SHA256 Hashes

Mark Nakamae

Cal Poly SLO

June 2020

Description

The SHA2 (Secure Hash Algorithm 2) is a fairly well-known hashing algorithm described [here](https://en.wikipedia.org/wiki/SHA-2). This implementation focuses on an implementation that uses 8-bit words to 50-bit words. The upper bound of 50-bits was arbitrarily chosen, as implementing the full 256-bit iteration would have taken too long to compute.

This project specifically looked at collisions using the PyCrypto library. A word was randomly generated, hashed, and entered into a Python dictionary. Then, the program continued randomly generating words, hashing them, and comparing them to the original dictionary entry to see if a collision occurred. Generally speaking, larger bit-words had longer runtimes, since they took longer to find a collision in a dictionary entry from the hash. This also correlated to more dictionary entries as well. The data was then entered into an Excel Spreadsheet using the xlwt library.

Explore Pseudo-Randomness and Collision Resistance

SHA256 Program Hamming Distance of 1 Results

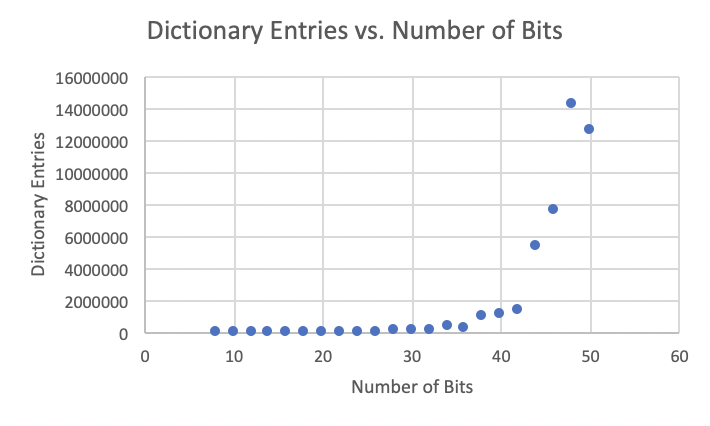
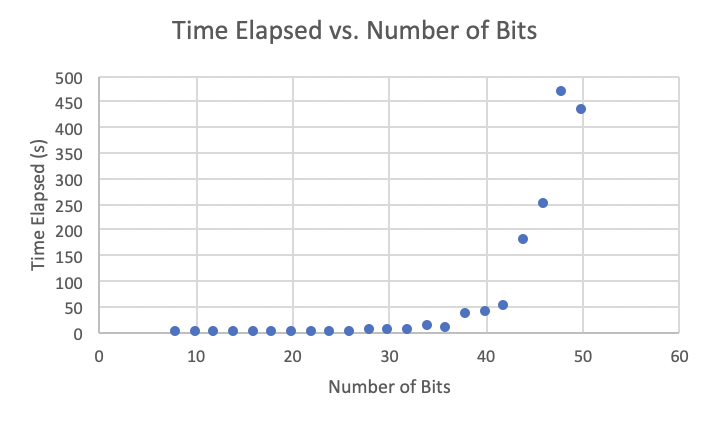


*What do you observe? How many of the bytes are different between the two digests?*

We found no distinguishable pattern between hashes of strings with a hamming distance of 1. In fact, almost all bytes were different between the digests as shown in the examples above.

Collisions Program Results

|  |  |  |
| --- | --- | --- |
| **Number of Bits** | **Time Elapsed** | **Dictionary Entries** |
| 8 | 0.001225948 | 35 |
| 10 | 0.002065897 | 62 |
| 12 | 0.001941919 | 59 |
| 14 | 0.012593985 | 378 |
| 16 | 0.017798185 | 550 |
| 18 | 0.009170294 | 263 |
| 20 | 0.014846087 | 453 |
| 22 | 0.092112064 | 2733 |
| 24 | 0.17911911 | 5126 |
| 26 | 0.289658785 | 8711 |
| 28 | 1.380840063 | 42318 |
| 30 | 1.074311972 | 33039 |
| 32 | 2.832461119 | 87273 |
| 34 | 8.679316998 | 268090 |
| 36 | 7.989431143 | 246255 |
| 38 | 34.14928102 | 961785 |
| 40 | 38.87270308 | 1070557 |
| 42 | 50.12440801 | 1292382 |
| 44 | 179.4916022 | 5345491 |
| 46 | 248.4463518 | 7576598 |
| 48 | 468.7364092 | 14140602 |
| 50 | 431.2411809 | 12540366 |

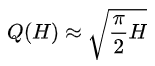


*What is the maximum number of files you would ever need to hash to find a collision on an n-bit digest?*

The maximum number occurs when there is a collision on the final input, so the maximum number of files is also the number of inputs in an n-bit hash, which is **2n + 1**.

*Given the Birthday Bound, what is the expected number of hashes before a collision on an n-bit digest? Is this what you observed?*

Using this equation from Wikipedia:

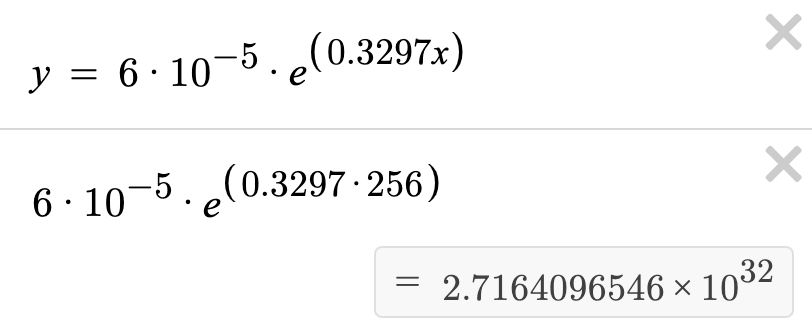


Where Q(H) represents the expected number of hashes before a collision. Thus, the expected number of hashes before a collision for an n-bit digest is approximately:

Q(2n) =

*Given the data you’ve collected, speculate on how long it might take to find a collision on the full 256-bit digest.*

According to the data collected, the best fit line was an exponential curve with the following equation, where *x* represents the number of bits in the hash:



From this equation, one could conclude that a 256-bit digest would take approximately 2.716 \* 1032 seconds to find a collision. Granted, this number was found from an excel best fit curve, but regardless this task would still take a considerably long time.

*Given an 8-bit digest, would you be able to break the one-way property (i.e. can you find* any *pre-image)? Do you think this would be easier or harder (i.e. more or less work) than finding a collision? Why or why not?*

Since these modern hashes are for the most part preimage attack resistant, it would be much easier to find a collision for an 8-bit digest, rather than try to crack the one-way property of a hash.

Code

**from** Crypto.Hash **import** SHA256

**import** time

**import** random

**import** string

**import** xlwt

**from** xlwt **import** Workbook

#compute hash for input string

**def** get\_hash(string):

h = SHA256.new()

h.update(string.encode('utf-8'))

**return** h.hexdigest()

#compute an n-bit hash for input string

**def** get\_hash\_truncated(string, num\_bits):

hashed = str(get\_hash(string))

trunc = "{0:256b}".format(int(hashed, 16))

**return** trunc[:num\_bits]

#test hash differences for hamming distance of 1

**def** test\_hamming():

alpha = "abcdefghijklmnopqrstuvwxyz"

**for** ch **in** alpha:

tmp = "abcd" + ch

**print**("---- " + tmp + " ----")

**print**("0x" + get\_hash(tmp))

**print**()

#produce random string to be used for test cases

**def** get\_random\_string():

alpha = "abcdefghijklmnopqrstuvwxyz"

**return** "".join(random.choice(alpha) **for** i **in** range(20))

#finds a collision for a specified input string hashed with n-bits

#and writes the outputs to an excel file

**def** collisions(m0, num\_bits, sheet):

hashes = {get\_hash\_truncated(m0, num\_bits) : m0}

start\_time = time.time()

**while**(True):

m1 = get\_random\_string()

#if collision is found and strings are not the same, exit loop

**if**(get\_hash\_truncated(m1, num\_bits) **in** hashes **and**

hashes[get\_hash\_truncated(m1, num\_bits)] != m0):

**break**

#collision is not found, add hashed value to dictionary

**else**:

hashes[get\_hash\_truncated(m1, num\_bits)] = m1

#output to terminal

**print**(" \* \* \* \* \* \* \* \* COLLISION FOUND \* \* \* \* \* \* \* \*")

**print**("Hash was truncated to " + str(num\_bits) + " bits")

**print**("Original Message (m0): " + m0)

**print**("Computed Message (m1): " + m1)

**print**("Dictionary Entries: " + str(len(hashes)))

**print**("Time Elapsed: " + str(time.time() - start\_time) + " seconds")

#output to excel sheet

sheet.write(int(((num\_bits - 8) / 2) + 1), 0, num\_bits)

sheet.write(int(((num\_bits - 8) / 2) + 1), 1, time.time() - start\_time)

sheet.write(int(((num\_bits - 8) / 2) + 1), 2, len(hashes))

#find collisions for 8, 10, 12, ..., 50 bit hashes

**def** test\_collisions():

wb = Workbook()

sheet1 = wb.add\_sheet("collisions.py")

sheet1.write(0, 0, "Number of Bits")

sheet1.write(0, 1, "Time Elapsed")

sheet1.write(0, 2, "Dictionary Entries")

**for** bits **in** range(8, 52, 2):

collisions(get\_random\_string(), bits, sheet1)

**print**()

wb.save('asgn3.xls')

**def** main():

test\_hamming()

test\_collisions()

**if** \_\_name\_\_ == '\_\_main\_\_':

main()