Group Project:
Blockchain Chain of Custody

- Due Date: Apr. 22, 2024, 11:59 pm
- Done By: Groups
- Checkpoint: Mar. 13 w/ progress reports
- Options: 1-Programming Language-based, 2- Open-source framework-based
- Submission: Gradescope (Source code, Demo video, Report)

The Chain of Custody form is a critical element of a forensic investigation because examiners use it to record the history of the evidence from the time it is found until the case is closed or goes to court. By keeping this record, examiners can show that the integrity of the evidence has been preserved and is not open to compromise. In the unfortunate event that evidence *does* become contaminated, the chain of custody will clearly identify the responsible individual.

A Chain of Custody form keeps track of three pieces of important information (in addition to all the details that uniquely identify the specific piece of evidence):

1. **Where** the evidence was stored?
2. **Who** had access to the evidence and **when?**
3. **What** actions were taken to the evidence?

As an example, please refer to this generic chain of custody from NIST:

- Regular URL: https://www.nist.gov/document/sample-chain-custody-formdocx
- Google's cached version: https://webcache.googleusercontent.com/search?q=cache:EDTx4jL_PqQJ:https://www.nist.gov/document/sample-chain-custody-formdocx+&cd=1&hl=en&ct=clnk&gl=us

For this project, your group will write a program that will be a digital equivalent of a chain of custody form. Each entry in the form will be stored in a blockchain of your own creation.

Blockchain technology has been touted as a solution that will improve every aspect of digital communication. However, real-world results have been practically non-existent. Consider this flow chart that explores some use cases where a blockchain may be a promising idea:

Common Requirements

Your blockchain chain of custody program must implement the following commands:

```
bchoc add -c case_id -i item_id [-i item_id ...] -c creator -p
password(creator’s)
bchoc checkout -i item_id -p password
bchoc checkin -i item_id -p password
bchoc show cases -p password
bchoc show items -c case_id -p password
bchoc show history [-c case_id] [-i item_id] [-n num_entries] [-r] -p
password
bchoc remove -i item_id -y reason [-o owner] -p password(creator’s)
bchoc init
bchoc verify
```

[ ] >> optional arguments

Where the parameters must conform to the following specifications:

add

Add a new evidence item to the blockchain and associate it with the given case identifier. For users’ convenience, more than one item_id may be given at a time, which will create a blockchain entry for each item without the need to enter the case_id multiple times. The state of a newly added item is CHECKEDIN. The given evidence ID must be unique (i.e., not already used in the blockchain) to be accepted. The password must be that of the creator.

checkout

Add a new checkout entry to the chain of custody for the given evidence item. Checkout actions may only be performed on evidence items that have already been added to the blockchain. The password must be that of anyone from the owners.

checkin

Add a new checkin entry to the chain of custody for the given evidence item. Checkin actions may only be performed on evidence items that have already been added to the blockchain. The password must be that of anyone from the owners.

show cases

Display a list of all the cases that have been added to the blockchain.

show items

Display all the items corresponding to the case number in the request.

show history

Display the blockchain entries for the requested item giving the oldest first.

-r, –reverse

Reverses the order of the block entries to show the most recent entries first.

remove

Prevents any further action from being taken on the evidence item specified. The specified item must have a state of CHECKEDIN for the action to succeed. The password must be that of the creator.

init

Sanity check. Only starts up and checks for the initial block.

verify

Parse the blockchain and validate all entries.

-c case_id

Specifies the case identifier that the evidence is associated with. Must be a valid UUID. When used with log only blocks with the given case_id are returned.

-i item_id

Specifies the evidence item's identifier. When used with log only blocks with the given item_id are returned. The item ID must be unique within the blockchain. This means you cannot re-add an evidence item once the remove action has been performed on it.

-p password

Has to be one of the creators or owners. The passwords will be provided to you.

-n num_entries

When used with history, shows num_entries number of block entries.

-y reason, --why reason

Reason for the removal of the evidence item. Must be one of: DISPOSED, DESTROYED, or RELEASED. If the reason given is RELEASED, -o must also be given.

-o owner

Information about the lawful owner to whom the evidence was released. At this time, text is free-form and does not have any requirements.

Data structure

Every block in the blockchain will have the same structure:

Offset	Length (bytes)	Field Name - Description
0x00, 0$_{10}$	32* (256 bits)	Previous Hash - SHA-256 hash of this block's parent
0x20, 32$_{10}$	8 (64 bits)	Timestamp - Regular Unix timestamp. Must be printed in ISO 8601 format anytime displayed to the user. Stored as an 8-byte float (double).
0x28, 40$_{10}$	32 (256 bits)	Case ID - UUID (Encrypted using AES ECB, stored as hex)
0x48, 72$_{10}$	32 (256 bits)	Evidence Item ID - 4-byte integer. (Encrypted using AES ECB, stored as hex)
0x68, 104$_{10}$	12** (96 bits)	State - Must be one of: INITIAL (for the initial block ONLY), CHECKEDIN, CHECKEDOUT, DISPOSED, DESTROYED, or RELEASED.
0x74, 116	12 (96 bits bits)	Creator - Free form text with max 12 characters
0x80, 128	12 (96 bits bits)	Owner - Free form text with max 16 characters (Must be one of Police, Lawyer, Analyst, Executive)
0x8c, 140$_{10}$	4 (32 bits)	Data Length (byte count) - 4-byte integer.
0x90, 144$_{10}$	0 to (2^32)	Data - Free form text with byte length specified in Data Length.

Note

*The length of the Previous Hash field is only 32 bytes, it has 4 bytes alignment.

**The State field is padded with an extra byte (for a total of 12 bytes or 96 bits), making the Data Length field's offset 0x70, or byte 112 in decimal.

If you use Python to do the project, I recommend you use the struct format string `"32s d 32s 32s 12s 12s 12s I"` to pack and unpack the first 6 fields of the block, which will handle the byte alignment issue for you.

Finally, you don't have to add extra paddings at the end of the block to have the exact block size, unless it is the string fields.

The Cae ID has to be a UUID and the Item ID has to be a 4-byte integer. Both these values have to be encrypted using AES ECB mode of encryption before storing in the data file. (The key will be given to you)
The actual values should only be shown if a valid password is provided with the show command. Otherwise, the encrypted values should be shown.

The location of the blockchain file doesn't matter while you are implementing and locally testing your program. However, when we grade your assignment, we will set the environment variable BCHOC_FILE_PATH to the path to the file your program should use.

Important

Make sure that your program checks the BCHOC_FILE_PATH environment variable first before using any other path! Otherwise, your program will fail the grading test cases.

When the program starts it should check if there are any existing blocks and create a block with the following information if it doesn't find any:

- Previous Hash: None, null, etc.
- Timestamp: Current time
- Case ID: None, null, etc.
- Evidence Item ID: None, null, etc.
- State: "INITIAL"
- Creatore: None, null, etc.
- Owner: None, null, etc.
- Data Length: 14 bytes
- Data: The string: "Initial block"

All block data must be stored in a binary format. Plain text, JSON, CSV, and other similar formats are invalid for this assignment.

All timestamps must be stored in UTC and account for the difference between local time and UTC.

> **Note**
>
> For a Python library that helps deal with timestamps, check out Maya.

Important

There are MANY conditions that could put your program into an error state. Whenever this occurs, your program should exit with a non-zero exit status.

Using this convention will have a few benefits. First, it will force you to do the work of thinking through the various execution paths that could lead to an error state, which is an excellent exercise that will develop your software engineering skills. Second, it gives you the freedom of coming up with your own meaningful messages to the user, rather than me coming up with them for you. Third, it makes it simpler for us to grade your program because all we have to check in these cases is the exit code of your program to verify it is functioning correctly, while also decreasing potential string-matching errors.

As the link above on exit status discusses, "The specific set of codes returned is unique to the program that sets it." This means you get to define your own exit codes and what they mean. As long as you use the convention of zero indicating success and non-zero indicating failure (error), you can choose to use whatever code values you like.

Example

Below are some examples of input/output for your program. Lines beginning with $ are the input and everything else is the output from the given command.

More to be added

Initializing the blockchain:

```
$ bchoc init
Blockchain file not found. Created INITIAL block.
```

Checking the initialization:

```
$ bchoc init
Blockchain file found with INITIAL block.
```

Warning

Normally, you should be incredibly careful about accepting user input that you later use and print to the screen. But for the purposes of this project, you don't need to worry about sanitizing input.

Verifying the blockchain:

```
$ bchoc verify
Transactions in blockchain: 6
State of blockchain: CLEAN
```

Verifying the blockchain when it has errors:

```
$ bchoc verify
Transactions in blockchain: 6
State of blockchain: ERROR
Bad block: ca53b1f604b633a6bc3cf75325932596efc4717fca53b1f604b633a6bc3cf732
Parent block: NOT FOUND
```

Or:

```
$ bchoc verify
Transactions in blockchain: 6
State of blockchain: ERROR
Bad block: 9afcca9016f56e3d12f66958436f92f6a61f8465ca53b1f04ba633a1bc3cf75f
Parent block:
99bcaaf29b1ff8dac2c529a8503d92e43921c335da53bb1f604b63a6bc3cf25e
Two blocks were found with the same parent.
```

Or:

```
$ bchoc verify
```

```
Transactions in blockchain: 6
State of blockchain: ERROR
Bad block: 99bcaaf29b1ff8dac2c529a8503d92e43921c335c853b1f604b6c33a6c3c475d
Block contents do not match block checksum.
```

Or:

```
$ bchoc verify
Transactions in blockchain: 6
State of blockchain: ERROR
Bad block: e3f2b0427b57241225ba1ffc2b67fecd64d07613fa3b1f604b633a6bac1cf75c
Item checked out or checked in after removal from chain.
```

Note

For testing purposes, you can assume that a blockchain will only have one error in it. If this weren't the case, it would matter which direction you traverse the chain while validating, and I don't want you to have to worry about that.

In cases of errors (invalid operations), exit with error code 1.

```
###
    #
    #  Test cases:( Final tests may have some changes)
    #
    # . init with no pre-existing file #1
    # . init with pre-existing valid file (single block) #2
    # . init with pre-existing valid file (multiple blocks) #3
    # F init with pre-existing invalid file (single block) #4
    # F init with pre-existing invalid file (multiple blocks) #5
    # F init with additional parameters #6
    #
    # . add before init - should create initial block #7
    # . add with one case_id and one item_id - status of the added item is CHECKEDIN #8
    # . add with one case_id and two item_id values #9
    # F add with no case_id given #10
    # F add with no item_id given #11
    # F add with duplicate item_id given #12
    #
    # . checkout after add #13
    # . checkin after checkout after add #14
    # F checkin after add #15
    # F checkin after checkin after checkout after add #16
    # F checkout before add #17
    # F checkin before add #18
    # F checkin after remove #19
    # F checkout after remove #20
    #
    # . remove after add #21, #22, #23
    # . remove after checkin after checkout after add #24, #25, #26
    # F remove before add #27
    # F add after removal #28
    # F checkin after removal #29
    # F checkout after removal #30
    # F remove after checkout #31
    # F remove with wrong why (not disposed, destroyed, or released) #32
    # F release without owner info #33
    #
    # . show history basic #34
    # . show history with case id #35
    # . show history with case id (no results) #36
    # . show history with evidence id #37
    # . show history with evidence id (no results) #38
    # . show history with case id and evidence id #39
```

```
# . show history with case id and evidence id (no results) #40
# . show history with random n < chain length #41
# . show history with random n > chain length #42
# . show history in reverse #43
# . show history in reverse with evidence id #44
#
# . verify good chain with several transactions #45
# F verify invalid initial block #46
# F verify invalid block after initial #47
# F verify with duplicate item #48
# F verify with duplicate parent block #49
# F verify double checkin #50
# F verify double checkout #51
# F verify double remove #52
# F verify checkout after remove #53
# F verify checkin after remove #54
# F verify remove before add #55
# F verify remove with wrong why #56
# F verify release without owner info #57
# F verify mismatch checksum #58
#
# . show cases #59
# . show items #60
#
###
```

Option 1: Programming Language-Based

Implementation

Your program must work on Ubuntu 18.04 64-bit with the default packages installed. You may find it helpful to set up a virtual machine to do your development. VirtualBox is a free and open-source VM system.

If you wish to use packages that are not installed on Ubuntu 18.04 64-bit by default, please submit a file with your code named `packages`, with a list of packages that you would like installed before calling `make`. Each line of `packages` must be a valid package name, one package per line. The submission system will automatically install all the dependencies that the package lists.

For example, if you were going to write your assignment in Haskell, you could install the GHC compiler with the following `package` file:

```
ghc
ghc-dynamic
```

Weekly Report

Creating a weekly report for a group project is a great way to keep track of progress, identify any issues, and ensure that everyone is on the same page. It will start in February and you will get the detailed information then.

Checkpoint

We are going to have a checkpoint to check and verify the direction of the project on Mar. 13. You can have some questions and discussions with the TA. There is a submission form (TBA) by midterm.

Submission Instructions

You (the only single man in the group) will need to submit the report and the source code, along with a Makefile and README. The Makefile must create your executable, called `bchoc`, when the command `make` is run. Your README file must be plain text and should contain your name, ASU ID, and a description of how your program works.

A prior TA compiled some resources on how to write a Makefile which might be helpful:

https://www.cs.swarthmore.edu/~newhall/unixhelp/howto_makefiles.html

Report

Just like in forensic investigations, your work on this project must be accompanied by a 5-page report, 12 points, 1.5 spaces, and 1" margins. Include the following in the report:

- Requirements of the project in your own words. This will help you ensure you've captured all the details from above and understand what is expected.
- Design decisions made and why, including programming language, method of storing and parsing the blockchain, etc.
- Challenges you faced while working on the project and your solutions. Include any other lessons learned.
- Discussion on why a blockchain is an inappropriate or appropriate choice to produce a chain of custody solution in digital forensics. How can it be overcome or strengthened?
- Appendix (they do not count toward your 5-page requirement)
    - Screenshots, coding blocks, execution results
    - References (URLs, papers, articles)
    - Describe the role and contributions of each member.
- **Create a demo video (YouTube) and include the link in the report as a reference and it shows satisfaction with the common requirements.**

The report needs to be submitted separately with the implementation using Gradescope. You can see another submission for this report in the Gradescope.

Submission Site

Gradescope

Option 2: Open Source Block Chain Frameworks

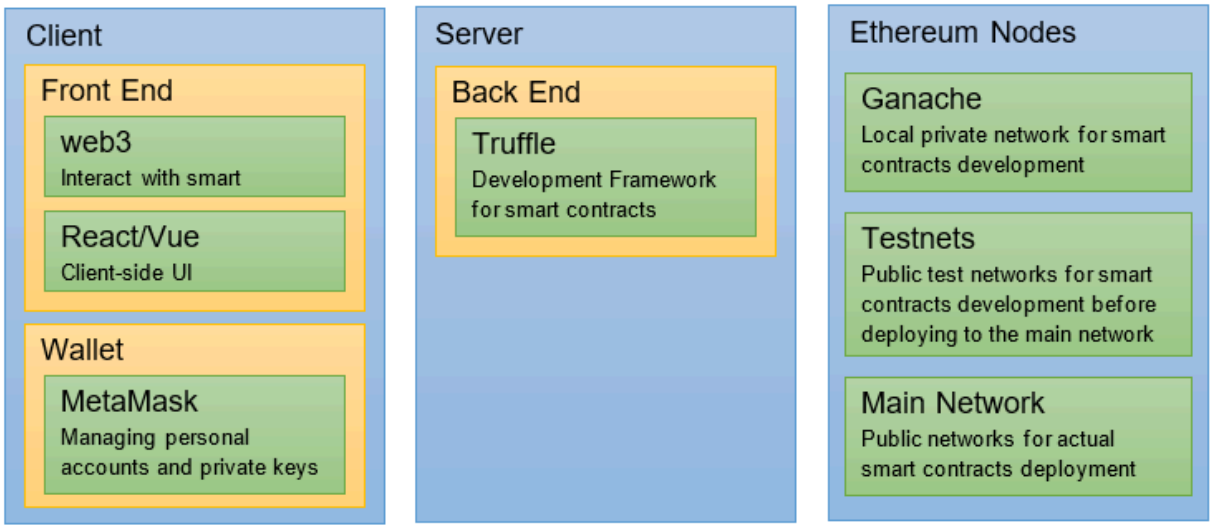
•Hyperledger, •Ethereum, •EOS •Corda •Ripple •Bitcoin •Solana •Cosmos

Hyperledger Guide

https://docs.google.com/document/d/19zzOpoZjWXVhLEActlsJ5ZHIIJCqlUcZhNRnaM057To/edit?usp=sharing

Ethereum Starter Guide

https://www.dappuniversity.com/articles/blockchain-app-tutorial



Ethereum Decentralized Application (Dapp) Architecture and Frameworks

System Requirements

- Operating System: Ubuntu Linux 16.04 / 18.04 LTS or MacOS 10.13 / 10.14
- Hardware: A minimum of 4GB of memory and 2 CPU cores are required. However, for a production deployment, it is recommended to have at least 8GB of memory and 4 CPU cores.
- Docker: Version 17.06 or higher is required.
- Docker Compose: Version 1.14.0 or higher is required.
- Go: Version 1.14 or higher is required.
- Node.js: Version 10.x LTS or higher is required.
- npm: Version 6.x or higher is required.

Submission Instructions

- You (only one man in the group) will need to submit the report (next slide) and the source code you created,
- Your README file must be plain text and should contain your name, ASU ID, and a description of how your program works.
- Make and submit a video recording that includes
- How to run and how each transaction is stored in the database and the creation of blocks along with the source code. (you can use Hyperledger Explorer if you use HLF)
- Instructions on how to run the source code and necessary installations required to run the code.

REPORT

Just like in forensic investigations, your work on this project must be accompanied by a 5-page report, 12 points, 1.5 spaces, 1” margins. Include the following in the report:

- Requirements of the project in your own words. This will help you ensure you’ve captured all the details from above and understand what is expected.
- Design decisions made and why, including programming language, method of storing and parsing the blockchain, etc.
- Challenges you faced while working on the project and your solutions. Include any other lessons learned.
- Discussion on why a blockchain *is not* an appropriate choice for a production chain of custody solution.
- **The demo video (YouTube) you made should be included in the report as a reference and it shows satisfaction with the common requirements.**
- **The report needs to be submitted separately with the implementation using Gradescope.**

- **Generative AI usages: All the generated information must be cited, you should follow Chicago style: https://libguides.asu.edu/ld.php?content_id=73033847**
    - **For example, a numbered footnote or endnote might look like this:**

      1. Text generated by ChatGPT, OpenAI, March 7, 2023, https://chat.openai.com/chat.

      2. ChatGPT, response to "Explain how to make pizza dough from common household ingredients," OpenAI, March 7, 2023.

      3."A modern office rendered as a cubist painting," image generated by OpenAI's DALL·E 2, March 5, 2023.

I encourage you to include screenshots in your report but know that they do not count toward your 5-page requirement, so they should be part of an appendix and referenced accordingly in the text.

Submission Site

Gradescope