Trevor Parsons

Project 3 Documentation

Task 1

I decided to use the MD5 hashing algorithm included in Java Security for comparison against SHA-256. The “Short File” that was used was 16 random bytes long, the “Long File” was 1MB of random bytes, and the “Cycles” output is the number of times that the file contents were successfully hashed. Here is some sample program output from Task 1:

SHA-256 Short File Cycles: 5928

Hash Value: 232f0b05af2584f92ef9879114492b52b7c798ec9e8212afa1e473a5209e1833

SHA-256 Long File Cycles: 136

Hash Value: ba816a9835966b8cf4d1be48dcffdc50b0144121b099d1caec688e88f74e6f27

MD5 Short File Cycles: 6465

Hash Value: d8db462ebd827130da9b2b3dbc31e0cd

MD5 Long File Cycles: 302

Hash Value: 95129a559cfd5556bdecec2b20707e10

-Note: My Laptop that made these calculations has an Intel Core i7-4702 MQ CPU @ 2.20 GHZ and 12 GB of RAM. I used the Java Security class for all cryptographic functions in each task.

As expected, the number of short file hashes was much higher than for the longer file, and MD5 completed more cycles per second (as it has an output size of 128-bits). Using the Birthday Paradox, I can calculate that for MD5, 2^(128/2) attempts on average will be needed to find a collision, and 2^64 = 1.84e19. Using the cycles/second from the short file, this means that 1.84e19 / 6465 = 2.8e15 seconds which is equal to 90,478,332 years for my computer to find a collision.

For SHA-256, 2^128 attempts will be needed on average, and 2^128 = 3.4\*10^38. Using the cycles/second from the short file, 3.4\*10^38 / 5928 = 5.7\*10^34 seconds which is equal to 1.8\*10^27 years for my computer to find a collision.

Task 2

I was able to find a hash that began with the exact digits of my birthday (5191992):

Found 5, Time Required: 0.003 seconds.

5727a68fc696e3beb655b29e78dbb53f8da150162c11e98ab3ee5712e2e4db0

Found 51, Time Required: 0.006 seconds.

51c75d51daa4d68a927573e8ae347bdf903e8ba5d9d89a2fc2403705774e7c43

Found 519, Time Required: 0.035 seconds.

519f9d2ee88ce73b1376bcc35c02bc177a214832f0d90ba858c71ac3cfdb642e

Found 5191, Time Required: 0.173 seconds.

5191639094654db65d73fe489fe9cdbba3362cf974a78bef13c63cb8fe8203ff

Found 51919, Time Required: 0.948 seconds.

51919353d50b125db17de738ddbed8af9bf3e67feaabcf97ae5311506905d3e6

Found 519199, Time Required: 32.794 seconds.

5191994c5fa97017194436555064558e0832c1e201a6d5444589a0450f93c607

Found 5191992, Time Required: 533.443 seconds

519199288d49350c22b25e9efca13d7a89dc28ae2831d8f4fe7b1b28a10e22f

Total time: 533.443 seconds.

I started with a byte array of all zeros, and then began randomizing the bytes in the array. Because the bytes were randomized, I got variable results for how long it took to find hash values that began with the proper strings.

In order to find the average time required to find a hash that has all of the digits of my birthday, I would have to run this program many times and calculate the time average for all of the runs. One thing that I found interesting was that when first tried to find my birthday digits in the ‘0191992’ format but was unable to find any hash values that began with ‘0’, so I removed the ‘0’ from the string and was able to find the desired hashes. I’m not sure if this is due to the nature of the output for SHA-256 or if the hash function is not correctly implemented.

Task 3

Sample program output for Task 3:

Number of hashes in 10 seconds: 2349997

Lowest value found: 3.762712747153584E70

Number of hashes in 20 seconds: 4968214

Lowest value found: 1.1666827661863277E70

Number of hashes in 30 seconds: 7431602

Lowest value found: 7.492206852134187E69

After running program three, the lowest value found was much higher than I expected, but I could not find any information about the maximum and minimum hex values that are produced by the SHA-256 algorithm to verify it. For the random string, I used a random UUID value and converted it to string to be used in each hash iteration.

The only concrete thing that I can see for creating a formula to determine the lowest expected value for X seconds is that is that the larger X is, the lower the value that will be found is (generally speaking). However, I assume that as X gets very large, the rate at which the lowest expected value decreases also decreases, so there is probably some exponential relationship between time and lowest value found, such as something like:

Expected Lowest Value = (Some Constant) / (Seconds \* Number of cycles)^(some power)