Hash Attack

September 21, 2023

Gage Moore

Introduction

Hash functions are a fundamental component of cryptography, used to convert arbitrary data into fixed-length hash values. The strength of hash functions lies in their ability to produce unique hash values for different inputs, thereby ensuring data security.

In this report, the results of an analysis will be explored in which a series of preimage and collision attacks were run on preimages of varying bit sizes in order to assess the difficulty of finding hash collisions.

Important Terms

Preimage The input data to a hash function

Digest The output data of a hash function, also known as a hashcode or hash value

Preimage Attack An attack in which an adversary, when given a preimage, finds another preimage that hashes to the same digest

Collision Attack An attack in which an adversary successfully finds two arbitrary preimages that hash to the same digest

SHA1 Secure Hash Algorithm 1 developed by the NSA that has since been proven to be cryptographically insecure

Procedure

Using Python, hash attacks were performed by hashing random data using SHA1 and truncated the digest to bit sizes of 8, 10, 16, 20, and 24. At each bit size, both attacks were run 50 times with the number of attempts until a collision recorded for each of the 50 rounds.

Results

The tables below include 50 numbers for each attack, each number being the number of attempts, (hashes), until a collision was found.

Data for a bit size of 8:

Attack Type	Attempts Until Collision	Stats
Preimage	54, 39, 285, 200, 200, 383, 74, 14, 94, 127, 141, 36, 636, 367, 219, 50, 248, 33, 115, 12, 401, 100, 61, 15, 41, 31, 1, 241, 175, 63, 198, 11, 178, 96, 14, 3, 15, 101, 88, 95, 33, 96, 328, 153, 80, 72, 28, 164, 153, 400	Mean: 135.24 Median: 95.5 Range: 635
Collision	8, 12, 4, 8, 5, 16, 25, 18, 5, 10, 21, 16, 19, 10, 32, 12, 20, 17, 11, 21, 12, 10, 11, 12, 9, 20, 13, 6, 31, 15, 21, 21, 10, 15, 15, 7, 25, 15, 13, 10, 24, 22, 9, 9, 17, 4, 20, 15, 19, 13	Mean: 14.66 Median: 14.0 Range: 28

Data for a bit size of 10:

Attack Type	Attempts Until Collision	Stats

Preimage	72, 112, 206, 57, 86, 328, 151, 63, 509, 207, 1736, 132, 847, 134, 1904, 1107, 215, 348, 165, 2964, 1195, 555, 169, 37, 412, 1028, 656, 38, 434, 562, 38, 114, 238, 1118, 532, 1286, 500, 252, 344, 47, 1121, 540, 1, 367, 1568, 979, 1276, 701, 46, 470	Mean: 559.34 Median: 357.5 Range: 2963
Collision	28, 67, 9, 34, 14, 42, 17, 11, 10, 20, 29, 23, 31, 31, 30, 23, 22, 19, 32, 17, 16, 21, 23, 26, 35, 20, 16, 29, 35, 33, 62, 32, 42, 29, 38, 12, 52, 44, 31, 23, 28, 12, 44, 9, 18, 20, 13, 36, 37, 51	Mean: 27.92 Median: 28.0 Range: 58

Data for a bit size of 16:

Attack Type	Attempts Until Collision	Stats
Preimage	64906, 12286, 253, 10900, 1836, 47109, 14236, 15770, 120741, 1699, 55371, 20156, 2063, 17808, 75379, 55361, 59140, 6263, 12287, 4127, 33492, 81048, 87616, 39154, 2861, 86133, 3423, 14408, 2139, 12391, 2920, 123241, 24308, 17681, 55253, 16924, 50840, 17, 88185, 2216, 4013, 23000, 17896, 5758, 18298, 55769, 109996, 23545, 26552, 31472	Mean: 33164.8 Median: 18097.0 Range: 123224
Collision	201, 157, 448, 263, 231, 246, 289, 320, 320, 174, 278, 70, 300, 205, 228, 556, 405, 120, 269, 375, 304, 294, 86, 358, 50, 278, 274, 378, 236, 151, 328, 247, 374, 63, 198, 90, 258, 106, 244, 87, 36, 138, 130, 81, 67, 456, 83, 80, 184, 72	Mean: 223.72 Median: 233.5 Range: 520

Data for a bit size of 20:

Attack Type	Attempts Until Collision	Stats
Preimage	998923, 916866, 311620, 62994, 102889, 858807, 201917, 380431, 302109, 315519, 245229, 1267886, 334548, 500034, 359748, 122906, 1430222, 430141, 528947, 223262, 166062, 455638, 868592, 112404, 690592, 579140, 32447, 520915, 1473113, 114947, 153277, 28006, 459845, 1112139, 56454, 161721, 1170519, 295049, 240647, 146065, 141815, 117431, 15621, 12340, 80269, 46085, 249752, 503666, 1102817, 614028	Mean: 432327.88 Median: 306864.5 Range: 1460773
Collision	857, 700, 931, 708, 1026, 831, 1165, 950, 707, 1457, 515, 1061, 1095, 521, 797, 601, 934, 2353, 872, 447, 543, 1506, 1298, 1493, 1024, 1202, 556, 738, 1346, 576, 415, 910, 1919, 773, 979, 640, 972, 932, 657, 1008, 779, 1924, 432, 1337, 1089, 677, 1139, 1305, 1141, 693	Mean: 970.62 Median: 931.5 Range: 1938

Data for a bit size of 24:

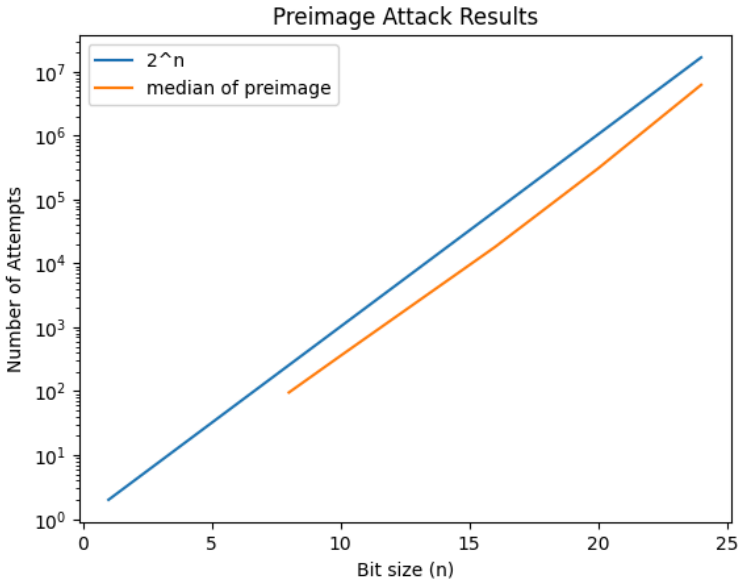
Attack Type	Attempts Until Collision	Stats

Preimage	5263909, 4143900, 22589464, 585584, 5562489, 9442398, 1897920, 8493881, 24979750, 12884427, 7736695, 2068490, 1288291, 6918793, 5424432, 308389, 17151239, 7115204, 7626089, 3412268, 10969104, 9928696, 4857229, 7053243, 21019422, 1689457, 13136241, 130981, 5079540, 7444729, 1597646, 9378265, 12047138, 695567, 23528063, 5548871, 7813092, 5712, 20987915, 7327755, 5012254, 3670830, 4955751, 3724768, 1403200, 23875605, 9454267, 45798255, 382319, 506729	Mean: 8478325.12 Median: 6240641.0 Range: 45792543
Collision	1780, 4308, 2868, 3676, 2321, 3371, 731, 4222, 3985, 4108, 1232, 2142, 3343, 3764, 6224, 2215, 1278, 2575, 5142, 1925, 4196, 2882, 4564, 4544, 2582, 1383, 3784, 2291, 2633, 3420, 6668, 5300, 6581, 3457, 4165, 2827, 7863, 1388, 374, 3866, 2985, 3333, 2775, 1177, 5383, 2479, 1055, 2218, 3334, 3146	Mean: 3277.26 Median: 3239.5 Range: 7489

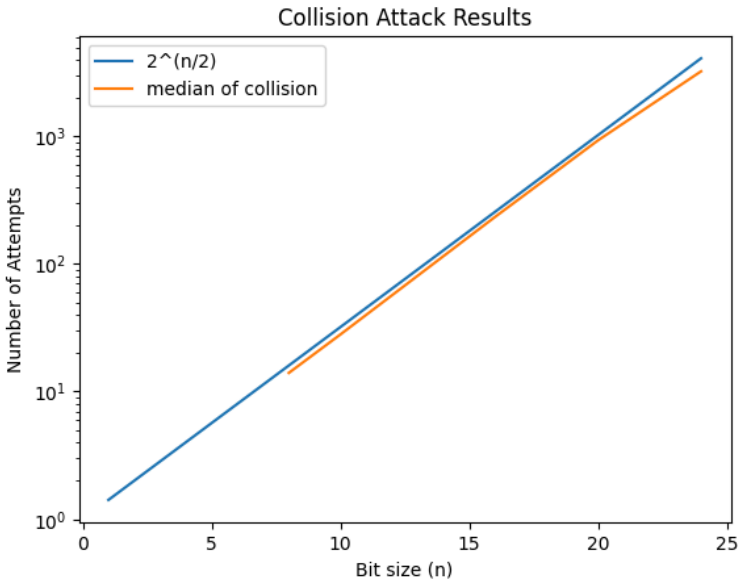
Analysis

The theoretical time that a preimage attack is expected to take is represented as 2^n where n is the number of bits of the digest. Similarily, the theoretical time that a collision attack is expected to take is represented as $2^{\frac{n}{2}}$. The difference between the efficiency of the two attacks lies in the nature of how collision attacks complete when any match is found whereas preimage attacks are grounded to a specific preimage.

When graphing the medians from the results above over increasing bit sizes, we can visually determine how closely the procedural results follow the theoretical results.



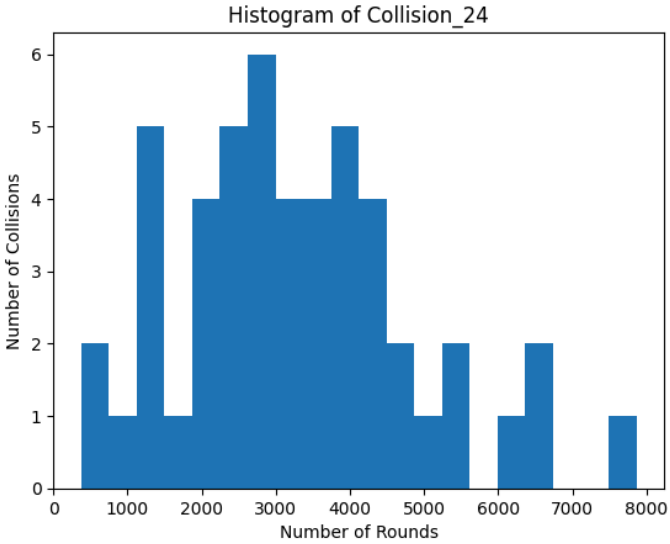
The plot above compares 2^n, (blue), with the median of our preimage results, (orange). Notably, both lines are mostly parallel and the results on the y axis run close to one another. This indicates that our procedural results did not stray very far from what was expected.



The plot above compares $2^{\frac{n}{2}}$, (blue), with the median of our collision results, (orange). Notably, both lines are mostly parallel and the results on the y axis run very close to one another. As before, this indicates that our procedural results did not stray very far from what was expected.

Note on Variance

The ranges from the data that was collected were often very large. There were a number of very high and very low outliers included in each dataset which is what compelled me to use medians, rather than means, to evaluate the effectiveness of the procedure. These outliers can be attributed to computational issues that result from running huge, long-running experiments on highly randomized data. As an example of such outliers, below is a histogram of the results for collision attacks on bit sizes of 24. Though upper outliers exist, the data congregates on a normal curve around the median.



Python Script Implementation

The script I used to calculate these results is included below for reference.

```
from hashlib import sha1
from secrets import token_hex


def truncate(string, n):
    binary = bin(int(string, 16))[2:]
    truncated = binary[:n]
    return hex(int(truncated, 2))[2:]


def hash(string, n):
    hash = sha1(string.encode()).hexdigest()
    return truncate(hash, n)


def generate_random_string():
    return token_hex(32)


def preimage_test(n):
    string_hash = hash(generate_random_string(), n)
    i = 0
    while True:
        test_hash = hash(generate_random_string(), n)
        if string_hash == test_hash:
            break

        i += 1

    return i


def collision_test(n):
    i = 0
    hashes = []

    while True:
        test_hash = hash(generate_random_string(), n)
        if test_hash in hashes:
            break

        hashes.append(test_hash)
        i += 1

    return i


'''
Entry point for the program.
Conducts preimage and collision attacks 50 times for each given bit size.
Prints the results of each attack.
'''
def main():
```

```
    bit_sizes = [8, 10, 16, 20, 24]

    for n in bit_sizes:
        preimage_results = []
        collision_results = []

        for _ in range(50):
            preimage_results.append(preimage_test(n))
            collision_results.append(collision_test(n))

        print('------------------')
        print("Results for n = " + str(n) + ":")
        print("preimage results: " + str(preimage_results))
        print("collision results: " + str(collision_results))

        print()

if __name__ == '__main__':
    main()
```