Practical Homework 1

Foundations of Blockchain Technology and Cryptocurrencies

Fall 2019

Sharif Blockchain Lab
Sharif University of Technology
Department of Electrical Engineering

Deadline: 17th Aban 23:55

- Read each problem completely before coding. Provide exactly what the questions ask for.
- Code submissions must be done in Python. You can use libraries and functions of SHA_{256}. Search for the hashlib library.
- Pay attention to the input and output. Answers will be judged automatically. Answers must be exactly like the samples.
- For problems with test cases (Problem 2 and 4), sample inputs and outputs has been provided. Provide the output exaclty like the samples.
- Some codes need to be implemented as Python classes to be judged. Code samples for these problems have been provided. **Do not change the name of the classes, functions and files that is supposed to be filled.** You may add other functions, classes and files if needed.
- If you have difficulties in submitting your answers, ask your questions in Quera.
- The policy for delayed submission is as follows: up til 24 hours after the deadline, 75 percent of the grade is given. Between 24 hours and 48 hours after the deadline, 50 percent of the grade is given. 48 hours after the deadline, submission is closed.

Problem 1 - Hash function (250 points)

The goal of this exercise is to find a preimage and collision for a hash function. We construct the hash function $H(x)$ as the 24 least significant bits of the SHA_{256} of x. More precisely,

$$H(x) := SHA_{256}(x) \ mod \ 2^{24}.$$

For example lets have $m_1 =$ "$Salam$" and $m_2 =$ "$Khodahafez$", therefore

$$SHA_{256}(m_1) = \text{e6aff8d7aabcab5ae824356814a091ba2867835cd6b50e6103b95e29866732a5},$$
$$\text{H}(m_1) = \text{6732a5},$$
$$SHA_{256}(m_2) = \text{82129f7ea63192c88bfa22f2613e5150ce483505549ddf08399d4589e536e8ae},$$
$$\text{H}(m_2) = \text{36e8ae}.$$

Use websites like https://passwordsgenerator.net/sha256-hash-generator/ to check that you are generating the same hashes.

Part 1 (100 points)

Assume a login website only stores $H(x)$ of passwords. Your Password is your Sharif student id. Find two other passwords m_1 and m_2 such that you can login to your account with these two passwords as well. For example if your student number is "91234567" then you have to find two inputs m_1 and m_2 in which $H(m_1) = H(m_2) = H(91234567)$.

Hint: Do not try to implement SHA_{256}! Use libraries in the language that you are programming with.

Submission: Create a file named "ans.txt". In the first line, write your student number. In the second and third line, write two strings with the described property. Compress this file and send it in Quera. **No code is required for this part**.

Part 2 (150 points)

Assume the website stores the hash of usernames as well. So every user in this website is known by a tuple (H(username),H(password)). Write a code that output two users that are identical to website. (H(username1)=H(username2),H(password1)=H(password2),password1≠password2≠username1≠username2)
There is a time limit of 1s for the judgment of this part.

Submission: You must submit your code in the specified format. A sample has been provided. The function $findCollsion()$ must return a list with four values.

Hint: A brute force approach will not get you a full score. See The Birthday Problem.

Problem 2 - Computing Discrete Logarithm (300 points)

In this problem, we will find an algorithm to compute discrete logarithm in Z_p^*, when p is a prime number such that $p = 2^n + 1$ for some n. More precisely, for a given prime number p, generator g and arbitrary $y \in Z_p^*$, we want to calculate x such that $g^x = y \bmod p$.
Let $x = b_{n-1}2^{n-1} + b_{n-2}2^{n-2} + ... + b_1 2 + b_0$ be the binary representation of x.
There is a time limit of 1s for the judgment of this question

Submission: For each part, submit a code in python. The inputs and outputs are as described below. Read the input from the standard input stream (console), and write the output in the standard output stream (console).

Remark: If you get a full score in any of the three parts, you will get the scores for the parts before that.

Input: (The format of the input is the same for all three parts)
In the first line, q, the number of queries. Then for $i = 0$ to $q - 1$, in lines $3i + 2$, $3i + 3$, $3i + 4$, a prime p, a generator g of Z_p^* and a value y in Z_p^*.

Part 1 (100 points)

Calculate the least significant bit of x, i.e. b_0, for each of the queries.
Output: For each query, output the LSB of x, i.e. b_0, in a separate line such that $g^x = y \bmod p$.
Hint: Use $y^{(\frac{p-1}{2})}$.

Part 2 (100 points)

Calculate the 2 least significant bits of x, i.e. b_1b_0, for each of the queries.
Output: For each query, output the 2 LSB's of x, i.e. b_1b_0, in a separate line such that $g^x = y \bmod p$.

Part 3 (100 points)

Calculate x for each of the queries.
Output: For each query, output x in a seperate line such that $g^x = y \bmod p$.

Hint: A brute force approach will not get you a full score.

Problem 3 - Merkle Tree (300 points)

In this problem, we will learn how merkle trees are constructed. To do this we will first introduce padding.

Padding: We know how to construct a merkle tree when the number of files is a power of two. Now consider the case where the number of files isn't a power of two. We need a unique way of constructing the tree. Our padding method works as it is shown in Figure 1. At each level, if the number of existing hashes are even, we construct the higher level, otherwise, we duplicate the last value. In Figure 1, the letters are real hashes, and the letters with a prime (') are the copies of the corresponding letters.

Part 1 - Constructing Merkle Tree (150 points)

Write a code that calculates the merkle root of 20 files. Use SHA_{256} as the hash function. We will be using the hex format of the hashes of the files (see the example of Question 1). In the intermediate stages, concat the strings of the hashes from the level below.

19 files have been given to you in txt format. Make an 20th file named "file20.txt" and put in the 'resource' directory. In this file, write one sentence about yourself. Then construct a merkle tree with these 20 files (order them with the name of the files). Use the padding method explained above whenever necessary. **Use the template provided.**

Submission: Fill the source.py file, such that the function *calculate_merkle_root*() returns the hex format of the merkle root. Submit a zip file like the one provided in the sample.

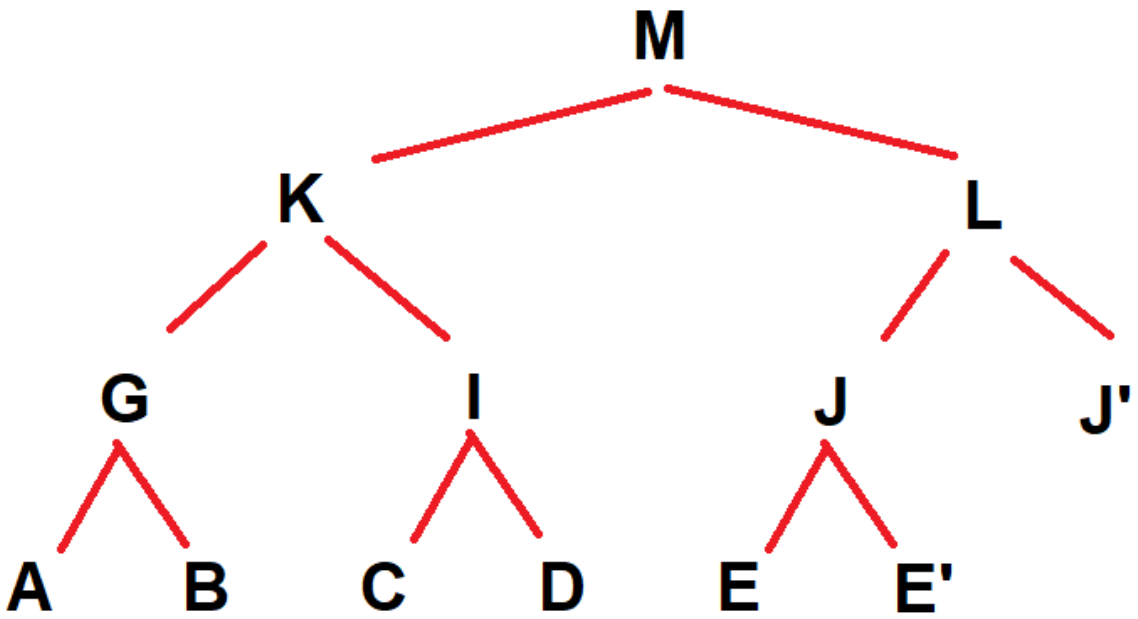

Figure 1: How to pad a merkle tree to construct it.

Part 2 - Merkle Proof (150 points)

In this problem we will see how the merkle tree constructed in the previous section can be used as a proof of existence of data.

Submission: Submit a code in python. The inputs and outputs are as described below. Read the input from the standard input stream (console), and write the output in the standard output stream (console).

Input: In the first 20 lines, the hahses of the 20 files is provided in hex format. In line 21, a number m is written which denotes the file that we want the proof for.

Output: In the first line, print the merkle root of the data provided. In the second to sixth line, provide the proof for the m^{th} file. Output the hashes from the leaves to the root (bottom to top). Note that an example proof has been provided in the Figure 2.

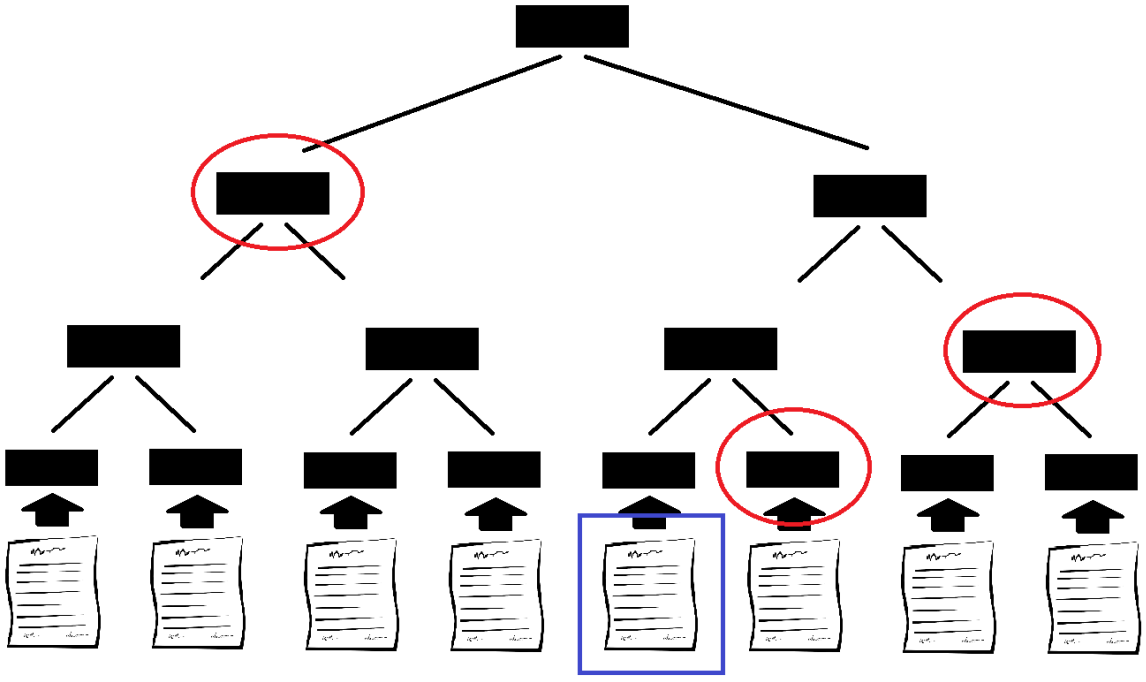

Figure 2: In this example, we want to proof the existence of the 5th file (blue box). In a real case scenario, we will send the file and the hashes in the red circles. But here we will omit sending the file and only send the hashes in the red circles.

Problem 4 - Implementing LCR Algorithm (150 points)

In this problem you are going to implement LCR leader election algorithm for an asynchronous ring. You can find more details about this algorithm on Nancy Lynch, 15.1.1.

Input: Please look at the source codes which has been given to you. You will find a config.json file. It is our input. It is an array of arrays which first value of each array is the UID of corresponding node and the second one is the delay of channel which delivers message to the next node.

Output: After some amount of time each node has to write to the console that who is the leader. Remember that this is an asynchronous setting, Not only liveness is not guaranteed but also there is no termination rule for the algorithm.

Submission: Just sumbit node.py file. Please do not change other files.