COMP4137/COMP7200 Programming Project: Implementation of a Mini Blockchain

COMP4137/COMP7200

Henry Hong-Ning Dai

March 12, 2025

Important Milestones:

- Group Registration Deadline: **17:00 Mar. 20, 2025**
- Project Phase I Deadline: **17:00 April 3, 2025**
- Project Phase II Deadline: **17:00 April 24, 2025**

1 Introduction

In this project, you are required to complete a programming project for implementing a mini blockchain system. You will jointly work with your teammates to complete this project that contains the components: (1) Transaction generation, (2) Verifiable Merkle tree of transactions, (3) Construction of blockchain, and (4) Integrity verification of transactions and blocks. You are suggested to adopt Python or Java (or JavaScript) as the programming language to implement your project. If you want to use other languages, please let us know.

Upon completion of this project, you need to submit a comprehensive technical report written in English. Moreover, you are also required to present your project in a short video (within 5 minutes).

2 Objectives

After completing this project, you shall obtain an in-depth understanding of how the blockchain system works and be able to build a blockchain platform from scratch. Moreover, you shall earn experience in problem-solving and advanced programming techniques.

3 Requirements

Projects should be done in groups, each with **THREE to FIVE students**. Please specify your work distribution of each member in the final project report. The deadline for grouping is **17:00 Mar. 20, 2025**. For the students who cannot form a group, TAs will help to assign a group manually after the grouping deadline.

Please write the documents in your own words and make sure that the materials (including codes) used have been properly referenced. Please notice the HKBU plagiarism booklet: `http://ar.hkbu.edu.hk/curr/avoid_plagiarism/`.

4 Tasks

A full-fledged blockchain requires multiple technologies, including cryptographic algorithms, data structures, digital signatures, digital hash, peer-to-peer systems, and incentive mechanisms. Due to the limited time, you only need to implement the following basic components: (1) Transaction generation, (2) Verifiable Merkle tree of transactions, (3) Construction of blockchain, and (4) Integrity verification of transactions and blockchains.

4.1 Transaction Generation

Before transaction generation, you need to create a blockchain account for the user. The user will own a pair of private key and public key, where the public key will be used as the user' account for transactions in the blockchain. You need to consider adopting well-known public key cryptographic algorithms, such as Elliptic Curve Cryptography (ECC), RSA, EI-Gamal, etc. To achieve this goal, you may refer to Python cryptography libraries[1] or Java cryptography libraries[2].

Single-input single-output transactions. Your blockchain only generates single-input single-output (SISO) transactions. Take an SISO transaction as an example, in which a user Alice will make a transaction to another user Bob.

One transaction consists of a unique transaction ID, data, an input, and an output, described as follows:

1. Transaction ID: the transaction ID is calculated by taking a hash[3] of the transaction contents.
2. Data: You can fill in this field by the amount of virtual coins and a digital signature (the crypto-hash of the amount of virtual coins).
3. Output: the output consists of the destination (receiver) address.
4. Input: the input consists of the source (sender) address.

Note: You may need to use the sender's private key to generate the digital signature to authenticate the sender at the receiver by verifying it with his/her public key.

The generated transactions will be used for building the verifiable Merkle tree and construction of the blockchain.

4.2 Verifiable Merkle Tree

After obtaining a number of transactions, you can build a verifiable Merkle tree on top of these transactions. You need to adopt SHA-256 hash function or other alternative ones in Python[4] or Java[5]. Consider Figure 1 as an example, in which you can build the Merkle tree from four transactions. Note that you can always assume the number of transactions is a power of 2 for simplicity.

[1] `https://pypi.org/project/securesystemslib/0.14.2/`, `https://pypi.org/project/secp256k1/`

[2] `https://www.java.com/en/configure_crypto.html`

[3] You may choose SHA256 or alternative ones in Python`https://docs.python.org/3/library/hashlib.html#module-hashlib` or Java`https://commons.apache.org/proper/commons-codec/apidocs/org/apache/commons/codec/digest/DigestUtils.html`

[4] `https://docs.python.org/3/library/hashlib.html#module-hashlib`

[5] `https://commons.apache.org/proper/commons-codec/apidocs/org/apache/commons/codec/digest/DigestUtils.html`

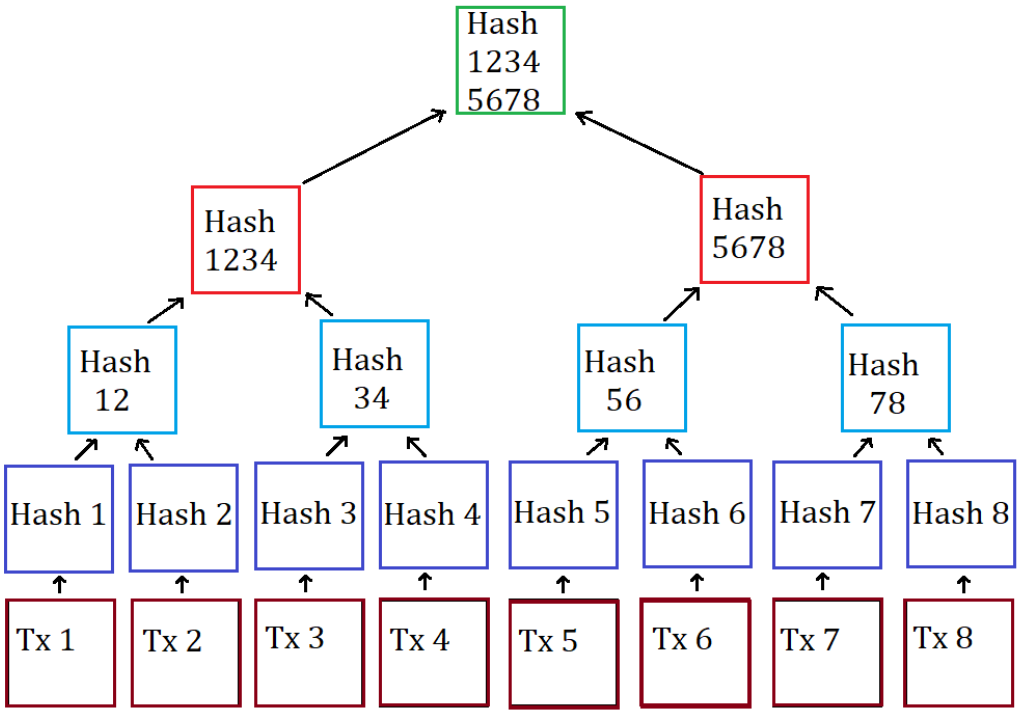

Figure 1: An example of the Merkle tree.

4.3 Construction of Blockchain

Your blockchain essentially consists of (1) a Header and (2) a number of transactions, where the transactions are generated in Section 4.1.

The header contains the following information:

1. The Merkle tree of all the transactions in the block in Section 4.2;
2. Any data to be included in the block;
3. The previous hash, which is generated by the previous block;
4. The time stamp, which is the time when the block is generated;
5. Nounce, which is used to mine a block (see Section 4.4);
6. The current hash, which is generated by the current block.

Note that you will also need to adopt the SHA-256 hash function to generate hash values.

4.4 Mining a block

In this step, you need to implement a Proof-of-Work protocol. As a miner, you need to 1) combine all the information in the block and start nonce from 0, 2) calculate the SHA-256 hash value of all the information, 3) if the output is under the target, add the new block to the blockchain; otherwise, increment nonce by 1 and repeat step 2).

It is worth noting that you may consider a fixed target value (which be refereed to lecture 5).

4.5 Integrity Verification

In this step, you need to design and implement a test program, which can simulate the process of falsifying some transactions or modifying the data (even other fields) in a block. Your implemented system will report whether the transactions/block are falsified.

Hint: You need to generate hash values of falsified blocks/transactions and compare them with the hash values of the header and the Merkle tree.

5 Project Final Report

5.1 Content

Your final report shall include:

1. Introduction and contributions made by your group mates;
2. Related tools/libraries to complete your project;
3. The pipeline flow of your blockchain system and an overall architecture of your system;
4. Design and implementation details (including at least pseudocode, the working flow of algorithms, data structures, and flow charts);
5. Experimental results based on simulation;
6. Conclusion;
7. References;
8. Appendices (if any)[6].

Kindly note that you should not simply copy/paste your program codes or screenshots of your program codes in the final report. Your program codes should be submitted separately as Reproducibility Artifacts (see Section 7).

Requirements of your report:

- Your report should be fully written in English.
- You should write the report in your own words - DO NOT directly copy words/texts from other papers or technical blogs.
- You should properly cite references or other related studies if necessary.
- You should explicitly explain figures or tables when you refer to them.

Any violation of the above rules may lead to a penalty for your score.

5.2 Format

You could use either latex or MS Word to write the final report. However, your group needs to submit a PDF file in the course Moodle before the final report deadline **17:00 April 24, 2025**.

[6]Feel free to include additional material in your report if you wish to include, e.g., additional experimental results and a comprehensive literature survey. Your supplementary material should be appended after the 10 pages (the normal content should be presented within 10 pages).

5.3 Submission

Please submit the completed project report and source codes in the course Moodle according to the given deadline. The implemented methods (codes) that you have used are also suggested to be submitted though they should be in limited size (e.g., 50 MB) due to the limitation of the course Moodle. You also need to specify the compile and execution methods for your codes (i.e., a README file). If your codes are larger than 50 MB, you may offer a URL link to let us download your codes. Details about the source codes and the data can be referred to Section 7.

The deadline for submitting the completed codes of Tasks 1 and 2 (Sections 4.2) is **17:00 April 3, 2025**.

The deadline for submitting the completed codes of Tasks 3, 4, 5 (Sections 4.5) is **17:00 April 24, 2025**.

Note: Late submission will be penalized.

6 Project Presentation Video

Each group should submit a video no longer than 5 minutes to present your project. Your video should present the motivation, the main idea of your proposed solution, and the main results.

The video file MUST meet the following requirements:

- Landscape orientation 16:9
- Resolution: 1280 x 720 (=16:9 ratio, 720p)
- FPS: 30 frames/sec
- In .MP4 format optimized for streaming
- The Video Codec MUST be H.264
- The Audio Codec MUST be AAC
- Max file size: 500MB

The deadline for submitting the project presentation video is **17:00 April 24, 2025**.

7 Evaluation of Reproducibility Artifacts

Artifacts comprise software, datasets, environment configuration, mechanized proofs, benchmarks, test suites with scripts, etc. A complete artifact package must contain (1) the computational artifacts and (2) instructions/documentation describing the contents and how to use them. Regarding the artifact, in particular the code and the datasets, scripts should be provided to support the compilation, deployment, and execution to support the reproducibility of experiments.

The artifact description must be included in a README file along with the artifact, and it must include the following aspects:

- Artifact Identification: (i) the report's title, (ii) the group No, the student names, and student IDs, and (iii) an abstract describing the main contributions of the project and how the role of the artifact in these contributions. The abstract may include a software architecture or data models and its description to help to understand the artifact and a clear description on to what extent the artifact contributes to the reproducibility of the experiments in the report.

- Artifact Dependencies and Requirements: (i) a description of the hardware resources required, (ii) a description of the operating systems required, (iii) the software libraries needed, (iv) the input dataset needed to execute the code or when the input data is generated, and (v) optionally, any other dependencies or requirements. Best practices to facilitate the understanding of the descriptions indicate that unnecessary dependencies and requirements should be suppressed from the artifact.

- Artifact Installation and Deployment Process: (i) the process description to install and compile the libraries and the code, and (ii) the process description to deploy the code in the resources. The description of these processes should include an estimation of the installation, compilation, and deployment times.

- Reproducibility of Experiments: (i) a complete description of the experiment workflow that the code can execute, (ii) an estimation of the execution time to execute the experiment workflow, (iii) a complete description of the expected results and an evaluation of them, and most importantly (iv) how the expected results from the experiment workflow relate to the results found in the report. Best practices indicate that the expected results from the artifact should be in the same format as the ones in the submitted report to facilitate the understanding of the scope of the reproducibility. For instance, when the results in the report are depicted in a graph figure, ideally, the execution of the code should provide a (similar) figure (there are open-source tools that can be used for that purpose such as gnuplot and Matplotlib).

8 Grading Criteria

Your project will be graded primarily based on the following weighting scheme:

- Project Phase I (30%)
- Project Phase II (30%)
- Project Report (20%)
- Evaluation of reproducibility artifacts on executable program codes (10%)
- Presentation and demonstration (10%)