POW Project Report

How to Run:

1. Run the file “pow_m12804663/src/ ProofOfWork.py”
    a. This will run through the TargetGen, SolutionGenereation, Verify, and PerformanceCheck functions with default values and should output a combination of the pictures below
2. If you would like to pass inputs, there are four options:
    a. TargetGen
        i. ProofOfWork.py TargetGen
        ii. ProofOfWork.py TargetGen 20 ../data/target.txt
    b. SolutionGenereation
        i. ProofOfWork.py SolutionGenereation
        ii. ProofOfWork.py SolutionGenereation ../data/target.txt ../data/input.txt ../data/solution.txt
    c. Verify
        i. ProofOfWork.py Verify
        ii. ProofOfWork.py Verify ../data/target.txt ../data/input.txt ../data/solution.txt
    d. PerformanceCheck
        i. ProofOfWork.py PerformanceCheck
        ii. ProofOfWork.py PerformanceCheck ../data/input.txt

Description:

For this project, I decided to write the proof of work (POW) code in the latest version of Python within Visual Studio 2022 and worked in Windows OS. I used the latest hashlib library to implement the SHA-256 hash encryption.

TargetGen:

This function generates a target based on the POW difficulty that is passed in as a parameter along with the target file to write the target to.

I programmed this by creating a string of 0’s of d bits length and a string of 1’s of 256 – d bits length. Then I concatenated the two strings which produces the target.

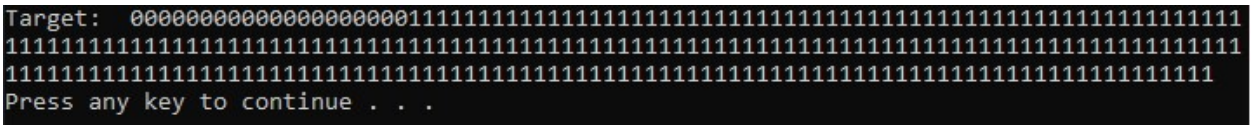

Figure 1: Difficulty = 20

SolutionGeneration:

This function generates a binary solution by iterating through every possible nonce value until a solution is found.

I first retrieved the input and target values from the two associated files. Then I iterated through all possible nonce values until a solution was found. To do this, I converted the nonce integer to binary and concatenated the input message string with the string version of the binary nonce value. Then I computed the hash value of the encoded combined string and retrieved the digest (or raw byte value) of the hash. Next, I converted the byte value to bits and then compared the target value with the retrieved bit value from the hash. If the bit value was less than or equal to the target, then it was a solution. Finally, I wrote the solution to a specified file and printed it.

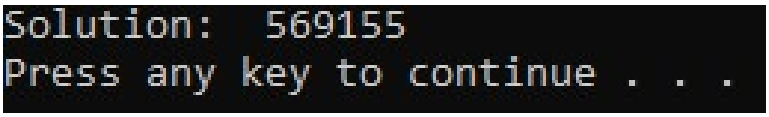

Figure 2: Solution of Figure 1 target given message = my UC ID

Verify:

This function verifies if a given solution is valid or not within the context of the target.

To accomplish this, I began by obtaining the input, target, and solution values from the three files respectively. Next, I converted the solution integer to binary and computed the concatenated hash value of the input message and the binary value. Then I converted the raw byte value of the hash to binary and checked if the binary value was less than or equal to the target value. If it was, then the function would print a 1, otherwise it would print a 0.

Figure 3: Verification of solution from Figure 2

PerformanceCheck:

This function iterates through difficulty levels 16 through 24 and produces a different solution for each level.

I started by looping through each difficulty level and calling the TargetGen function mentioned previously. Then I called SolutionGeneration, which returns the solution, and incremented this solution by a value of 1. This was so that each level would have a different solution. For example, the first difficulty level would start at 0 and end at solution n. Then the next difficulty level would start at n + 1 which would avoid looping through the same invalid solutions and repeating the same answer. Next, I calculated how much time had elapsed to find a solution for each difficulty and printed that value out. I also summed up the total time for all the difficulty values combined and printed that value. Finally, I printed the solution and difficulty levels.

```
Target:  00000000000000001111111111
11111111111111111111111111
Solution:  93540
Elapsed Time: 0.5370 seconds
Total Time:   0.5370 seconds
Difficulty: 16

Target:  00000000000000001111111111
11111111111111111111111111
Solution:  175315
Elapsed Time: 0.4920 seconds
Total Time:   1.0290 seconds
Difficulty: 17

Target:  00000000000000000111111111
11111111111111111111111111
Solution:  189541
Elapsed Time: 0.0850 seconds
Total Time:   1.1141 seconds
Difficulty: 18

Target:  00000000000000000011111111
11111111111111111111111111
Solution:  569155
Elapsed Time: 2.1420 seconds
Total Time:   3.2561 seconds
Difficulty: 19

Target:  00000000000000000001111111
11111111111111111111111111
Solution:  1714198
Elapsed Time: 6.5968 seconds
Total Time:   9.8529 seconds
Difficulty: 20

Target:  00000000000000000000111111
11111111111111111111111111
Solution:  4480101
Elapsed Time: 15.8030 seconds
Total Time:   25.6559 seconds
Difficulty: 21

Target:  00000000000000000000011111
11111111111111111111111111
Solution:  16721552
Elapsed Time: 71.8886 seconds
Total Time:   97.5445 seconds
Difficulty: 22

Target:  00000000000000000000001111
11111111111111111111111111
Solution:  36621279
Elapsed Time: 118.9679 seconds
Total Time:   216.5124 seconds
Difficulty: 23

Target:  00000000000000000000000111
11111111111111111111111111
Solution:  45732373
Elapsed Time: 55.2339 seconds
Total Time:   271.7462 seconds
Difficulty: 24
```

Figure 4: Difficulty levels 16 – 24 with time values