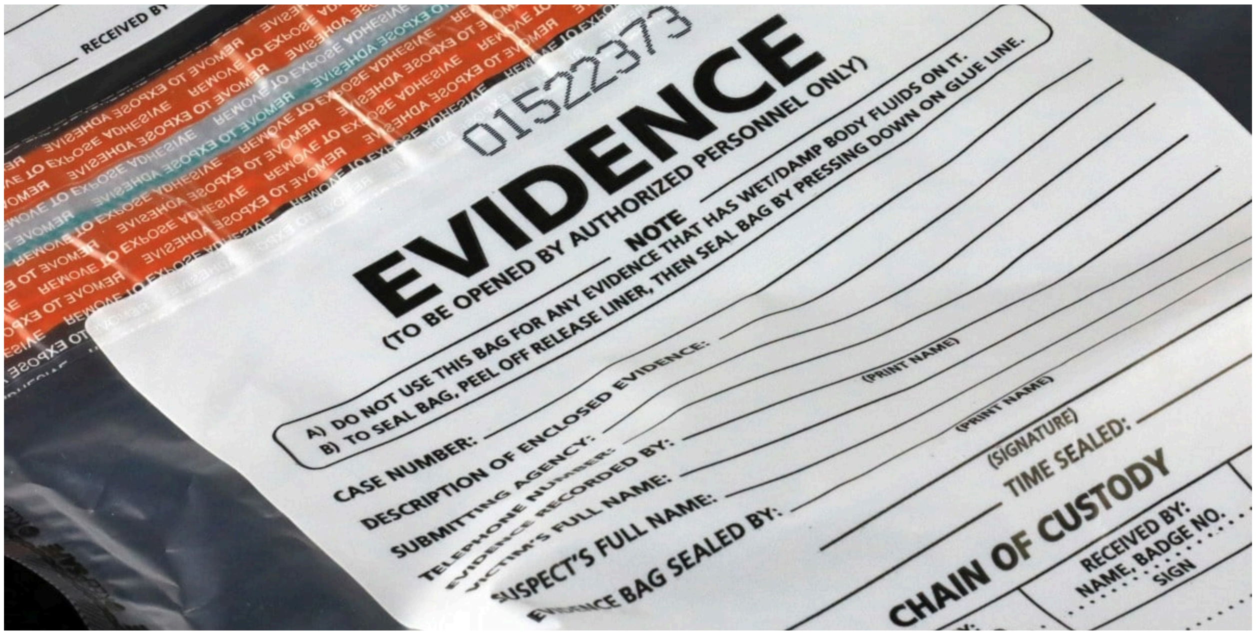


Group Project: Blockchain Chain of Custody

- **Due Date:** 04/30/2025, 11:59 pm
- **Done By:** Groups
- **Checkpoint:** 03/17/2025
- Progress report: **03/15/2025** w/ progress reports
- **Options:** (1) Programming Language-based, (2) Open-source framework-based
- **Submission: Gradescope** (Source code, Demo video, Report) https://www.gradescope.com/courses/945861
- **TA Presentation Slides**

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: NIST's sample chain of custody form
- ~~Google's cached version: Google's Cached Version~~

Following are some additional examples of chain of custody forms:

- Govt. of West Virginia.pdf
- American Association of Seed Control Officials.pdf
- Association of Public Health Laboratories.pdf

Recommended Reading Materials for Chain of Custody:

- CISA Insights: Chain of Custody and Critical Infrastructure Systems
- Chain of custody and life cycle of digital evidence.pdf
- Digital Chain of Custody - State of the Art.pdf
- The Chain of Custody in the Era of Modern Forensics.pdf

Project Description

In this project, your group will develop a program that functions as a digital version of a chain of custody form. A chain of custody form is crucial in digital forensics, as it tracks the history of all interactions with digital evidence. Your program will enable a forensics investigator to record all significant actions involving digital evidence, such as adding new evidence to a case, checking evidence in and out for investigation, and removing evidence from the case.

To ensure the integrity and immutability (unchangeability) of these records, each action will be stored in a Blockchain—a technology you're likely familiar with as a distributed and secure database. In simple terms, a blockchain is a chain of blocks, where each block contains a list of records. These blocks are linked together using cryptographic methods, making it extremely difficult to alter any information without altering all subsequent blocks.

In your project, you will create your own blockchain to store each entry in the chain of custody. By the end of this project, you will have implemented a secure and transparent system that can be used to maintain the integrity of digital evidence records.

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.

Introductory Learning Material for Blockchain: Introduction to Blockchain

Further Recommended Readings: What Is Blockchain? | IBM

Common Requirements

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

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

[ ] >> 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. The password must be that of anyone from the owners.
- **`‘show items’:`** Display all the items corresponding to the case number in the request. The password must be that of anyone from the owners.
- **`‘show history’:`** Display the blockchain entries for the requested item giving the oldest first. The password must be that of anyone from the owners.
- **`‘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.

New Function

- **‘Summary’ :** Iterate through the blocks and print the number of unique item Id’s, Number of blocks in CHECKEDIN, CHECKEDOUT, DISPOSED, DESTROYED and RELEASED. Takes case id as input. Output format shown below

```
Case Summary for Case ID: 1fd9fb21-9de0-4930-8ab0-5dd29fe29acd
Total Evidence Items: 3
Checked In: 3
Checked Out: 0
Disposed: 3
Destroyed: 0
Released: 0
```

Here are the descriptions of the possible arguments:

- **`-c case_id:`** Specifies the case identifier that the evidence is associated with. Must be a valid UUID. When used with show only blocks with the given case_id are returned.
- **`-i item_id:`** Specifies the evidence item’s identifier. When used with show 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.
- **`-r, -reverse:`** When used with history, reverses the order of the block entries to show the most recent entries first.
- **`-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.

Please Note: The purpose of the **'init'** command is to ensure the presence of the Genesis (initial) block in the blockchain. When **'init'** is called for the first time, the program should verify whether a blockchain binary file exists. If no such file is found, it should create one and insert the Genesis block (Initial Block). For subsequent **'init'** calls, the program must check the existence of both the blockchain file and the Genesis block.

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:

```
INITIAL BLOCK =
    Prev_hash = 0,  # 32 bytes
    Timestamp = 0,  # 08 bytes
    Case_id = b"0"*32,      # 32 bytes (32 zero's)
    Evidence_id = b"0"*32,  # 32 bytes (32 zero's)
    State = b"INITIAL\0\0\0\0\0",  # 12 bytes
    creator = b"\0"*12,     # 12 bytes (12 null bytes)
    owner = b"\0"*12,       # 12 bytes (12 null bytes)
    D_length = 14,  # 04 bytes
    Data = b"Initial block\0"
```

Also, note that:

* 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 Case 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.

IMPORTANT: 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. 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.

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

IMPORTANT: MANY conditions 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.

1. 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.
2. 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.
3. 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.

Passwords as Environment Variables

```
"BCHOC_PASSWORD_POLICE": "P80P",
"BCHOC_PASSWORD_LAWYER": "L76L",
"BCHOC_PASSWORD_ANALYST": "A65A",
"BCHOC_PASSWORD_EXECUTIVE": "E69E",
"BCHOC_PASSWORD_CREATOR": "C67C”
```

`AES_KEY = b"R0chLi4uLi4uLi4="` to be hard coded in your program

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.
`bchoc checkout -i item_id -p password`

```
$ ./bchoc add -c c84e339e-5c0f-4f4d-84c5-bb79a3c1d2a2 -i 1004820154 -g
1Ze4XNk8PuUp -p C67C
stdout from your program:
> Added item: 1004820154
> Status: CHECKEDIN
> Time of action: 2024-04-06T01:13:23.842840Z
```

```
$ ./bchoc add -c 30e7c77-f79c-4859-aa76-f4a91e644d3a -i 4070226336 -g
TSDt9YwEdzlc -p AzyatUM7g4jj
stdout from your program:
> Invalid password
Exit code of your program: 1
```

```
$ ./bchoc checkin -i 3741093622 -p P80P
stdout from your program:
> Case: 2193910a-767c-4b8d-abe7-7490c5841a3c
> Checked in item: 3741093622
> Status: CHECKEDIN
> Time of action: 2024-04-06T01:13:24.304725Z
```

```
$ ./bchoc checkout -i 3741093622 -p A65A
stdout from your program:
> Case: 2193910a-767c-4b8d-abe7-7490c5841a3c
> Checked out item: 3741093622
> Status: CHECKEDOUT
> Time of action: 2024-04-06T01:13:24.258536Z
```

```
$ ./bchoc show history -i 2139665479
stdout from your program:
> Case: c989c06dd4e66ec99879c9d7b6811d07
> Item: b4f8f5b6d332cbb9b40f0f1a080bc120
> Action: CHECKEDIN
> Time: 2024-04-06T01:13:33.424078Z
```

```
$ ./bchoc show history -i 3463648746 -p A65A
stdout from your program:
> Case: 0b711606-090e-40fb-8f9a-b82347a43887
> Item: 3463648746
> Action: CHECKEDIN
> Time: 2024-04-06T01:13:30.719569Z
>
> Case: 0b711606-090e-40fb-8f9a-b82347a43887
> Item: 3463648746
> Action: CHECKEDOUT
> Time: 2024-04-06T01:13:30.766074Z
```

Initializing the blockchain:

```
$ ./bchoc init
stdout from your program:
> Blockchain file not found. Created INITIAL block.
```

Checking the initialization:

```
$ ./bchoc init
stdout from your program:
> 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
stdout from your program:
> Transactions in blockchain: 6
> State of blockchain: CLEAN
```

Verifying the blockchain when it has errors:

```
$ ./bchoc verify
stdout from your program:
> Transactions in blockchain: 6
> State of blockchain: ERROR
> Bad block:
ca53b1f604b633a6bc3cf75325932596efc4717fca53b1f604b633a6bc3cf732
> Parent block: NOT FOUND
```

Or:

```
$ ./bchoc verify
stdout from your program:
> 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
stdout from your program:
> Transactions in blockchain: 6
> State of blockchain: ERROR
> Bad block:
99bcaaf29b1ff8dac2c529a8503d92e43921c335c853b1f604b6c33a6c3c475d
> Block contents do not match block checksum.
```

Or:

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

IMPORTANT: 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:

Each test case has different weights (points associated with it), for more details check

```
###
    #
    #  Test cases:( Final tests may have some changes)
    #
Test #000 - Makefile
Test #001 - init with no pre-existing file
Test #002 - init with pre-existing valid file
Test #003 - init with pre-existing valid file
Test #004 - init with pre-existing invalid file
Test #006 - init with additional parameters
Test #007 - add before init - should create initial block
Test #008 - add with one case_id and one item_id and status of the added
item is CHECKEDIN
Test #009 - add with one case_id and multiple item_id values and status of
the added item is CHECKEDIN
Test #010 - add with no case_id given
Test #011 - add with no item_id given
Test #012 - add with duplicate item_id given
Test #013 - checkout after add
Test #014 - checkin after checkout after add
Test #015 - checkin after add
Test #016 - checkin after checkin after checkout after add
Test #017 - checkout before add
Test #018 - checkin before add
Test #019 - checkin after remove
Test #020 - checkout after remove
Test #024 - remove after checkin
Test #025 - remove after checkin
Test #026 - remove after checkin
Test #027 - remove before add
Test #028 - add after remove
Test #029 - checkin after remove
Test #030 - checkout after remove
Test #031 - remove after checkout
Test #032 - remove with invalid why
Test #036 - log with case id
Test #037 - log with evidence id
Test #038 - log with evidence id
Test #039 - log with case id and evidence id
Test #040 - log with case id and evidence id
Test #041 - log with random n < chain length
Test #042 - log with random n > chain length
Test #043 - log in reverse
Test #044 - log in reverse with evidence ID
Test #045 - verify good chain with several transactions
Test #046 - verify invalid initial block
Test #047 - verify invalid block after initial
Test #048 - verify with duplicate item
Test #049 - verify with duplicate parent block
Test #050 - verify double checkin
Test #051 - verify double checkout
Test #052 - verify double remove
Test #053 - verify checkout after remove
Test #054 - verify checkin after remove
Test #055 - verify remove before add
Test #058 - verify mismatch checksum
Test #059 - show cases
Test #060 - show items

Test #061 - add with invalid password
Test #062 - remove with invalid password
Test #063 - remove after remove
Test #064 - log history with an invalid password

    ###
```

Track I: Programming Language-Based

Implementation

You can choose any general-purpose programming language (preferably Python) for implementing the solution. 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 your group project helps track progress, identify issues, and keep everyone aligned. Check the due date on Canvas and submit your report on time. Late submissions are not accepted. (Penalty: A 10% deduction from the project grade per missed report.)

- Weekly Report 1 for Group project: Due 02/14/2025
- Weekly Report 2 for Group project Due 02/28/2025
- Weekly Report 3 for Group project: Due 03/21/2025
- Weekly Report 4 for Group project: Due 04/11/2025
- Report Template: Form
- Submission: Canvas

Checkpoint

We are going to have a checkpoint to check and verify the direction of the project on **March. 17**. You should submit the progress report and any questions using this submission form by **March. 15**.

Final Submission Instructions

You (only one person in the group) will need to submit the following deliverables:

- A report: See details below
- The source code
- Makefile: The Makefile must create your executable, called **`bchoc`**, when the command **`make`** is run.
- README: Your README file must be plain text and should contain your name, ASU ID, and a description of how your program works. If you use Gen AI to complete this homework, you need to include the information following in the README file.

  Generative AI Acknowledgment: Portions of the code in this project were generated with assistance from ChatGPT, an AI tool developed by OpenAI. Reference: OpenAI. (2024). ChatGPT [Large language model]. openai.com/chatgpt

1. Generative AI: You need to insert citations for Generative AI directly in your code, please include a comment that acknowledges the tool used and provides relevant details (why you need this and what was the prompt for each help)

2. . Here's how you can do it:

```
# Generative AI Used: ChatGPT (OpenAI, March 24, 2025)
# Purpose: Needed help writing a function to calculate factorial recursively
# Prompt: "Write a Python function to compute factorial using recursion"

def bubble_sort(arr):
    n = len(arr)
    return arr
```

- Note: When using generative AI, please note that if your code is found to be substantially similar to another student's code or publicly available examples, it may result in an academic integrity violation. To avoid this, make sure you fully understand the generated code and rewrite it in your own words and style.

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 for track I

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.**

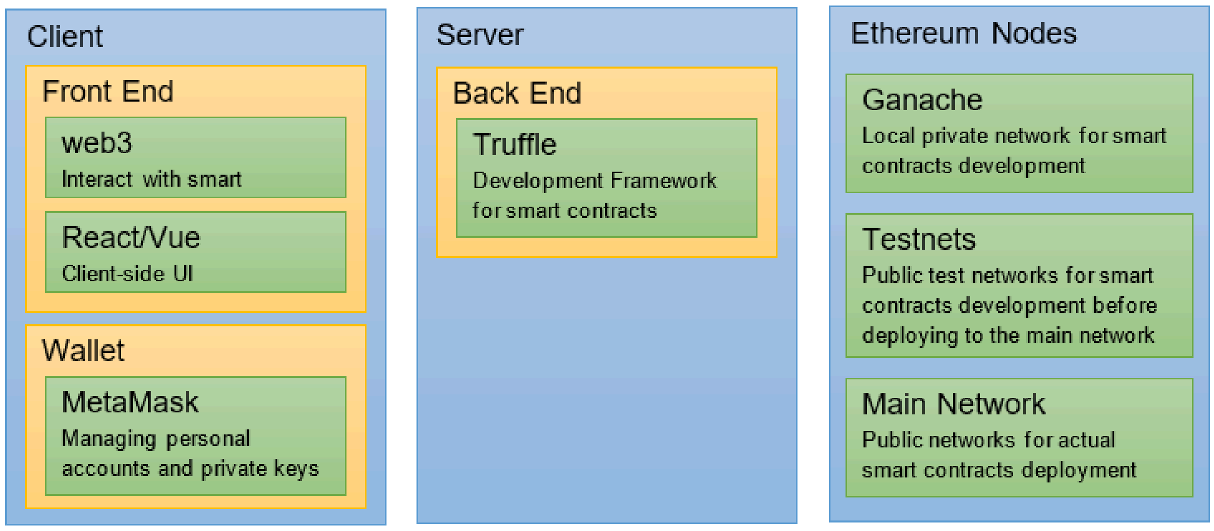
- Create one Report and submit it only once to **“Group Project Report: Track 1 Programming Language-based**” on Gradescope considering the following

- Your README file in the repository must be plain text and should contain a group name, member names, and a description of how your program works.

Track 2: Open Source Blockchain Frameworks

Hyperledger Guide: Hyperledger Guide

Ethereum Starter Guide: [How to Build Blockchain App - Ethereum Todo List 2019 | Dapp University]()


Client
Front End
web3
Interact with smart
React/Vue
Client-side UI
Wallet
MetaMask
Managing personal accounts and private keys
Server
Back End
Truffle
Development Framework for smart contracts
Ethereum Nodes
Ganache
Local private network for smart contracts development
Testnets
Public test networks for smart contracts development before deploying to the main network
Main Network
Public networks for actual smart contracts deployment

Ethereum Decentralized Application (Dapp) Architecture and Frameworks

System Requirements (optional)

- 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.

Final submission

You (only one person in the group) will need to submit the following deliverables:

- Upload: A report (See details below) on Gradescope
- Upload: The source code (zip file of repository) on Gradescope "Group Project Code: Track 2 Open Source Framework-based"
  - README: Your README file must be plain text and should contain your name, ASU ID, and a description of how your program works. If you use Gen AI to complete this homework, you need to include the information following in the README file.

Generative AI Acknowledgment: Portions of the code in this project were generated with assistance from ChatGPT, an AI tool developed by OpenAI. Reference: OpenAI. (2024). ChatGPT [Large language model]. openai.com/chatgpt

  - Generative AI: You need to insert citations for Generative AI directly in your code, please include a comment that acknowledges the tool used and provides relevant details. Here's how you can do it:

```
# This function was generated with assistance from ChatGPT, an AI tool
developed by OpenAI.
# Reference: OpenAI. (2024). ChatGPT [Large language model].
openai.com/chatgpt

def bubble_sort(arr):
    n = len(arr)
    return arr
```

- Upload: A report (See details below) on Gradescope

Report for track 2

Your work on this project must be accompanied by a 10-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.
- Include how to run and any necessary installations are required to run the code.
- Include how each transaction is stored in the database and the creation of blocks along with the source code. (you can use automation tools like Hyperledger Explorer if you use HLF)
- **Include URL links: demo video (e.g., Youtube), source code repository (e.g., Github)**
- Note that the demo video should show satisfaction with the common requirements.
- 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:
    - Text generated by ChatGPT, OpenAI, March 7, 2023, https://chat.openai.com/chat.
    - ChatGPT, response to "Explain how to make pizza dough from common household ingredients," OpenAI, March 7, 2023.
    - "A modern office rendered as a cubist painting," image generated by OpenAI's DALL·E 2, March 5, 2023.

- Create one Report and submit it only once to "**Group Project Report: Track 2 Open Source Framework-based**" on Gradescope considering the following

- Your README file in the repository must be plain text and should contain a group name, member names, and a description of how your program works.