information

Communication

Algebraic Fault Analysis of SHA-256 Compression Function and Its Application

Kazuki Nakamura [1], Koji Hori [2] and Shoichi Hirose [3,*]

[1] Graduate School of Engineering, University of Fukui, Fukui 910-8507, Japan; ms210428@u-fukui.ac.jp
[2] Tokai Rika Co., Ltd., Oguchi 480-0195, Japan
[3] Faculty of Engineering, University of Fukui, Fukui 910-8507, Japan
* Correspondence: hrs_shch@u-fukui.ac.jp

Abstract: Cryptographic hash functions play an essential role in various aspects of cryptography, such as message authentication codes, pseudorandom number generation, digital signatures, and so on. Thus, the security of their hardware implementations is an important research topic. Hao et al. proposed an algebraic fault analysis (AFA) for the SHA-256 compression function in 2014. They showed that one could recover the whole of an unknown input of the SHA-256 compression function by injecting 65 faults and analyzing the outputs under normal and fault injection conditions. They also presented an almost universal forgery attack on HMAC-SHA-256 using this result. In our work, we conducted computer experiments for various fault-injection conditions in the AFA for the SHA-256 compression function. As a result, we found that one can recover the whole of an unknown input of the SHA-256 compression function by injecting an average of only 18 faults on average. We also conducted an AFA for the SHACAL-2 block cipher and an AFA for the SHA-256 compression function, enabling almost universal forgery of the chopMD-MAC function.

Keywords: algebraic fault analysis; SHA-256 compression function; SAT solver; MAC function

Citation: Nakamura, K.; Hori, K.; Hirose, S. Algebraic Fault Analysis of SHA-256 Compression Function and Its Application. *Information* **2021**, *12*, 433. https://doi.org/10.3390/info12100433

Academic Editor: Murilo da Silva Baptista

Received: 23 August 2021
Accepted: 15 October 2021
Published: 19 October 2021

1. Introduction

1.1. Background

Side-channel attacks are severe threats to hardware implementations of cryptographic algorithms. Fault attacks (FAs) are side-channel attacks that intentionally cause faults in the cryptographic process on a hardware device and try to recover the secret information from internal information that is not usually output. They can cause faults, for example, by irradiating electromagnetic waves, such as lasers, or by manipulating the device's voltage. Fault attacks were first proposed by Boneh et al. in 1996 [1,2], and then Biham and Shamir proposed a differential fault analysis (DFA) for DES in 1997 [3]. The basic principle of DFAs is to recover the secret key by using the output difference between normal and fault-injected executions and the cryptographic algorithm from the point of fault injection to the output. DFAs were also applied to other ciphers [4–6]. In addition, DFAs on the compression functions of cryptographic hash functions were studied. Hemme et al. proposed a DFA against the SHA-1 compression function using a 32-bit random fault model [7]. Based on this attack, DFAs were also applied to the HAS-160 [8] and MD5 [9] compression functions. On the other hand, Courtois et al. proposed an algebraic fault analysis (AFA) for DES [10]. An AFA can analyze more information than a DFA and recover secret information with fewer fault injections. AFAs were also applied to other ciphers [11,12].

When performing an AFA, a SAT solver and an SMT solver are used to solve algebraic equations. A SAT solver is a program that judges whether a given problem is satisfiable or not and outputs a solution if it is satisfiable. The problem is given in the form of a CNF formula. An SMT solver is an extension of the SAT solver for propositional logic and deals with predicate logic satisfiability problems. STP [13] is a kind of SMT solver, which solves bit-vector theory. STP converts a given problem into a CNF equation and then calls a SAT

solver to solve it. The SAT solver we used was CryptoMiniSat 5 [14], which is capable of processing cryptographic problems quickly.

1.2. Related Work

SHA-2 is a family of cryptographic hash functions standardized by NIST in FIPS PUB 180-4 [15]. SHA-256 is included in the family. It is widely deployed and is one of the most important cryptographic hash functions. HMAC is a MAC function standardized by NIST in FIPS PUB 198-1 [16].

Jeong et al. [17] recovered the secret key of HMAC/NMAC-SHA-2 by injecting a fault during the computation of HMAC/NMAC and reducing the number of steps in the SHA-2 compression function. Their fault model assumes that an attacker can change the number of steps in the SHA-2 compression function. Hao et al. [18] proposed an AFA for the SHA-256 compression function using a 32-bit random fault model. They recovered the whole of an unknown input of the SHA-256 compression function by injecting 65 faults. They also presented an almost universal forgery attack on HMAC-SHA-256 based on this result. Nejati et al. [19] assumed an advanced fault injection device and improved the AFA by Hao et al. [18] in 2018. They recovered the whole of an unknown input message block of the SHA-256 compression function with fewer fault injections.

1.3. Our Contribution

We first conducted computer experiments on various fault injection conditions and showed that one can recover the whole of an unknown input of the SHA-256 compression function by injecting about 18 faults on average. While our AFA reduces the number of fault injections compared to that of Hao et al., it requires more time to solve algebraic equations. However, a standard PC can solve them in less than an hour, and it still seems practical. Next, we performed an AFA for the block cipher SHACAL-2 [20]. It can recover the secret key with 12 fault injections by using a standard PC to solve algebraic equations in less than an hour. Finally, we performed an AFA for the SHA-256 compression function, which enabled almost universal forgery of the chopMD-MAC function [21].

1.4. Organization

In Section 2, we describe the SHA-256 compression function, the block cipher SHACAL-2, and the chopMD-MAC function. We also review the AFA for the SHA-256 compression function of Hao et al. In Section 3, we first describe the results of computer experiments on the AFA of Hao et al. and show that the number of injected faults can be greatly reduced. Next, we present the results of the AFA for SHACAL-2 and the AFA for the SHA-256 compression function, which enables almost universal forgery of the chopMD-MAC function. Section 4 is the conclusion.

2. Materials and Methods

This section describes the SHA-256 compression function [15], the SHACAL-2 block cipher [20], and the chopMD-MAC function [21]. We also review the AFA for the SHA-256 compression function of Hao et al. [18].

2.1. SHA-256 Compression Function

SHA-256 is an iterated hash function producing a 256-bit output. The SHA-256 compression function takes a 512-bit message block and a 256-bit intermediate hash value as an input and produces a 256-bit new intermediate hash value as an output.

The algorithm of the SHA-256 compression function is described below. It is also depicted in Figure 1. In the following description, the operation $+$ denotes addition modulo 2^{32}, $\oplus$ denotes bitwise XOR, $\wedge$ denotes bitwise AND, $\neg$ denotes bitwise NOT, and $\|$ denotes the concatenation of bit strings.

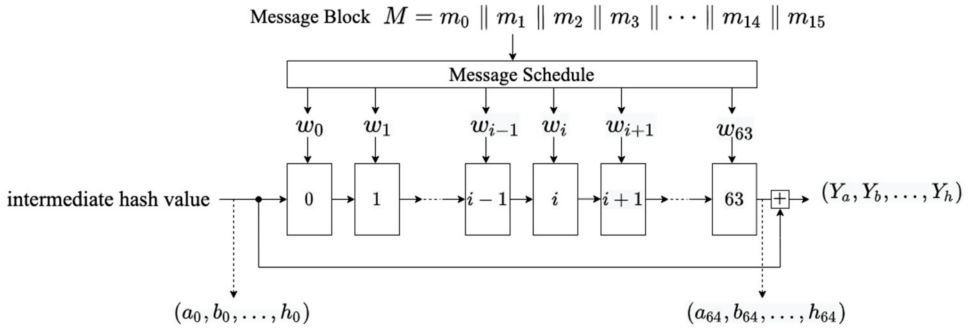

Figure 1. SHA-256 compression function.

The SHA-256 compression function first divides a given 512-bit message block M as follows:

$$M = m_0 \parallel m_1 \parallel m_2 \parallel m_3 \parallel \cdots \parallel m_{14} \parallel m_{15}$$

where $m_j \in \{0,1\}^{32}$. Next, it computes $w_j \in \{0,1\}^{32}$ $(j = 0,\ 1,\ \ldots,\ 63)$ using the following message schedule:

$$w_j = \begin{cases} m_j & \text{if } 0 \le j \le 15, \\ \sigma_1(w_{j-2}) + w_{j-7} + \sigma_0(w_{j-15}) + w_{j-16} & \text{if } 16 \le j \le 63. \end{cases}$$

σ_0 and σ_1 are defined as follows. $\mathbf{ROTR}(n,\ x)$ denotes the right n-bit cyclic shift of x, and $\mathbf{SHR}(n,\ x)$ denotes the right n-bit shift of x.

$$\sigma_0(x) = \mathbf{ROTR}(7,\ x) \oplus \mathbf{ROTR}(18,\ x) \oplus \mathbf{SHR}(3,\ x),$$

$$\sigma_1(x) = \mathbf{ROTR}(17,\ x) \oplus \mathbf{ROTR}(19,\ x) \oplus \mathbf{SHR}(10,\ x).$$

Next, it assigns an input intermediate hash value to $(a_0,\ b_0,\ \ldots,\ h_0)$ and calculates the following steps for $i = 0,\ 1, \ldots,\ 63$ (Figure 2). $a_i,\ b_i,\ \ldots,\ h_i$ are called chaining values, $\alpha_i,\ \beta_i$ are called auxiliary values, and k_i is called a step constant.

$$\alpha_i = h_i + \Sigma_1(e_i) + \mathrm{Ch}(e_i,\ f_i,\ g_i) + k_i + w_i,$$

$$\beta_i = \Sigma_0(a_i) + \mathrm{Maj}(a_i,\ b_i,\ c_i),$$

$$a_{i+1} = \alpha_i + \beta_i,\ b_{i+1} = a_i,$$

$$c_{i+1} = b_i,\ d_{i+1} = c_i,$$

$$e_{i+1} = d_i + \alpha_i,\ f_{i+1} = e_i,$$

$$g_{i+1} = f_i,\ h_{i+1} = g_i,$$

where Ch, Maj, Σ_0, Σ_1 are defined as follows:

$$\mathrm{Ch}(x,\ y,\ z) = (x \wedge y) \oplus (\neg x \wedge z),$$

$$\mathrm{Maj}(x,\ y,\ z) = (x \wedge y) \oplus (x \wedge z) \oplus (y \wedge z),$$

$$\Sigma_0(x) = \mathbf{ROTR}(2,\ x) \oplus \mathbf{ROTR}(13,\ x) \oplus \mathbf{ROTR}(22,\ x),$$

$$\Sigma_1(x) = \mathbf{ROTR}(6,\ x) \oplus \mathbf{ROTR}(11,\ x) \oplus \mathbf{ROTR}(25,\ x).$$

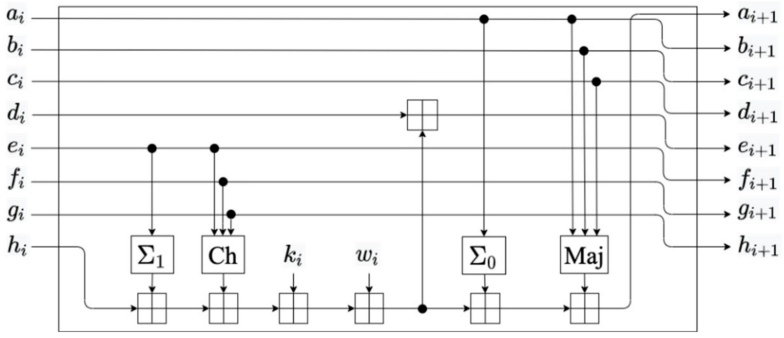

Figure 2. Round function of SHA-256 compression function.

Finally, the SHA-256 compression function outputs H as a new intermediate hash value:

$$H = Y_a \parallel Y_b \parallel Y_c \parallel Y_d \parallel Y_e \parallel Y_f \parallel Y_g \parallel Y_h$$

where

$$Y_a = a_{64} + a_0,\ Y_b = b_{64} + b_0,$$
$$Y_c = c_{64} + c_0,\ Y_d = d_{64} + d_0,$$
$$Y_e = e_{64} + e_0,\ Y_f = f_{64} + f_0,$$
$$Y_g = g_{64} + g_0,\ Y_h = h_{64} + h_0.$$

2.2. SHACAL-2

SHACAL-2 [20] is a block cipher based on the SHA-256 compression function. A plain text is given as an intermediate hash value, and a secret key as a message block. The ciphertext is $(a_{64},\ b_{64},\ \ldots,\ h_{64})$. The key length is recommended to be at least 128 bits.

2.3. ChopMD-MAC

The chopMD-MAC function [21] adds a chop function to the output of the MD hash function, whose initial value is the secret key. The chop function truncates an n-bit input and outputs an s-bit output (Figure 3). In this work, we assume that chopMD-MAC uses the SHA-256 compression function ($n = 256$) and $s = 128$.

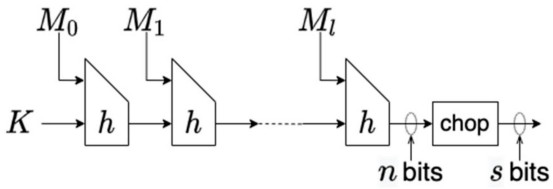

Figure 3. chopMD-MAC function.

2.4. Algebraic Fault Analysis for the SHA-256 Compression Function by Hao et al.

This section reviews the AFA procedure for the SHA-256 compression function in the study by Hao et al. [18]. It assumes that a 32-bit fault is injected into a specified chaining value among $a_i, b_i, \ldots, h_i$ $(i = 0, 1, \ldots, 63)$. When a fault is injected, the chaining value is changed to a random value unknown to the attacker, and a faulty output is obtained after the cryptographic operation. With the faulty output and the normal output, an algebraic equation is constructed for the operations from the injected fault location to the output. Through repeated fault injection into the same chaining value, multiple algebraic equations are obtained for the same operations. STP is used to solve the algebraic equations.

The process of the AFA for the SHA-256 compression function can be divided into two phases. Phase 1 recovers the chaining value $p_{63} = (a_{63},\ b_{63}, \ldots,\ h_{63})$. Phase 2 recovers the whole input to the SHA-256 compression function.

2.4.1. Phase 1

In this phase, 14 faults are injected into c_{60}, and the 14 corresponding faulty outputs are obtained. Next, the algebraic equations are constructed based on the normal and faulty outputs and the operations (including feed-forward operations) for the four steps from 60 to 63. STP solves the algebraic equations and returns a solution for p_{63}. Hao et al. reported that they succeeded in recovering p_{63} in all of the 100 trials of the procedure above. They also reported that they always succeeded in recovering p_{63} in the same way by injecting 13 faults into c_{58} or c_{59}.

In addition, they examined the number of chaining values p_i's recovered at the same time and the computation time for the cases where 13 faults were injected into one of a_{59}, b_{59}, $\ldots, h_{59}$. The results showed that there was a positive correlation between the number of recovered chaining values and the percentage of computation time larger than 200 s.

2.4.2. Phase 2

First, from the non-faulty chaining value p_{63} obtained in Phase 1 and the correct output Y, the faulty chaining value $p^*_{63} = (a^*_{63}, b^*_{63}, \ldots, h^*_{63})$ corresponding to a faulty output Y^* can be calculated as follows:

$$\begin{aligned}
a^*_{63} &= Y^*_b - Y_b + a_{63},\\
b^*_{63} &= Y^*_c - Y_c + b_{63},\\
c^*_{63} &= Y^*_d - Y_d + c_{63},\\
d^*_{63} &= Y^*_e - Y_e - (T^*_{63} - T_{63}) + d_{63}\\
e^*_{63} &= Y^*_f - Y_f + e_{63},\\
f^*_{63} &= Y^*_g - Y_g + f_{63},\\
g^*_{63} &= Y^*_h - Y_h + g_{63},\\
h^*_{63} &= T^*_{63} - T_{63} + f_1(e^*_{63}, f^*_{63}, g^*_{63}) - f_1(e_{63}, f_{63}, g_{63}) + h_{63},
\end{aligned}$$

where

$$\begin{aligned}
T^*_{63} &= h^*_{63} + \Sigma_1(e^*_{63}) + \mathrm{Ch}(e^*_{63}, f^*_{63}, g^*_{63}) + k_{63} + w_{63},\\
f_1(e_{63}, f_{63}, g_{63}) &= \Sigma_1(e_{63}) + \mathrm{Ch}(e_{63}, f_{63}, g_{63}).
\end{aligned}$$

Phase 2 proceeds as follows. First, 13 faults are injected into c_{56}, and faulty outputs are obtained. For each faulty output Y^*, the corresponding faulty chaining value p^*_{63} is calculated. Based on these values, the algebraic equations for the seven steps from 56 to 62 are constructed. Subsequently, STP solves them and finds the values of $w_{59}, w_{60}, w_{61}, w_{61}, w_{62}$. Hao et al. claimed that they were successful in all of their 100 trials.

Second, 13 faults are injected into c_{52}. For each faulty output Y^*, the faulty chaining value p^*_{63} is calculated in the same way as above. For faulty p^*_{63} and non-faulty p_{63}, p^*_{59} and p_{59} are calculated, respectively, with recovered $w_{59}, w_{60}, w_{61}, w_{62}$. Then, based on p^*_{59} and p_{59}, algebraic equations for the seven steps from 52 to 58 are constructed. STP solves them and finds the values of $w_{55}, w_{56}, w_{57}, w_{58}$.

In a similar way, by injecting faults into c_{48} and c_{44}, w_{51}, w_{52}, w_{53}, w_{54} and w_{47}, w_{48}, w_{49}, w_{50} are recovered. The input message block $m_0 \parallel m_1 \parallel \cdots \parallel m_{15}$ can be calculated from the recovered $w_{47}, \ldots, w_{62}$ according to the message schedule. Once the input message block is recovered, w_{63} can be calculated. Next, p_{64} can be calculated from p_{63} and w_{63}, and the input intermediate hash value $p_0 = H - p_{64}$ is recovered.

In summary, by injecting 65 32-bit random faults in total (13 faults are injected five times), the whole input of the SHA-256 compression function can be recovered.

3. Results

3.1. Detailed Examination of AFA

We conducted computer experiments on various fault-injection conditions in the AFA for the SHA-256 compression function described in Section 2. In the experiments, we implemented the SHA-256 compression function in the C language and simulated fault injections. STP was used as an automatic tool, and CryptoMiniSat5 was used as a SAT solver. The experiments were performed on a PC with Intel(R) Xeon(R) Silver 4110 CPU @ 2.10 GHz, 32 G memory, and Ubuntu 18.04.3 LTS.

For Phase 1, we first injected 13 faults into each of the chaining values from a_{60} to h_{60}, and tried to recover as many chaining values $p_i = (a_i, b_i, \ldots, h_i)$ as possible. The results are shown in Table 1. The computation time is the average of 100 trials. The results show that the fault injections into e_{60} can recover more chaining values than the fault injections into other positions.

Table 1. Recovered chaining values and computation time.

Fault Position	Recovered Chaining Values	Computation Time [s]
a_{60}	none	-
b_{60}	none	-
c_{60}	p_{63}	20.3
d_{60}	p_{63}, p_{62}	176.5
e_{60}	p_{63}, p_{62}, p_{61}	66.7
f_{60}	p_{63}, p_{62}	82.0
g_{60}	p_{63}	109.6
h_{60}	p_{63}, p_{62}	110.0

Next, we changed the number of faults injected into c_{60} or e_{60} in Phase1 and c_{56} or e_{56} in Phase 2 and measured the number of times the correct chaining values were recovered out of 100 trials. The results are shown in Tables 2 and 3 for Phase 1 and Phase 2, respectively. For Phase 1, except for the cases where four or fewer faults were injected, the correct chaining values were recovered with higher probabilities when faults were injected into e_{60} than when faults were injected into c_{60}. For Phase 2, even when only three faults were injected into e_{56}, all 100 trials were successful.

Table 2. Success rate and average time for each number of faults (Phase 1). "-" indicates that the result was not output within 72 h.

Phase 1	Fault Position c_{60}		Fault Position e_{60}	
Number of Faults	Success Rate [%]	Average Time [s]	Success Rate [%]	Average Time [s]
14	100	11.6	100	39.9
13	97	11.0	100	41.7
12	98	10.9	100	41.0
11	100	11.2	100	37.4
10	96	10.5	99	47.4
9	93	11.2	100	45.5
8	88	10.8	100	59.3
7	80	10.3	96	44.5
6	75	12.6	89	88.0
5	50	13.7	82	123.9
4	30	31.3	-	-
3	1	112.5	-	-

Table 3. Success rate and average time for each number of faults (Phase 2).

Phase 2	Fault Position c_{56}		Fault Position e_{56}	
Number of Faults	Success Rate [%]	Average Time [s]	Success Rate [%]	Average Time [s]
14	100	1.00	100	81.34
13	97	0.93	100	83.48
12	98	0.91	100	89.20
11	100	0.86	100	95.11
10	96	0.84	100	67.34
9	93	0.76	100	72.71
8	88	0.77	100	102.73
7	80	0.73	100	98.08
6	75	0.70	100	102.01
5	50	0.64	100	87.25
4	30	0.77	100	121.75
3	1	0.78	100	492.76

From the experimental results described above, the procedure of AFA with the smallest expected number of faults (with the cases of the colored parts in Tables 2 and 3) is given below.

Phase 1: Inject five faults into e_{60} and recover the chaining value p_{63}.

Phase 2:

(1) Inject three faults into e_{56} and recover w_{59}, w_{60}, w_{61}, and w_{62}.
(2) Inject three faults into e_{52} and recover w_{55}, w_{56}, w_{57}, and w_{58}.
(3) Inject three faults into e_{48} and recover w_{51}, w_{52}, w_{53}, and w_{54}.
(4) Inject three faults into e_{44} and recover w_{47}, w_{48}, w_{49}, and w_{50}.
(5) Calculate the input message block and intermediate hash value.

For the procedure, the expected number of faults is 18.1, and the computation time is about 35 min.

3.2. AFA for SHACAL-2

Since SHACAL-2 feeds the key to the message input of the SHA-256 compression function, the goal of the AFA is to recover the message input of the SHA-256 compression function. SHACAL-2 outputs the chaining value $(a_{64}, b_{64}, \ldots, h_{64})$ as a ciphertext without the feed-forward operation of the SHA-256 compression function. Thus, we can start the AFA with Phase 2 in Section 2. The procedure is shown below:

(1) Inject three faults into e_{57} and recover w_{60}, w_{61}, w_{62}, and w_{63}.
(2) Inject three faults into e_{53} and recover w_{56}, w_{57}, w_{58}, and w_{59}.
(3) Inject three faults into e_{49} and recover w_{52}, w_{53}, w_{54}, and w_{55}.
(4) Inject three faults into e_{45} and recover w_{48}, w_{49}, w_{50}, and w_{51}.
(5) Calculate the input message.

For the procedure with 12 fault injections in total, all of 100 trials were successful. The total computation time was about 33 min.

3.3. AFA for Almost Universal Forgery of chopMD-MAC

For almost universal forgery of the chopMD-MAC function, we recover the hash value input $(a_0, b_0, \ldots, h_0)$ of the SHA-256 compression function under the condition that the message input and only 128 bits of the 256-bit output are known. Since $(a_0, b_0, \ldots, h_0)$ can be calculated from p_{63} and the message input, only p_{63} has to be recovered.

We assumed that the chop function outputs four words among the eight-word output $(Y_a, Y_b, \ldots, Y_h)$ of the SHA-256 compression function. We injected 14 faults into c_{60} and analyzed the SHA-256 compression function under the condition mentioned above. Since we found that the SHA-256 compression function does not propagate the faults to Y_h, we

excluded the cases that the chop function outputs Y_h. We verified all the other 35 cases. Table 4 shows all the successful cases (p_{63} was recovered). The success rate is for 10 trials.

Table 4. Average time and success rate of the analysis for chop-MD.

Output of Chop Function	Average Time [s]	Success Rate [%]
(Y_a, Y_b, Y_d, Y_e)	1072	100
(Y_a, Y_b, Y_c, Y_e)	46,344	80
(Y_a, Y_b, Y_e, Y_f)	37,574	100
(Y_a, Y_b, Y_e, Y_g)	49,950	70
(Y_b, Y_c, Y_e, Y_f)	2012	90
(Y_b, Y_d, Y_e, Y_f)	15,444	80

4. Conclusions

In this study, we changed the fault-injection condition of the AFA for the SHA-256 compression function in the study by Hao et al. and conducted computer experiments. We found that the whole input of the SHA-256 compression function can be recovered with a smaller number of faults. We also demonstrated the results of computer experiments on the AFA for SHACAL-2 and the AFA for the SHA-256 compression, which enabled almost universal forgery of chopMD-MAC. Future work should include an AFA for the SHA-512 compression function.

Author Contributions: Conceptualization, K.H. and S.H.; methodology, K.H. and S.H.; software, K.N. and K.H.; validation, K.N., K.H.; formal analysis, K.H.; investigation, K.N. and K.H.; resources, S.H.; data curation, K.N. and K.H.; writing—original draft preparation, K.N. and K.H.; writing—review and editing, S.H.; visualization, K.N.; supervision, S.H.; project administration, S.H.; funding acquisition, S.H. All authors have read and agreed to the published version of the manuscript.

Funding: The authors were supported in part by JSPS KAKENHI Grant Number JP18H05289. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript, or in the decision to publish the results.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The data presented in this study are available on request from the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Boneh, D.; DeMillo, R.A.; Lipton, R.J. On the Importance of Checking Cryptographic Protocols for Faults (Extended Abstract). In *Advances in Cryptology-EUROCRYPT '97, International Conference on the Theory and Application of Cryptographic Techniques, Konstanz, Germany, 11–15 May 1997*; Fumy, W., Ed.; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1233, pp. 37–51.
2. Boneh, D.; DeMillo, R.A.; Lipton, R.J. On the Importance of Eliminating Errors in Cryptographic Computations. *J. Cryptol.* **2001**, *14*, 101–119. [CrossRef]
3. Biham, E.; Shamir, A. Differential fault analysis of secret key cryptosystems. In *Advances in Cryptology-CRYPTO '97, 17th Annual International Cryptology Conference, Santa Barbara, CA, USA, 17–21 August 1997*; Kaliski, B.S., Ed.; Springer: Berlin/Heidelberg, Germany, 1997; Volume 1294, pp. 513–525.
4. Dusart, P.; Letourneux, G.; Vivolo, O. Differential Fault Analysis on A.E.S. In *Applied Cryptography and Network Security, First International Conference, ACNS 2003, Kunming, China, 16–19 October 2003*; Zhou, J., Yung, M., Han, Y., Eds.; Springer: Berlin/Heidelberg, Germany, 2003; Volume 2846, pp. 293–306.
5. Li, R.; Li, C.; Gong, C. Differential Fault Analysis on SHACAL-1. In *Sixth International Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2009, Lausanne, Switzerland, 6 September 2009*; Breveglieri, L., Koren, I., Naccache, D., Oswald, E., Seifert, J.P., Eds.; IEEE Computer Society: Washington, DC, USA, 2009; pp. 120–126.
6. Kitae, J.; Lee, C. Differential Fault Analysis on Block Cipher LED-64. In *Future Information Technology, Application, and Service*; Hyuk, J., Park, J., Leung, V., Wang, C.L., Shon, T., Eds.; Springer: Dordrecht, The Netherland, 2012; Volume 164, pp. 747–755.

7. Hemme, L.; Hoffmann, L. Differential Fault Analysis on the SHA1 Compression Function. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography, FDTC 2011, Tokyo, Japan, 29 September 2011*; Breveglieri, J., Guilley, S., Koren, I., Naccache, D., Takahashi, J., Eds.; IEEE Computer Society: Washington, DC, USA, 2011; pp. 54–62.
8. Jinkeon, K.; Kitae, J.; Jaechul, S.; Seokhie, H. Differential Fault Analysis on HAS-160 Compression Function. In *Computer Science and Its Applications*; Yeo, S.S., Pan, Y., Lee, Y., Chang, H., Eds.; Springer: Dordrecht, The Netherland, 2012; Volume 203, pp. 97–105.
9. Li, W.; Tao, Z.; Gu, D.; Wang, Y.; Liu, Z.; Liu, Y. Differential Fault Analysis on the MD5 Compression Function. *J. Comput.* **2013**, *8*, 2888–2894. [CrossRef]
10. Courtois, N.; Ware, D.; Jackson, K. Fault-Algebraic Attacks on Inner Rounds of DES. In Proceedings of the eSmart 2010 European Smart Card Security Conference, Sophia Antipolis, France, 22–24 September 2010.
11. Bouillaguet, C.; Derbez, P.; Fouque, P.A. Automatic Search of Attacks on Round-Reduced AES and Applications. In *Advances in Cryptology-CRYPTO 2011—31st Annual Cryptology Conference, Santa Barbara, CA, USA, 14–18 August 2011*; Rogaway, P., Ed.; Springer: Berlin/Heidelberg, Germany, 2011; Volume 6841, pp. 169–187.
12. Zhao, X.; Guo, S.; Zhang, F.; Shi, Z.; Ma, C.; Wang, T. Improving and Evaluating Differential Fault Analysis on LED with Algebraic Techniques. In *2013 Workshop on Fault Diagnosis and Tolerance in Cryptography, Los Alamitos, CA, USA, 20 August 2013*; Fischer, W., Schmidt, J.M., Eds.; IEEE Computer Society: Washington, DC, USA, 2013; pp. 41–51.
13. Ganesh, V.; Dill, D.L. A Decision Procedure for Bit-Vectors and Arrays. In *Computer Aided Verification, 19th International Conference, CAV 2007, Berlin, Germany, 3–7 July 2007*; Damm, W., Hermanns, H., Eds.; Springer: Berlin/Heidelberg, Germany, 2007; Volume 4590, pp. 519–531.
14. CryptoMiniSat5. Available online: https://www.msoos.org/cryptominisat5/ (accessed on 23 August 2021).
15. National Institute of Standards and Technology. *Secure Hash Standard (SHS). FIPS PUB 180-4*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2015.
16. National Institute of Standards and Technology. *The Keyed-Hash Message Authentication Code (HMAC). FIPS PUB 198-1*; National Institute of Standards and Technology: Gaithersburg, MD, USA, 2008.
17. Jeong, K.; Lee, Y.; Sung, J.; Hong, S. Security Analysis of HMAC/NMAC by Using Fault Injection. *J. Appl. Math.* **2013**, *2013*, 101907:1–101907:6. [CrossRef]
18. Hao, R.; Li, B.; Ma, B.; Song, L. Algebraic Fault Attack on the SHA-256 Compression Function. *IJORCS* **2014**, *4*, 1–9. [CrossRef]
19. Nejati, S.; Horáček, J.; Gebotys, C.; Ganesh, V. Algebraic Fault Attack on SHA Hash Functions Using Programmatic SAT Solvers. In *Principles and Practice of Constraint Programming—24th International Conference, CP 2018, Lille, France, 27–31 August 2018*; Hooker, J., Ed.; Springer: Cham, Switzerland, 2018; Volume 11008, pp. 737–754.
20. Modifications to NESSIE Submissions Selected for 2nd Phase. Available online: https://www.cosic.esat.kuleuven.be/nessie/tweaks (accessed on 23 August 2021).
21. Naito, Y.; Sasaki, Y.; Wang, L.; Yasuda, K. Generic State-Recovery and Forgery Attacks on ChopMD-MAC and on NMAC/HMAC. In *Advances in Information and Computer Security—8th International Workshop on Security, IWSEC 2013, Okinawa, Japan, 18–20 November 2013*; Sakiyama, K., Terada, M., Eds.; Springer: Berlin/Heidelberg, Germany, 2013; Volume 8231, pp. 83–98.