CSE 469 Group Project Report

Title of the Project	Blockchain Chain of Custody		
Group Number	3		
Group Members	Souradip Nath	1225226964	snath8@asu.edu
	Fatimah Alyousef	1217797490	fhalyous@asu.edu
	Yazan Alahmadi	1217804549	yalahma2@asu.edu

I. INTRODUCTION

The aim of this group project is to implement a digital equivalent of the blockchain chain of custody (CoC). It is a form to store the history of evidence; including when it is added or removed, checked in or checked out, and the information of the owner of the evidence, in addition to different items of the evidence. In order to build the digital version of CoC, we have to implement seven functionalities.

The first function is "**add**", which takes a unique case id and one or more unique item IDs as inputs. It adds the evidence to the blockchain and associates it with the given case id, and it creates an entry block for each item. When new evidence is added, its state is marked as CHECKEDIN. The second function is "**checkout**", which takes the item ID as input and creates a new state entry (checkout) in the blockchain. The third function is "**checkin**", which takes the item ID as input and creates a new state entry (checkin) in the blockchain. The second and third functions only work when an item is already added to the blockchain, otherwise, it will exit the program.

The fourth function is "**log**", which prints the blockchain entries from oldest to newest evidence items added. It takes optional arguments; r for displaying the blockchain entries in reverse order, and num_entries for displaying the number of entries. If a case ID is given, the blocks that are identified with it are printed. The log function can also take item ID as input, then it would return the blocks with that ID. The fifth function is "**remove**", which prevents any further action from being done regarding the evidence. To remove an item, its state must be marked with CHECHEDIN. The function takes the item ID, and one of three reasons to indicate why the item would be removed; DISPOSED, DESTROYED, or RELEASED - owner information must be given if the reason is RELEASED.

The sixth function is "**init**", which checks the initial block of the blockchain. If the program doesn't find any existing blocks when it first starts, it should create a block with the startup information; *Previous Hash: null, Timestamp: Current time, Case ID: null, Evidence Item ID: null, State: "INITIAL", Data Length: 14 bytes,* and *Data: "Initial block"*. Finally, the seventh function is **"verify"**; it validates the blockchain when it has no error, and if an error is found, it illustrates what is missing. For example, it returns an error when the parent block of the case is not found, or multiple blocks are found with the same parent block.

II. DESIGN AND IMPLEMENTATION DETAILS

This section explains the details of the design decisions made for the implementation of the digital Chain of Custody using Blockchain technology. The primary aspects of the design are the structure of the blockchain, an efficient storing method for easy parsing, and some fundamental operations on the blockchain such as adding a new case/evidence, checking in/out evidence, removing a piece of evidence, etc, and input handling.

(A) STRUCTURE OF THE BLOCKCHAIN: In this work, the blockchain has been designed following the Object Oriented Programming(OOP) paradigm, as a class named "Blockchain", which represents an abstraction to the structure of each block in the blockchain chain of custody. The class Blockchain has the following member variables: *prev_hash, timestamp, case_id, item_id, state, data_length, data,* and *states*. Table 1 describes each member variable

Variable Name	Datatype	Description
prev_hash	String	SHA-512 hash of the previous block of the current block
timestamp	Double	The block creation time in ISO 8601 format
case_id	Integer	The unique UUID assigned to a case
item_id	Integer	The unique IDs assigned to each evidence of a case
state	String	State of the block, might be one of the following: INITIAL, CHECKEDIN, CHECKEDOUT, DISPOSED, DESTROYED, or RELEASED
data_length	Integer	The byte count of the data field
data	String	Free form text with a length stored in the data_length field
states	Python Dictionary	A dictionary mapping each possible state to its byte representation

Table 1: Description of the Blockchain class structure

in detail. The class also contains a constructor function *__init__* and a member method called *get_binary_data(.)* which packs the member variables of an object of type Blockchain in the binary format.

(B) STORING THE BLOCKCHAIN: In this work, the blockchain has been stored in binary format, in a file called *blockchain.bin*. In the dev environment, this file resides in the same directory as the source file, and it gets created when the code is executed for the first time. In the test environment, the path to this file is obtained through the environment variable

BCHOC_FILE_PATH. Hence, to accommodate both these options, the program first checks the value of the BCHOC_FILE_PATH variable; if it gets None, then it looks for the file in the same level directory and creates the file if it does not exist already.

As discussed in the previous subsection, each block in the blockchain is technically an object of type Blockchain. Therefore, to store the blocks in the binary format, every new block is packed using the *struct* library of Python following the following format "32s d 16s I 12s I". This explicitly specifies how many bytes and what data type must be used to store the individual variable in the object, which forms the header of each block. In this work, for most of the blocks, the *data* field is left empty, and the focus has been primarily on storing the header information.

(C) PARSING THE BLOCKCHAIN: As discussed in the previous subsection, the entire blockchain chain of custody is stored in a binary file. Hence, for performing any update/add/delete operation on the blockchain, the first step is to parse the binary file to get the current status of the blockchain. Every new block is then appended to the end of the chain, as the next block of the current last block. Algorithm 1 describes the way the blockchain has been parsed in this work.

Algorithm 1: Pseudocode for Blockchain Parsing

```
start_index := 0
last_index := length(blockchain)

index := start_index
while index < last_index do
        block_header := blockchain[start_index, start_index+76]
        Unpack block_header to prev_hash, timestamp, case_id, item_id, state,
        data_len

        block_data := blockchain[start_index+76, start_index+76+data_len]
        Unpack block_data

        block := block_header + block_data
        block_size := length(block)

        index := index + block_size
```

(D) OPERATIONS ON THE BLOCKCHAIN: The blockchain structure represents a digital equivalent of the Chain of Custody form. Hence, the operations that are performed on a CoC

form, can also be performed on the blockchain. The specific operations defined in this work are, *init, add, checkin, checkout, remove, log,* and *verify*. All these methods take the *blockchain.bin* file as input along with some method-specific parameters, parse the file as discussed in the previous section, perform read/write operations on the binary file, and return a status code following the convention of zero indicating success and non-zero indicating failure (error).

(E) INPUT HANDLING: The program accepts input in the form of command line arguments, i.e., either one of the operations discussed above has to be passed as a command while running the program along with its required parameters. The command line argument gets processed through a function called *parse(.),* which parses it, identifies the specific command, and makes a call to its corresponding defined method. The function also identifies the parameters that are passed

```
parse("add -c d2c6b728-68ac-11ed-9022-0242ac120002 -i 123456789")
add(case_id="d2c6b728-68ac-11ed-9022-0242ac120002", item_id=123456789)
```

along with the command. The example above shows the command line argument and the corresponding function call by the *parse(.)* function.

The codebase also contains a Makefile which processes the source code and creates an executable called "bchoc". It creates the executable in two steps, first, it creates a copy of the source code and names it 'bchoc', and then it informs the operating system that the mode of this file is executable.

The source code is written in Python 3.10 in Ubuntu 22.04.1 LTS. The reasons for choosing Python for the implementation are: (1) Python supports the Object-Oriented Programming (OOP) paradigm, (2) Python is platform-agnostic, hence it supports cross-OS development, (3) The standard library of Python grants a lot of pre-built features that allow programmers to work with date and time formats, manage operating systems, apply hash functions, etc., with less effort, (4) Python has a high speed of development compared to other programming languages, and (5) Python has a smooth learning curve due to its clear and concise syntax.

III. CHALLENGES

There are quite a few challenges that we faced while working on the project. Starting off the semester we were a team of 4 people. However, quite a few weeks into the semester, two of the group members withdrew from the course, and only one member was added to the team in return. That set us back quite some time. Nevertheless, we managed to continue working as a group to achieve the main goal of completing the project ahead of its due date in order to stay on track and revisit any potential issues that might come up.

Another main challenge that we faced while implementing the functions is to account for every possible case. Especially since we have to fulfill all the requirements, and some command lines cannot pass only after running another command. For instance, the *remove* function cannot be performed unless *add* function is called for the given item ID.

When implementing the *remove* function, it was challenging at first to indicate whether the given item could be removed or not, so we started by checking the reason for removing the item. If the reason was one of DISPOSED, DESTROYED, or RELEASED, then the item could pass the first check and proceed to be parsed. Moreover, when storing the block information, we had to create a few changes in the format of storing the owner information when the given reason is RELEASED. Thus, we solved it by checking if the owner's information is given or not and storing the data length of the owner.

IV. DISCUSSION

When doing a digital forensics investigation, one of the main go-to methods for investigators is developing a chain of custody through a blockchain. Blockchains offer so many beneficial attributes for the conduction of a lawful forensic investigation. One of the main advantages that a blockchain provides to digital investigations is the integrity of the data. A blockchain offers investigators the privilege of monitoring and maintaining each data entry in the blockchain, as well as prohibiting the modification of data. Also, due to the nature of blockchains, if data is entered into the blockchain, it can not be deleted afterward, this also plays a huge role in terms of maintaining the integrity of the investigation, as well as enhancing the authenticity and reliability of evidence in digital forensics investigations. Furthermore, blockchains can make use of encryption methods to keep confidential information from unauthorized access. This can be implemented when each block entry is

signed and encrypted via the usage of various mechanisms, such as the private and public key mechanisms.

Although a chain of custody blockchain is still not completely safe from cybercrime manipulation of data, at the moment, it still seems to be the most reliable option to proceed with digital forensics investigations. There are multiple methods that blockchain developers can use to enhance the reliability of the blockchain for it to become more effective, one of which is the usage of strong encryption methods. Instead of using one of the previously popular hashing methods such as SHA, digital forensics investigators can use further secure encryption methods such as a fuzzy-hashed blockchain to further secure the confidentiality of the blockchain, which has proven to be very effective for implementation in the blockchain chain of custody (Elgohary et al., 2022).

APPENDIX

Execution Screenshots. The following are some of the screenshots of the execution of the "bchoc" commands in the development environment, Ubuntu 22.04.1 LTS.

```
souradip@souradip-Lenovo-IdeaPad-S145-15IIL:~/Desktop/Forensics Projects/CSE469-GroupProject$ ./bchoc init
Blockchain file not found. Created INITIAL block.
souradip@souradip-Lenovo-IdeaPad-S145-15IIL:~/Desktop/Forensics Projects/CSE469-GroupProject$ ./bchoc add -c 65cc391d-6568-4dcc-a3f2-86a2f04140f3 -i 123456789
BlockChain file found with INITIAL block.
Case: 65cc391d-6568-4dcc-a3f2-86a2f04140f3
Added item: 123456789
        Status: CHECKEDIN
        Time of action: 2022-11-30T13:49:24.966313Z
souradip@souradip-Lenovo-IdeaPad-S145-15IIL:~/Desktop/Forensics Projects/CSE469-GroupProject$ ./bchoc checkout -i 123456789
Case: f34041f0-a286-f2a3-cc4d-68651d39cc65
Checked out item: 123456789
        Status: CHECKEDOUT
        Time of action: 2022-11-30T13:49:49.612734Z
souradip@souradip-Lenovo-IdeaPad-S145-15IIL:~/Desktop/Forensics Projects/CSE469-GroupProject$ ./bchoc checkin -i 123456789
Case: f34041f0-a286-f2a3-cc4d-68651d39cc65
Checked in item: 123456789
        Status: CHECKEDIN
        Time of action: 2022-11-30T13:50:00.473146Z
```

Screenshot 1: Execution of *init, add, checkout,* and *checkin* commands

```
souradip@souradip-Lenovo-IdeaPad-S145-15IIL:~/Desktop/Forensics Projects/CSE469-GroupProject$ ./bchoc remove -i 123456789 -y RELEASED -o "Souradip Nath"
Case: 65cc391d-6568-4dcc-a3f2-86a2f04140f3
Removed item: 123456789
        Status: RELEASED
        Owner info: Souradip Nath
        Time of action: 2022-11-30T14:03:16.572635Z
```

Screenshot 2: Execution of the *remove* command with RELEASED flag

```
souradip@souradip-Lenovo-IdeaPad-S145-15IIL:~/Desktop/Forensics Projects/CSE469-GroupProject$ ./bchoc log
Case: 00000000-0000-0000-0000-000000000000
Item: 0
Action: INITIAL
Time: 2022-11-30T13:47:51.643883Z

Case: 65cc391d-6568-4dcc-a3f2-86a2f04140f3
Item: 123456789
Action: CHECKEDIN
Time: 2022-11-30T13:49:24.966313Z

Case: 65cc391d-6568-4dcc-a3f2-86a2f04140f3
Item: 123456789
Action: CHECKEDOUT
Time: 2022-11-30T13:49:49.612734Z

Case: 65cc391d-6568-4dcc-a3f2-86a2f04140f3
Item: 123456789
Action: CHECKEDIN
Time: 2022-11-30T13:50:00.473146Z

Case: 65cc391d-6568-4dcc-a3f2-86a2f04140f3
Item: 123456789
Action: RELEASED
Time: 2022-11-30T13:52:48.987230Z
```

Screenshot 3: Execution of the *log* command

Execution Results. Our implementation of the blockchain chain of custody program was tested by the *gradescope autograder* against 58 test cases, testing the seven different functions under different scenarios. The final version of the program successfully passed all 58 test cases securing a full score of 85.0.

Codebase. The entire project codebase is publicly accessible on GitHub and can be accessed through the following link: https://github.com/chloechngo/CSE469-GroupProject. The repository also contains a README file that briefly explains how the code works.

Reference. Elgohary, H. M., Darwish, S. M., & Elkaffas, S. M. (2022). Improving Uncertainty in Chain of Custody for Image Forensics Investigation Applications. IEEE Access, 10, 14669–14679. https://doi.org/10.1109/access.2022.3147809