CS 5158/6058 Data Security and Privacy, Spring 2018
Project 4: Proof of Work

Instructor: Dr. Boyang Wang

Due Date: 04/10/2018 (Tuesday), 11:59pm.
Format: Please submit a zip file of your code in Blackboard.
Total Points: 10 points
Note: This is an individual project.

1 Project Description

In bitcoin mining, the most important step is to find solutions for Proof of Work (POW). In this project, you will need to simulate the mining process by implementing Proof of Work (POW). More specifically,

– Given a POW difficulty d, your program should be able to compute a target t based on this POW difficulty d, where the first d bits in this target are all 0s and the rest of the bits in this target are 1s. In addition, you program should be able to write this target to a file.

– Given an input m and a target t, your program should be able to compute a POW solution s, s.t.,

$$\mathsf{Hash}(m||s) \leq t$$

where $||$ is a concatenation operator. For example, if $m =$ `bear` and $s =$ `cats`, then $m||s =$ `bearcats`. In addition, you program should be able to write this solution s to a file.

– Given an input m, a POW solution s, and a target t, your program should be able to verify whether this POW solution is a valid solution for input m. Specifically, you program should output

$$\begin{cases} 1, & \text{if} \quad \mathsf{Hash}(m||s) \leq t \\ 0, & \text{otherwise} \end{cases}$$

In this project, you should use SHA256 as your hash function.

2 Basic Requirements

Programming Language: You can use either C/C++, Python or Java. If you choose to use C/C++, CMake is recommended (but not required). You can choose any IDE you like, the code you submit should be able to compile and run in Linux or Windows. Please provide a clear description about how to compile and run your code in the Readme file.

Crypto Libraries: You can leverage third-party libraries, such as `openssl` (C/C++), `BouncyCastle` (Java), etc., to implement SHA256. You do not need to build the functions of SHA256 by yourself.

Program Directory: Please name your project folder as `pow_m123456`, where `pow` is the name of this project and `m123456` is your UCID. The recommended directories of your program should be organized as follows:

```
./pow_m123456/src
./pow_m123456/build
./pow_m123456/data
./pow_m123456/Readme.txt
```

Normally, folder `src` should include all the source files and your own header files, e.g., `.cpp` and `.h` files. All the object files and executable files, e.g., `.o` files, should be under folder `build`. Folder `data` has all the given files and data, and also includes all the files and results generated by the program. In `Readme.txt` file, you should write a description of your code, show which language and version you use, and illustrate how to compile, run and use your code.

3 Project Details

In this project, you should use SHA256 as your hash function. Assume a message m is stored in file "`../data/input.txt`", and assume this message is your UCID, for example

```
m123456
```

1. Target Generation Function:

(a) Given a POW difficulty d from the command line, compute a target t based on this POW difficulty d, where the first d bits in this target are all 0s and the rest of the bits in this target are 1s. Since we use SHA256 in this project, it indicates that given d, the first d bits are 0s and the rest of $256 - d$ bits in this target are 1s, e.g.,

$$t = \underbrace{00...00}_{d}\underbrace{11...11}_{256-d}$$

You also need to print this target in the terminal, and write this target to file "`../data/target.txt`". There is no requirement in terms of the format of this target as long as it is consistent with your read function.

2. Solution Generation Function:

(a) Read a target t from file "`../data/target.txt`, and read an input m from file "`../data/input.txt`, find a solution s, s.t.,

$$\mathsf{Hash}(m||s) \leq t$$

print this solution, and write this solution to file "`../data/solution.txt`". Note that since everyone's input message (i.e., UCID) is different, the solution for each student might be different. You code should be generic, i.e., it should be able to find solutions for other inputs as well.

3. Verification Function:

(a) Read an input m from file "`../data/input.txt`, read a solution from file "`../data/solution.txt`", read a target t from file "`../data/target.txt`, output and print 1 if this solution is valid, i.e., $\mathsf{Hash}(m||s) \leq t$; otherwise, output and print 0.

4. (CS6058 Only) Performance of POW:

(a) Given an input m (i.e., your UCID) from file "`../data/input.txt`", generate 6 different solutions for 6 different difficulties. More specifically, generate a solution s_i for each difficulty i, where $21 \leq i \leq 26$. **All the 6 solutions should be different**. Record and compare the running time of finding each of those solutions, and include the solutions you found, the running time and comparison in your one-page report (in pdf). In your report, please describe details of your implementation, such as OS, language, crypto libraries, etc. You can use tables or figures to present the comparison. Please put this report under your project folder and submit it together with your code.

4 Evaluation

Your project will be evaluated in three aspects.

1. **Correctness of Functions (75%):** Your program should be able to correctly run all the functions described in this project. If for some reason, your code cannot be compiled but the logic of your code is correct, you will still get partial credits.
2. **Comments and Descriptions (15%):** Write comments and explain each function in your code, such as inputs, outputs, etc. You may need some of the functions in other projects. Detailed comments on each function can save your time in other projects. In addition, please clearly explain how to compile and run your code in `Readme.txt`. For graduate students, the scores of your report in Task 4 will be included in this aspect. Undergraduate students do not need to submit a report, since Task 4 is for graduate students only.
3. **Coding Style (10%):** A good coding style is always important, especially for large projects. Please keep each function simple, try to avoid long functions, and create multiple `.h` and `.cpp` files if needed. For example, it is not a good idea to put everything in the main function.

5 Examples

This section provides some examples, which can help you understand the functions of your project. If your code can provide the same functionalities, you can customize the number of arguments and the order of arguments, as long as you describe it clearly in the Readme file.

Example 1: The following command generates a target based on a difficulty

```
pow target d ../data/target.txt
```

where `pow` is the name of your executable file, `target` is the argument for target generation function, `d` is difficulty, and the result of this target will be written to file `../data/target.txt`. Note that, depending on where your executable file is, the paths of your input and output files might be different.

Example 2: The following command generates a solution based on an input and a target

```
pow sol ../data/input.txt ../data/target.txt ../data/solution.txt
```

where this function reads an input from file `../data/input.txt`, reads a target from file `../data/target.txt`, computes a solution and writes this solution to file `../data/solution.txt`.

Example 3: The following command verifies whether a solution is valid.

```
pow verify ../data/input.txt ../data/target.txt ../data/solution.txt
```

where this function reads an input from file `../data/input.txt`, reads a target from file `../data/target.txt`, reads a solution to file `../data/solution.txt`, and outputs either 1 (valid) or 0 (invalid) in terminal.