CS61065 Theory and Applications of Blockchain

Assignment - 1 (Autumn 2024)

Deadline: August 22 2024 EOD

Each student needs to solve and upload this assignment individually. Please check the submission instructions at the end.

We have a zero-tolerance policy on plagiarism. Any form of plagiarism will lead to zero marks for all the cases that will get detected. Also follow the policies regarding use of AI tools as discussed during the first class.

There are no restrictions on the programming languages to be used. We would prefer Python, but you can choose your favorite programming language.

Part A

Write a program to simulate a model for validating blocks in a blockchain. Each block has a set of transactions and a nonce value. The program needs to identify a valid nonce such that the hash of the block header starts with a specified number of leading zeros, and the sum of the ASCII values of all characters in the hash is divisible by a given number ***D***.

Input:
The test case contains ***B*** blocks. The first line of the input will be the integer ***B***. This is followed by inputs for ***B*** blocks. The first line of each block input contains an integer ***T***, the number of transactions in that block. This is followed by ***T*** lines, each a string of length ***S*** representing a transaction (the string will have lowercase characters only). These ***T*** lines are followed by two integers, ***Z*** and ***D***:

- ***Z*** is the number of leading zeros required in the hash.
- ***D*** is the divisor for the sum of the ASCII values of the hash characters.

Output:
For each block in the test case, print the **valid nonce** and **the hash** of the block header.
Note: While taking the SHA-256 hash, convert it to the hexadecimal string representation. The block header is formed by concatenating all transactions followed by the nonce.

Note:

If there are odd number transactions in a block, consider the last block twice.

Constraints:

- $0<T\leq 50$
- $S=20$
- $1\leq Z\leq 8$
- $1\leq D\leq 100$

Example:
Input:

```
2
3
ysaqiblcmbaawxdixzrs
pnbcgrefcuqxnftimghe
tzniehbjkldzxucqtufw
2 4
2
vkqcjncnswfsdzddyiwl
fuewvxwdvypokxdtzxcl
3 4
```

Output:

```
Valid Nonce: 1670
Hash: 0080a354e367afc1f97d111adbd98195d7f9fd734ad45fb6bd8ea122e6a078f7
Valid Nonce: 3407
Hash: 000af16d3d27538838caa225430284b087fbd946187d528e5ad5ebb413ccd123
```

Explanation:
For the first block, the program finds the nonce 1670, which results in a hash starting with 2 leading zeros, and the sum of the ASCII values of the hash characters is divisible by 4. Similarly, for the second block, the nonce 3407 results in a hash starting with 3 leading zeros, and the sum of the ASCII values of the hash characters is divisible by 4. The exact valid nonces and hashes will vary based on the input transactions.

Part B

Write a program to construct blocks from a set of transactions and verify their integrity using RSA cryptography. The program should form a blockchain structure of B blocks, linked by their respective hashes. The genesis block contains a single, given transaction. Subsequent blocks contain T sets of transactions. For each block, compute the RSA signature of the block header and verify if the previous block's header hash is correctly embedded in the current block's signature.
For each block, the block version is '02000000'.

Block Structure:

- **Block Version:** A fixed string '02000000' representing the block format.
- **Previous Block Hash:** The RSA signature of the previous block's header.
- **Merkle Root:** The root hash of the Merkle tree constructed from block transactions (computed separately).
- **Transaction Data:** A set of T transactions.
- **Nonce:** A random value (reused from Part A's)

RSA Signature Generation:

1. The block header (version, previous block hash, Merkle root, and nonce) is concatenated into a single string.
2. This string is hashed using a secure hash function (e.g., SHA-256).
3. The hash is signed using the private key of the block creator.
   Hash of the block header = Hash(block version + previous block header hash + merkle root)
   ***Merkle root computed separately.**

Block Verification:

1. Extract the previous block hash from the current block's signature.
2. Verify the RSA signature of the current block's header using the block creator's public key.
3. If the signature is valid and the extracted previous block hash matches the actual hash of the previous block, the block is considered valid.

Input:

The input will be provided in a text file. Each line represents a block. The format for each block is as follows:

- The number of blocks (B).
- For each block:
  - **Number of transactions:** An integer T
  - **Transactions:** T lines, each containing a transaction (string)
  - **Public key:** A string representing the public key

Output:

For each block in the test case, print "Valid" or "Invalid" based on the signature verification result.

Additional Considerations:

- The first block (genesis block) is not included in the input. Therefore, the first block in the input is actually the second block in the blockchain. The genesis block is formed of only one transaction. This is the coinbase transaction with the string 'coinbase'.

  Hash of the coinbase block header = Hash(block version + merkle root)

Example:
Input:

```
2
1
ysaqiblcmbaawxdixzrs
<public_key_string>
<private_key_string>
2
vkqcjncnswfsdzddyiwl
pnbcgrefcuqxnftimghe
<public_key_string>
<private_key_string>
```

Output:

```
Valid
Invalid
```

Submission Instructions:

Both Part A and Part B of this assignment have to be submitted in Moodle.

For Moodle submission, use CSE Moodle: https://moodlecse.iitkgp.ac.in/moodle/login/index.php. We have created a course page there with the course name “CS61065: Theory and Applications of Blockchains”. You can join the course page using the key: STUD@A23BC