DAT 510: Assignment 3

Submission Deadline: 23:59, Friday, Nov. 8, 2024

Blockchain and Cryptographic Security

Introduction

In this assignment, you will build a toy blockchain by completing the provided skeleton code. The primary objective is to deepen your understanding of the core components of blockchain technology, with a particular emphasis on cryptographic aspects.

Part 1: Implementation

Objective

- **Implement** the fundamental components of a blockchain system, including wallets, transactions, blocks, and the blockchain itself.
- **Focus** on cryptographic elements such as hashing and digital signatures, using SHA-256 and ECDSA with the SECP256k1 elliptic curve.
- **Simplify** the system by excluding the consensus mechanism; assume a single miner independently solves the cryptographic puzzle and adds blocks to the blockchain.

Implementation Details

Cryptographic Algorithms

The following cryptographic algorithms are used:

- **SHA-256**: Used for hashing data.
- **ECDSA (Elliptic Curve Digital Signature Algorithm)**: Utilized for digital signatures, specifically with the SECP256k1 elliptic curve.

Note: For ECDSA and SHA-256, use the following Python libraries:

- ECDSA Python Package
- Hashlib Python Package

Core Components

The toy blockchain consists of the following core components:

1. **Wallet**: Manages users' private and public keys, enabling transaction creation.
   Refer to lecture notes 9, 10, 13, 16.

2. **Transaction**: Represents the transfer of value or data between participants.
   Refer to lecture notes 11, 13, 16.
3. **Block**: A collection of transactions that are chained together to form the blockchain.
   Refer to lecture notes 11, 16.
4. **Blockchain**: Oversees the chain of blocks and validates transactions and blocks.
   Refer to lecture note 16.

Workflow Overview

To understand how the toy blockchain operates, review the `main.py` file. The process involves the following steps:

1. **Initialize the Blockchain**: Set up the initial state of the blockchain.
2. **Generate Users**: Create 10 users using the `Wallet` class.
3. **Create Transactions**: Generate 100 transactions between the users.
4. **Mine a Block**: Collect the transactions into a single block and mine it.

Detailed Workflow

The workflow of your blockchain code from a cryptographic perspective involves the following steps:

1. **Key Generation and Address Creation (Wallet Class):**
   - Users (nodes) generate a pair of cryptographic keys (private and public) using ECDSA (Elliptic Curve Digital Signature Algorithm).
   - The public key serves as the user's address, simplifying the process of identifying the sender or recipient in a transaction.
2. **Transaction Creation and Signing (Transaction Class):**
   - When creating a transaction, the wallet includes the sender's address, recipient's address, and the amount to be transferred.
   - The transaction is then signed using the sender's private key, producing a digital signature. This signature ensures the authenticity of the transaction and proves that the transaction was indeed created by the owner of the private key.
   - The `sign_transaction` method uses the SHA-256 hash of the transaction data as input to ECDSA for signing, generating a unique signature for the transaction.
3. **Transaction Validation (Transaction Class):**
   - Before a transaction can be added to a block, it must be validated. The `is_valid` method checks if:

    - The transaction contains a valid signature.
    - The sender's public key can verify the digital signature using ECDSA, confirming that the sender authorized the transaction.
    - Mining rewards or special transactions (from the address "0") bypass signature validation.

4. **Block Creation (Block Class):**
   - Each block contains an index, a timestamp, a list of transactions, a reference to the previous block's hash, and a nonce (used for Proof of Work).
   - The block's header (including transactions and other metadata) is serialized and hashed using SHA-256 to generate the block's unique hash.
5. **Proof of Work (Blockchain Class):**
   - The `proof_of_work` method implements the Proof of Work (PoW) algorithm. The block's nonce is iteratively incremented until the block's hash satisfies a condition where it starts with a certain number of leading zeros (defined by the blockchain's difficulty).
   - This process ensures that adding a new block requires computational work, making it costly to alter the blockchain.
6. **Block Validation and Addition (Blockchain Class):**
   - Once the PoW is solved, the block is added to the blockchain, but only after two checks:
     - The block's previous hash must match the hash of the last block in the chain (ensuring the integrity of the chain).
     - The block's proof (its hash) must be valid according to the PoW criteria.
   - The `is_valid_proof` method confirms the block hash is correct and follows the difficulty rules.
7. **Mining (Blockchain Class):**
   - The mining process involves selecting valid transactions from the pool, adding them to a new block, solving the PoW, and adding the block to the blockchain if successful.
   - The mined block is then linked to the previous block, forming the blockchain.
8. **Blockchain Integrity (Blockchain Class):**
   - The `is_chain_valid` method checks the validity of the entire chain. It ensures that:
     - The hash of each block matches the computed hash for that block.

  - Each block correctly references the hash of its predecessor.
  - All transactions within each block are valid, meaning the signatures are correct.

This cryptographic workflow ensures the security and integrity of the blockchain, making it resistant to tampering or fraud. Key cryptographic components used include SHA-256 for hashing and ECDSA for transaction signing and verification.

Implementation Steps

Complete the implementation by filling in the methods marked with `TODO` comments in the following files:

- `wallet.py` (contains the `Wallet` class)
- `transaction.py` (contains the `Transaction` class)
- `block.py` (contains the `Block` class)
- `blockchain.py` (contains the `Blockchain` class)

Detailed comments are provided within each method to guide you through the implementation process.

Testing and Validation

Testing Your Implementation

To verify your code, use the unit tests provided for each component:

- `test_wallet.py`
- `test_transaction.py`
- `test_block.py`
- `test_blockchain.py`

Run the tests using the following commands:

```
python -m unittest -v test_wallet.py
python -m unittest -v test_transaction.py
python -m unittest -v test_block.py
python -m unittest -v test_blockchain.py
```

Example Output

After completing the `compute_hash` method in `transaction.py`, running the test may produce the following result:

```
python -m unittest -v test_transaction.py
test_compute_hash (test_transaction.TestTransaction) ... ok
test_is_valid (test_transaction.TestTransaction) ... FAIL
test_sign_transaction (test_transaction.TestTransaction) ... FAIL
======================================================================
FAIL: test_is_valid (test_transaction.TestTransaction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/path/to/test_transaction.py", line 56, in test_is_valid
    self.assertTrue(self.transaction.is_valid())
AssertionError: None is not true
======================================================================
FAIL: test_sign_transaction (test_transaction.TestTransaction)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "/path/to/test_transaction.py", line 44, in test_sign_transaction
    self.assertIsNotNone(self.transaction.signature)
AssertionError: unexpectedly None
----------------------------------------------------------------------
Ran 3 tests in 0.000s

FAILED (failures=2)
```

When all tests pass, the output should look like this:

```
python -m unittest -v test_transaction.py
test_compute_hash (test_transaction.TestTransaction) ... ok
test_is_valid (test_transaction.TestTransaction) ... ok
test_sign_transaction (test_transaction.TestTransaction) ... ok
----------------------------------------------------------------------
Ran 3 tests in 0.015s

OK
```

Important: Do NOT modify the test code to make your implementation pass. If you believe your implementation is correct but the tests fail, please contact one of the TAs.

Code Approval

Your code will be approved once you:

- **Pass** all the tests.
- **Show** the output of `main.py`. (Run `python main.py`)
- **Provide explanations** for your implementation.

Part 2: Report Writing

Report Guidelines

You are required to write a report **as if you are the inventor of this toy blockchain**. The tone should reflect that all choices of cryptographic algorithms and technical components are your own proposed methods, including the entire system.

In addition to the fundamental aspects, your report should include the following contents:

- **System Overview**: Provide your own visual diagrams that illustrate how the entire system operates.
- **Algorithm Representation**: Use mathematical equations and proper pseudocode to describe the algorithms used in the blockchain.
- **Experimental Evaluation**: Conduct experiments by varying mining difficulty and the maximum transactions per block. Discuss the advantages and disadvantages of adjusting these parameters. You can reference the code in `main_async.py` to simulate real-world scenarios or create your own simulation.
- **Missing Features**: Identify and explain any components or mechanisms missing in this toy blockchain compared to a real blockchain (implementation not required).

Part 3: Bonus Points (Max. 10pts)

This is an OPTIONAL part. For part 3, create a separate project to avoid conflicts with the unit tests. Include any bonus implementations in your report. Choose one of the two options.

Option I. Integrating Assignment 2 with the Blockchain (Max. 10pt)

- **Objective**: Explore the possibility of using the toy blockchain as a database.
- **Tasks**:
  - Implement encrypted message storage into the existing `Transaction` class.
  - Implement a method to retrieve messages from the blockchain.
- **Considerations**:
  - Do we need to store the entire message?
  - How can we retrieve messages more efficiently?
- **Note**: The amount of bonus points awarded will depend on the quality and depth of your analysis.

Option II. Completing Missing Parts (Max. 10pt)

- **Objective**: Implement missing components in the toy blockchain to make it more comparable to Bitcoin.
- **Examples**:
  1. **Merkle Tree (5pt)**: Organize transactions in a block efficiently and enable quick verification of transaction integrity.
  2. **UTXO (Unspent Transaction Outputs) (10pt)**: Manage and track the ownership of funds by maintaining a set of unspent transaction outputs.
  3. **Transaction Fees and Incentives (5pt)**: Introduce fees to incentivize miners to include transactions in blocks and ensure the sustainability of the blockchain network.
  4. **Consensus Mechanism (10pt)**: Implement a decentralized consensus protocol, such as Proof of Work (PoW) or Proof of Stake (PoS), to secure the blockchain and validate new blocks.
- **Note**: You can earn a maximum of 10 points by selecting either one 10-point option or two 5-point options. Alternatively, you may choose just one 5-point option. The final score will depend on the quality of your implementation and analysis.

Additional Materials

- Bitcoin: https://bitcoin.org/bitcoin.pdf

Assignment Approval (by TA and SA)

Assignment approval will have a weight on your grade for the assignment. If you are not going to get the approval before the deadline, your assignment will not be evaluated and you will fail the assignment.

What needs to be done to get the approval of the assignment:

1. Show all parts of assignment are working i.e, show the code with proper comments, results.
2. The code should have a proper README file that describes the contents of the directory and any special instructions needed to run your programs (i.e. if it requires and packages, commands to install the package. describe any command line arguments with the required parameters).
3. Source code submitted for the assignment should be your own code. If you have used sources from the internet everything should be added to the references. If you used someone's code without reference, that will also be treated as plagiarism.
4. Provide the references in the code and Report, show these parts for TA's and Student Assistants.
5. You **CAN** use available libraries/packages/classes for implementing the core functionality of the assignment.

Use **Python** as a programming language for the implementation of this assignment.

Assignment Submission

Deadline: 23:59, Friday, Nov. 8, 2024 (submit your assignment through canvas)

Final submission:

1. Source Code
   - Source code submitted for the assignment should be your own code. If you have used sources from the internet everything should be added to the references. If you used someone's code without reference, that will also be treated as plagiarism.
   - Source code should be a single, compressed directory in .tar.gz or .zip format.
   - Before compressing the folder, ensure its name is `DAT510_MA3_YourName`.
   - Directory should contain a file called README that describes the contents of the directory and any special instructions needed to run your programs (i.e. if it requires packages, commands to install the package. describe any command-line arguments with the required parameters).
   - You **CAN** use available libraries/packages/classes for implementing the core functionality of the assignment.
2. A **separate** report with PDF format
   - Please upload the report separately.
   - Texts in the report should be readable by humans, and recognizable by machines;
   - Other formats will **NOT** be opened, read, and will be considered missing;
   - Report should follow the formal report style guide on the next page.
   - Compile your findings, implementation, and analysis into a comprehensive report (3000-5000 words).
   - Each student should write an individual report. Each report will be checked for plagiarism. If it is copied from somewhere else, you will fail the assignment.

NOTE: Please upload the archive file in *.zip, *.tar only and report in *.pdf format only to the website https://stavanger.instructure.com/.

Note: The assignment is individual and can **NOT** be solved in groups.

Project Title

Abstract

A one-paragraph summary of the entire assignment consists of any findings, conclusion and recommendations.

1. Introduction

- Background: Discuss the motivation behind creating a decentralized ledger system. Explain the limitations of existing centralized systems (e.g., vulnerability to single points of failure, lack of transparency).
- Objective: Clearly state the goals of your blockchain invention, such as enhancing security, ensuring transparency, and enabling trustless transactions.

2. System Architecture

Include diagrams and flowcharts that illustrate the components and interactions within your blockchain system.

3. Cryptographic Foundations

- Hash Functions (SHA-256): Explain the choice of SHA-256 for hashing data, its properties (collision resistance, pre-image resistance), and its role in ensuring data integrity.
- Digital Signatures (ECDSA with SECP256k1): Detail how digital signatures authenticate transactions, ensuring that only legitimate users can initiate transactions.

4. Core Components and Functionality

Detail each component of your blockchain system, explaining its role and operation.

- Wallet
  - Function: Manages private and public keys for users.
  - Operation: Describe key generation and usage in signing transactions.
- Transaction
  - Structure: Define the elements of a transaction (e.g., sender, receiver, amount, signature).
  - Validation: Explain how transactions are verified for authenticity and integrity.

- Block
  - Composition: Detail what constitutes a block, including transactions, timestamp, nonce, and previous block hash.
  - Hashing: Describe the hashing mechanism (SHA-256) used to secure the block.
- Blockchain
  - Chain Formation: Explain how blocks are linked to form the blockchain.
  - Integrity Maintenance: Discuss how the chain's immutability is preserved through cryptographic hashes.

5. Consensus Mechanism (Simplified)

- Current Approach: Since the assignment excludes a full consensus mechanism, describe the simplified approach where a single miner adds blocks.
- Future Enhancements: Speculate on potential consensus mechanisms (e.g., Proof of Work with multi-miners, Proof of Stake) that could be integrated to decentralize block validation.

6. Experimental Analysis

Conduct experiments to evaluate the performance and scalability of your blockchain system.

- a. Mining Speed
  - Difficulty Adjustment: Analyze how varying the difficulty level impacts the time required to mine a block.
  - Results: Present data (e.g., mining time vs. difficulty) using tables or graphs.
- b. Throughput
  - Transaction Rate: Measure the number of transactions your blockchain can process per second.
  - Scalability: Discuss how the system performs as the number of transactions increases.

7. (Optional) Advanced Blockchain Features

Include this section only if you have implemented the bonus components. If not, feel free to omit it.

8. Future Work

Acknowledge the limitations of your toy blockchain and outline potential enhancements to make it more robust and comparable to full-fledged blockchain systems.

9. Conclusion

Summarize the key aspects of your blockchain invention, reiterating its significance and potential impact on various industries. Reflect on the journey of developing the system and its foundational role in advancing decentralized technologies.

References

A bibliography of all of the sources you got information from in your report.