HashCash Lab

Moritz Walther

10.11.2020

The program HashCash_Emulation_Hashing.py was implemented as part of the Blockchain and Services for Fintech Enterprise Integration course. The purpose of the program is to simulate the Proof of Work algorithm implemented initially in HashCash, as well as the Bitcoin blockchain. The program builds a header based on various settings on the algorithm as well as inputs from the user. Salt is added with a random string. The header including the random string is then hashed with the SHA256 algorithm. The target difficulty is set by specifying the number of leading zeros which the hash must contain.

In order to implement the algorithm, I have elected to use Python. I used Python version 3.8.5. Python is simple, easy to read and has an excellent selection of modules which helped to implement the logic. The requirements specify that the nonce is to be encoded in Base64 format. Therefore, I have imported and used the Base64 module. The Python module only encodes Bytes and Strings so the integer is converted to a string after being incremented, and then Base64 encoding is performed. Basic modules such as String are imported to convert between types, and time is used in the code to measure the time it takes to find a suitable hash. Random is imported to generate a random String. Hashlib is a module which implements a host of different hashing algorithms and message digest tools. These include SHA256 and make it a very simple operation to generate a hash string.

In order to execute the program, the user simply needs to run it in their Python environment. They will be asked to input the number of leading zeros desired in the hash, thereby specifying the difficulty. From there the program executes all necessary steps. Depending on the difficulty specified the program may take some time to execute. When the program is finished it prints the hash it has found to the console. Furthermore, the number of tries needed as well as the time to finish are outputted.

As the number of leading zeros increases the difficulty increases exponentially. This is the central concept used in Proof of Work algorithms in blockchain networks. The difficulty is routinely adjusted so that on average even with the computing power of the whole network it should take approximately 10 minutes to solve the hash puzzle. This exponential growth in difficulty is demonstrated in the program implemented. Various tests where made. The results can be found in the table and chart below. Using my hardware and due to time constraints, it was only feasible to test the program with up to six leading zeros.

Table 1: Hashing trials and results

Difficulty (leading zeros)	Time (s)	Attempts needed
1	0,0030491352081299	4
1	0,0019986629486084	4
1	0,0020010471343994	4
1	0,0060026645660400	13
1	0,0039582252502441	11
2	0,0543270111083984	245
2	0,0979599952697753	317
2	0,0229141712188720	154
2	0,0800988674163818	421

2	0,0439729690551757	289
3	2,0289454460144000	12506
3	0,6263246536254880	4610
3	0,3692271709442130	2531
3	0,6900453567504880	4738
3	0,2170436382293700	1201
4	2,7653253078460600	18721
4	26,4775381088256000	190642
4	14,7091569900512000	82613
4	41,0826761722564000	179926
4	2,4491243362426700	13225
5	199,9337203502650000	1241627
5	40,6427655220031000	273796
5	100,1797327995300000	687675
5	39,0582163333892000	188202
5	22,2792778015136000	135308

Figure 1: Difficulty (Time) Needed to Solve Hash Puzzle

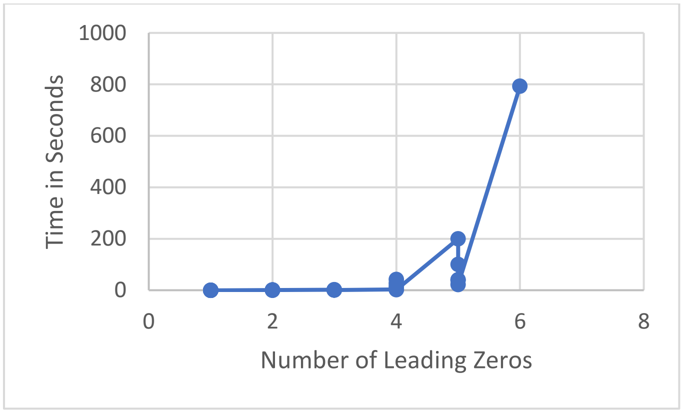
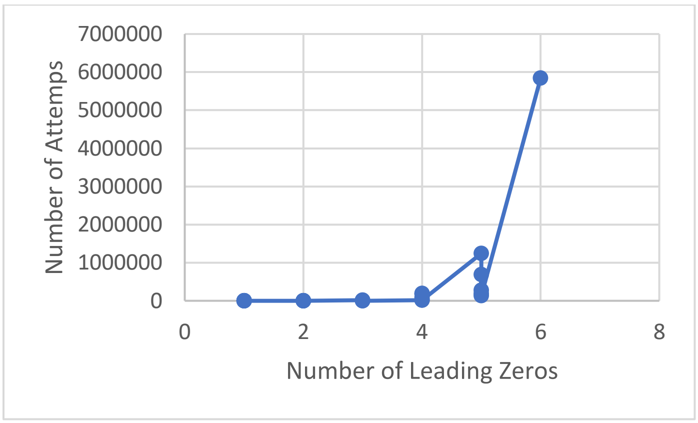


Figure 2: Difficulty (Loops) Needed to Solve Hash Puzzle


Number of Attemps
Number of Leading Zeros