s' % codeBB,
                 'Notary Code: %s' % codeNS,
                 'Cross-Signing Code: %s' % codeCommon]
    eventText = ' | '.join(eventText)
    notabase.eventlog.insert(datetimestamp=dtstampNS,
                             event=eventText)
    return (dtstampNS, codeNS, codeCommon)

@service.xmlrpc
def checkNotarizeSereboBB(blackboxID, notaryAuthorization,
                          BBCode, NCode, CommonCode):
    '''!
    Function to check for SEREBO Black Box notarization by
    SEREBO Notary.

    @param blackboxID String: ID of SEREBO black box - found
    in metadata table in SEREBO black box database.
    @param notaryAuthorization String: Notary authorization
    code of SEREBO black box (generated during black box
    registration - found in metadata table in SEREBO black box
    database.
    @param BBCode String: Notarization code from SEREBO Black
    Box.
    @param NCode String: Notarization code from SEREBO Notary.
    @param CommonCode String: Cross-Signing code from SEREBO
    Notary.
```

```
    @return: True if notarization is found; False if
    notarization is not found.
    '''
    if notabase(notabase.notarize_blackbox.blackboxID == \
         blackboxID) \
        (notabase.notarize_blackbox.notaryAuthorization == \
         notaryAuthorization)\
        (notabase.notarize_blackbox.codeBB == BBCode)\
        (notabase.notarize_blackbox.codeNS == NCode)\
        (notabase.notarize_blackbox.codeCommon == \
         CommonCode).count():
        return True
    else:
        return False
```

FILE NAME: SEREBO_NOTABASE.PY

```
'''!
Secured Recorder Box (SEREBO) Notary Database

Date created: 19th May 2018

License: GNU General Public License version 3 for academic or
not-for-profit use only


SEREBO 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
<http://www.gnu.org/licenses/>.
'''

notabase = SQLDB('sqlite://serebo_notabase.sqlite')

'''
```

```
Table registered_blackbox is to store registration data
of SEREBO Black Box.
'''
notabase.define_table('registered_blackbox',
    SQLField('datetimestamp', 'text'),
    SQLField('blackboxID', 'text', unique=True),
    SQLField('owner', 'text'),
    SQLField('email', 'text'),
    SQLField('architecture', 'text'),
    SQLField('machine', 'text'),
    SQLField('node', 'text'),
    SQLField('platform', 'text'),
    SQLField('processor', 'text'),
    SQLField('notaryAuthorization', 'text'))

'''
Table notarize_blackbox is to store notarization data
when SEREBO Black Box requests for SEREBO Notary's
notarization.
'''
notabase.define_table('notarize_blackbox',
    SQLField('blackboxID', 'text'),
    SQLField('notaryAuthorization', 'text'),
    SQLField('dtstampBB', 'text'),
    SQLField('dtstampNS', 'text'),
    SQLField('codeBB', 'text'),
    SQLField('codeNS', 'text'),
    SQLField('codeCommon', 'text'))

'''
Table eventlog is to keep a record of notable events
in SEREBO Notary.
'''
notabase.define_table('eventlog',
    SQLField('datetimestamp', 'text'),
    SQLField('event', 'text'))
```

REFERENCES

Bik, E.M., Fang, F.C., Kullas, A.L., Davis, R.J., and Casadevall, A. (2018). Analysis and Correction of Inappropriate Image Duplication: the Molecular and Cellular Biology Experience. Molecular Biology of the Cell *38*, e00309-18.

Boneh, D., and Boyen, X. (2006). On the impossibility of efficiently combining collision resistant hash functions. (Springer), pp. 570–583.

Broder, A., and Mitzenmacher, M. (2001). Using multiple hash functions to improve IP lookups. (IEEE), pp. 1454–1463.

Dolinski, K., and Troyanskaya, O.G. (2015). Implications of Big Data for cell biology. Molecular Biology of the Cell *26*, 2575–2578.

Ling, M.H. (2013). NotaLogger: Notarization Code Generator and Logging Service. The Python Papers *9*, 2.

Ling, M.H. (2018a). SEcured REcorder BOx (SEREBO) Based on Blockchain Technology for Immutable Data Management and Notarization. MOJ Proteomics & Bioinformatics *7*, 169–174.

Ling, M.H. (2018b). A Cryptography Method Inspired by Jigsaw Puzzles. In Current STEM, Volume 1, (Nova Science Publishers, Inc.), pp. 129–142.

Pierce, A. (2010). The Evolution of the Airplane Black Box. Tech Directions *70*, 12.

Pierro, M.D. (2008). Web2Py: Enterprise Web Framework (New Jersey, USA: John Wiley & Sons, Inc.).

Rasjid, Z.E., Soewito, B., Witjaksono, G., and Abdurachman, E. (2017). A review of collisions in cryptographic hash function used in digital forensic tools. Procedia Computer Science *116*, 381–392.

Suwinski, P., Ong, C., Ling, M.H.T., Poh, Y.M., Khan, A.M., and Ong, H.S. (2019). Advancing Personalized Medicine Through the Application of Whole Exome Sequencing and Big Data Analytics. Frontiers in Genetics *10*, 49.

Zheng, Z., Xie, S., Dai, H., Chen, X., and Wang, H. (2017). An Overview of Blockchain Technology: Architecture, Consensus, and Future Trends. In 2017 IEEE International Congress on Big Data (BigData Congress), pp. 557–564.