ECE: Information and Network Security Computer Project

Zhongze Tang, Zichen Zhu, Weijia Sun, and Zhuoran Qi

April 17, 2018

Abstract

The Secure Hash Algorithm 256 (SHA-256) is one of the strongest hash functions available. In order to study it in more details, we implement a hash-based crypto puzzle using Secure Hash Algorithm 256 (SHA-256) and examine the complexity of the crypto puzzle. In our project, timing measurements will be taken in order to estimate the amount of time it takes to solve the hash-based crypto puzzle.

1 Introduction

1.1 History

Developed by National Institute of Standards and Technology (NIST) in association with the National Security Agency (NSA), the Secure Hash Algorithm (SHA) was first published as the Secure Hash Standard in May 1993[1]. And in 1995, SHA-1 was published as the first revision to this algorithm, while SHA-0, the first version was withdrawn by the NSA. While the initial SHA function which is easily attacked, cryptanalysis on SHA-1 proved to be much more difficult. Besides, the NIST also released a series of more complex hash functions for which the output ranged from 224 bits to 512 bits, namely SHA-224, SHA-256, SHA-384 and SHA-512 (sometimes referred to as SHA-2). These hash algorithms proved to be more complex with non-linear functions added to the compression function, which leads to longer hashes and makes a feasible attack more difficult.

1.2 SHA-256 Algorithm

SHA-2 is a common name for four additional hash functions also referred to as SHA- 224, SHA-256, SHA-384 and SHA-512. Their suffix originates from the bit length of the message digest they produce. SHA-256 uses a block size of 512 bits, and iterates 64 rounds. SHA-256 uses sixty-four constants K^0, .., K^{63} of 32 bits each and eight registers to store intermediate results H_0, .., H_7. The values, which can be found in [2], are:

Table 1
SHA-256 Constants

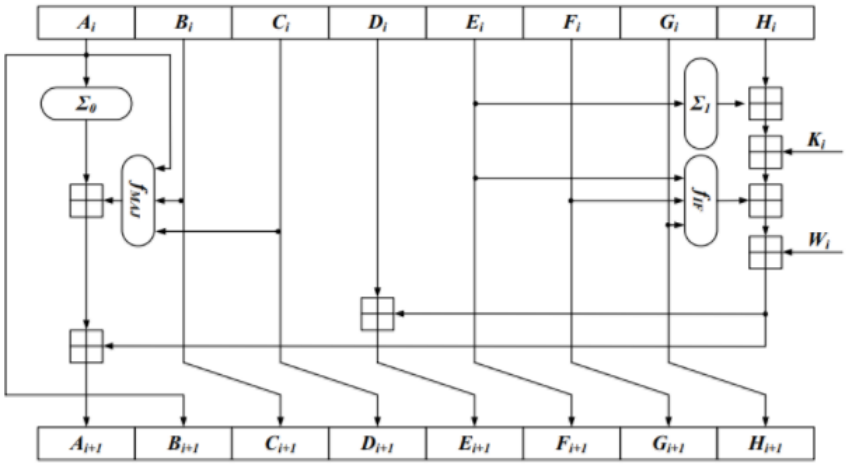
428a2f98	71374491	b5c0fbcf	e9b5dba5	3956c25b	59f111f1	923f82a4	ab1c5ed5
d807aa98	12835b01	243185be	550c7dc3	72be5d74	80deb1fe	9bdc06a7	c19bf174
e49b69c1	efbe4786	0fc19dc6	240ca1cc	2de92c6f	4a7484aa	5cb0a9dc	76f988da
983e5152	a831c66d	b00327c8	bf597fc7	c6e00bf3	d5a79147	06ca6351	14292967
27b70a85	2e1b2138	4d2c6dfc	53380d13	650a7354	766a0abb	81c2c92e	92722c85
a2bfe8a1	a81a664b	c24b8b70	c76c51a3	d192e819	d6990624	f40e3585	106aa070
19a4c116	1e376c08	2748774c	34b0bcb5	391c0cb3	4ed8aa4a	5b9cca4f	682e6ff3
748f82ee	78a5636f	84c87814	8cc70208	90befffa	a4506ceb	bef9a3f7	c67178f2

Table 2
The initial hash value, $H^{(0)}$

$H_0^{(0)}$	6a09e667
$H_1^{(0)}$	bb67ae85
$H_2^{(0)}$	3c6ef372
$H_3^{(0)}$	a54ff53a
$H_4^{(0)}$	510e527f
$H_5^{(0)}$	9b05688c
$H_6^{(0)}$	1f83d9ab
$H_7^{(0)}$	5be0cd19

Below is the schematic overview of a 0 SHA-2 round:

Fig. 1



The function definitions for SHA-256 are:

$$W_i = \left\{ \begin{array}{ll} M_i, & if \quad 0 \leq i \leq 15 \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16}, & if \quad 16 \leq i \leq 63 \end{array} \right\}$$

with
$\Sigma_0(x) = x \ggg 2 \oplus x \ggg 13 \oplus x \ggg 22$
$\Sigma_1(x) = x \ggg 6 \oplus x \ggg 11 \oplus x \ggg 25$
$\sigma_0(x) = x \ggg 7 \oplus x \ggg 18 \oplus x \ggg 3$
$\sigma_1(x) = x \ggg 17 \oplus x \ggg 19 \oplus x \ggg 20$
and
$f_{if}(b, c, d) = b \wedge c \oplus \neg b \wedge d$
$f_{maj}(b, c, d) = b \wedge c \oplus b \wedge d \oplus c \wedge d$
SHA-256 algorithm[1]:
SHA-256(M):
(*Let M be the message to be hashed *)
for each 512-bit block B in M **do**
(*Initialize the registers with the constants. *)
$a = H_0; b = H_1; c = H_2; d = H_3; e = H_4; f = H_5; g = H_6; h = H_7;$
for $i = 0$ to 63 **do**
(* Apply the 64 rounds of mixing. *)
$T_1 = h + \Sigma_1(e) + f_{if}(e, f, g) + K_i + W_i;$
$T_2 = \Sigma_0(a) + f_{maj}(a, b, c);$
$h = g; g = f; f = e; e = d + T_1; d = c; c = b; b = a; a = T_1 + T_2;$
(*After all the rounds, save the values in preparation of the next data block. *)
$H_0 = a + H_0; H_1 = b + H_1; H_2 = c + H_2; H_3 = d + H_3;$
$H_4 = e + H_4; H_5 = f + H_5; H_6 = g + H_6; H_7 = h + H_7;$
(*After all 512-bit blocks have been processed, return the hash. *)
return concat$(H_0, H_1, H_2, H_3, H_4, H_5, H_6, H_7)$;

1.3 Hash Collision

1.3.1 Birthday attack

Birthday attack relies on the higher likelihood of collisions which are found between random attack attempts as well as a fixed degree of permutations. This type of cryptographic attack can find a collision of a hash function in $\sqrt{2^n} = 2^{n/2}$, with 2^n being the classical preimage resistance security[3]. Given a set of H values, we choose n values uniformly at random thereby allowing repetitions. Define $p(n; H)$ as the probability that at least one value is chosen more than once during this experiment.

$p(n; H) \approx 1 - e^{-n(n-1)/(2H)} \approx 1 - e^{-n^2/(2H)}$,

Define $n(p; H)$ as the smallest number of values we need to choose, so that the probability of finding a collision is at least p.

$n(p; H) \approx \sqrt{2H \ln \frac{1}{1-p}}$,

Define $Q(H)$ as the expected number of values we need to choose before finding the first collision.

$Q(H) \approx \sqrt{\frac{\pi}{2}H}$.

Examples of birthday attack are as follows:

Table 3

Bits	Possible outputs (H)	Desired probability of random collision $(2s.f.)(p)$				
		10^{-18}	10^{-15}	10^{-12}	10^{-9}	10^{-6}
16	$2^{16}(\sim 6.5\times10^{3})$	<2	<2	<2	<2	<2
32	$2^{32}(\sim 4.3\times10^{9})$	<2	<2	<2	3	93
64	$2^{64}(\sim 1.8\times10^{19})$	6	190	6100	190,000	6,100,000
128	$2^{128}(\sim 3.4\times10^{38})$	2.6×10^{10}	8.2×10^{11}	2.6×10^{13}	8.2×10^{14}	2.6×10^{16}
256	$2^{256}(\sim 1.2\times10^{77})$	4.8×10^{29}	1.5×10^{31}	4.8×10^{31}	1.5×10^{34}	4.8×10^{35}
384	$2^{384}(\sim 3.9\times10^{115})$	8.9×10^{48}	2.8×10^{50}	8.9×10^{51}	2.8×10^{53}	8.9×10^{54}
512	$2^{512}(\sim 1.3\times10^{154})$	1.6×10^{68}	5.2×10^{69}	1.6×10^{71}	5.2×10^{72}	1.6×10^{74}

Bits	Possible outputs (H)	Desired probability of random collision $(2s.f.)(p)$				
		0.1%	1%	25%	50%	75%
16	$2^{16}(\sim 6.5\times10^{3})$	11	36	190	300	430
32	$2^{32}(\sim 4.3\times10^{9})$	2900	9300	50,000	77,000	110,000
64	$2^{64}(\sim 1.8\times10^{19})$	1.9×10^{8}	6.1×10^{8}	3.3×10^{9}	5.1×10^{9}	7.2×10^{9}
128	$2^{128}(\sim 3.4\times10^{38})$	8.3×10^{17}	2.6×10^{18}	1.4×10^{19}	2.2×10^{19}	3.1×10^{19}
256	$2^{256}(\sim 1.2\times10^{77})$	1.5×10^{37}	4.8×10^{37}	2.6×10^{38}	4.0×10^{38}	5.7×10^{38}
384	$2^{384}(\sim 3.9\times10^{115})$	2.8×10^{56}	8.9×10^{56}	4.8×10^{57}	7.4×10^{57}	1.0×10^{58}
512	$2^{512}(\sim 1.3\times10^{154})$	5.2×10^{75}	1.6×10^{76}	8.8×10^{76}	1.4×10^{77}	1.9×10^{77}

1.3.2 Differential cryptanalysis

Differential cryptanalysis is a general form of chosen plaintext attack, which needs attackers to have access to obtain ciphertexts for some set of plaintexts of their choosing[5]. The basic strategy of differential cryptanalysis is using pairs of plaintext related by a constant difference and then computing the differences of the corresponding ciphertext, hoping to detect statistical patterns in the distribution.

Essentially, for an n-bit nonlinear function, attackers need to ideally seek as close to 2^{n-1} as he can to achieve differential uniformity, when which happens, the differential attack requires as much work as simply brute forcing the key in order to determine the key.

1.4 Project Description

Our task is to implement a hash-based crypto puzzle using SHA-256. The way the crypto puzzle works is as follows. Alice issues a challenge to Bob: Alice creates a puzzle P that is B-bytes long, where $B < 32$. She challenges Bob to find a message M that, when fed into the SHA-256 function $h()$ yields the last B bytes equal to P. That is, $h(M) = [***\cdots***, P]$. Bob attempts to find such an M by generating random guesses and hashing them. When he finds one, he reports it to Alice and has solved the crypto puzzle.

2 Puzzle Solution

2.1 Basic Principle

For each B, we generate a B-bytes puzzle P at first, and then generate a random message M and calculate it's SHA-256(M). If the last B bytes of the SHA-256(M) equal to P, then we find the result

M. If not, re-generate an M until finding the result. Record the number of trials at the same time. Repeat this 1000 times and calculate the average number of trials.

Each P and M are generated Byte-by-Byte, and each Byte is generated by *Random().nextBytes()* function in *java.util.Random* package. For P, its length B is $1, 2, 3, \ldots$, until 31 Bytes due to the requirement of the project. And for M, its length is 56 Bytes. In the first step of SHA-256 algorithm, a 1 and several 0s will be added to the end of the message to make the length of $M = 448(mod512)$. So a 56-Bytes-long message M will guarantee the collision.

We use JAVA and the built-in Java.security package to implement this program. To speed up, we also use multithreading in our program. We deploy our program on the AWS c5.18xlarge server, which has 72 vCPU (which means it has 72 logical threads) and costs $3.06 per hour.

3 Result and Analysis

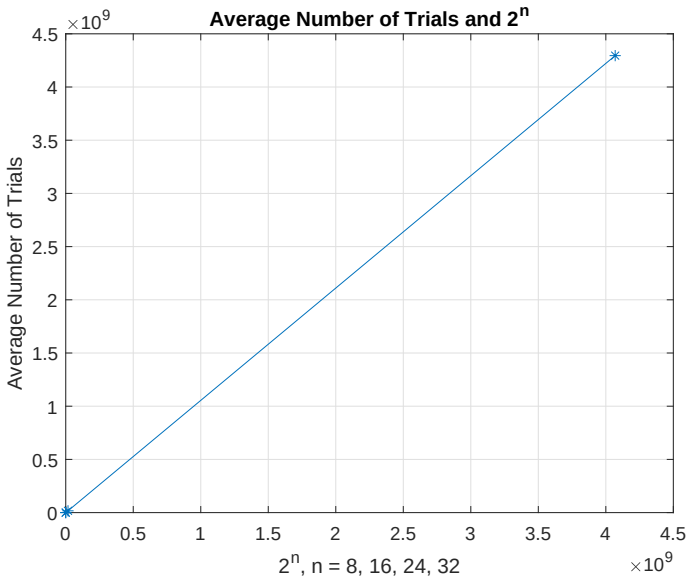
Table 4

B* (Bytes)	1	2	3	4
Ave time (s)	0.015	0.114	20.601	5255.7448
Ave No. of trials	253.898	69080.71	16887421.35	4069363775
$2^n (n = 8 \times B)$	256	65536	16777216	4294967296

*: Because we worry about our AWS bill, we only run the program with $B = 4$ seventy-two times. For $B = 1, 2$ and 3, we run the program 1000 times. When $B > 4$, it will take too much time to find a solution, so we give up the trial for $B > 4$ to save time and money.

According to the Table 4 and the Fig. 2, we can find that the average numbers of trials are very close to 2^n. It proves that our theoretical analysis is correct.

Fig. 2

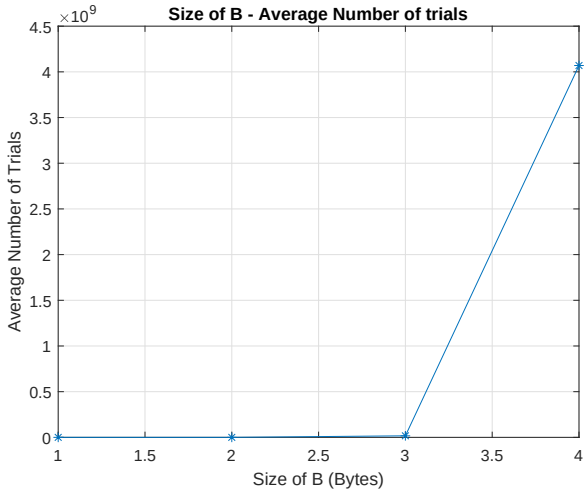


If we perform the curve fitting on Fig. 3(a) and Fig. 3(b), we will get two formulations:
$f(B) = 1.207 \times exp(5.485 \times B)$,
$g(B) = 1.241 \times 10^{-6} \times exp(5.542 \times B)$,
where $f(B)$ equals to the average number of trials, and $g(B)$ equals to the average time.

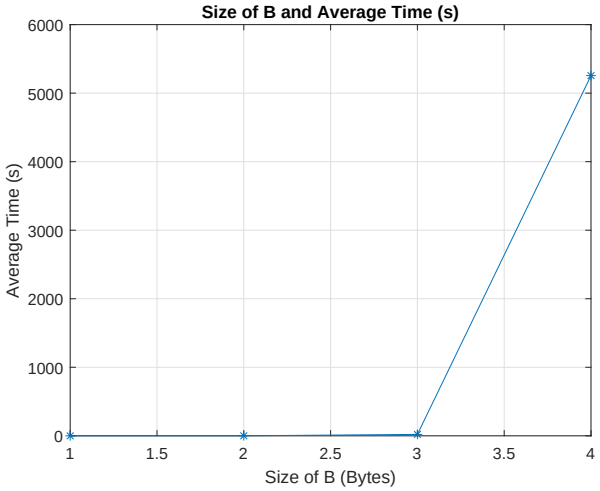
When $B = 32$, 2.04×10^{76} trials will be taken to find the solution on average, while 1.3×10^{71} seconds, about 4.12×10^{63} years, will be spent on average. It shows that it's impossible to find the original message by brute force method, at least at the mathematical level.

As shown in Fig. 3(c), average time and average number of trials are proportional, which indicates our timing measurements are reasonable.

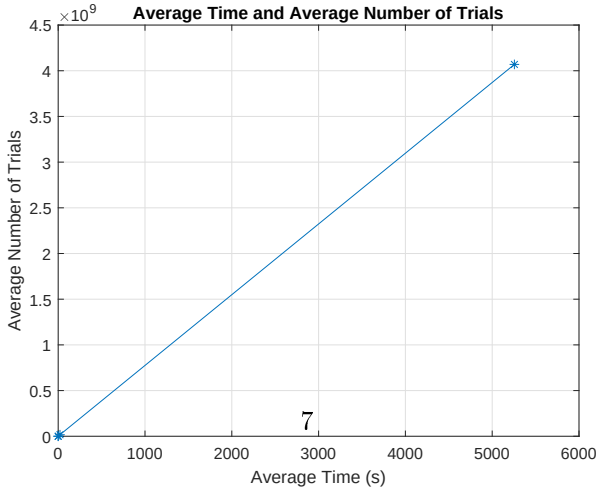
Fig. 3



(a) Size of B - Average Number of Trails



(b) Size of B and Average Time (s)



(c) Average Time and Average Number of Trials

4 Conclusion

The security of a Hash function depends on whether we can find an integral collision of such function. As a result of SHA-256's iterative structure, the corresponding complexity of collision will increase dramatically with an increased number of wheels of iteration algorithm, which makes it hard to find an integral collision.

And from our experiments, we find that to "reverse" the SHA-256 algorithm using brute-force method seems impossible, because we need 4.12×10^{63} years on average to find the original message on average.

In a word, it is very difficult to find an integral collision of SHA-256 with current attack methods, but we still need to improve our technologies to prevent potential threats from attackers.

References

[1] W. Penard and T. Werkhoven, "On the Secure Hash Algorithm family", Gerard Tel (ed.) - Cryptography in Context, pp.1-18, 2008 .

[2] Q. Dang, "Secure hash standard", Technical Report, National Institute of Standards and Technology, August 2002.

[3] J. Patarin, A. Montreuil, "Benes and Butterfly schemes revisited", Université de Versailles, 2007.

[4] R. He, J. Ma, "Analysis safety of SHA-256 algorithm ", Electronic Design Engineering, vol. 22, no. 3, pp. 31-33, February 2014.

[5] E. Biham, A. Shamir, "Differential Cryptanalysis of DES-like Cryptosystems", July 1990.