International Journal of Research in Computer Science
eISSN 2249-8265 Volume 4 Issue 2 (2014) pp. 1-9
www.ijorcs.org, A Unit of White Globe Publications
doi: 10.7815/ijorcs.42.2014.079

ALGEBRAIC FAULT ATTACK ON THE SHA-256 COMPRESSION FUNCTION

Ronglin Hao[1,2], Bao Li[2], Bingke Ma[2], Ling Song[2]
[1]*Department of Electronic Engineering and Information Science, University of Science and Technology of China, Hefei, 230027, CHINA*
Email: haorl@mail.ustc.edu.cn
[2]*State Key Laboratory of Information Security, Institute of Information Engineering, Chinese Academy of Sciences, Beijing, 100093, CHINA*
Email: bkma@is.ac.cn

Abstract: *The cryptographic hash function SHA-256 is one member of the SHA-2 hash family, which was proposed in 2000 and was standardized by NIST in 2002 as a successor of SHA-1. Although the differential fault attack on SHA-1compression function has been proposed, it seems hard to be directly adapted to SHA-256. In this paper, an efficient algebraic fault attack on SHA-256 compression function is proposed under the word-oriented random fault model. During the attack, an automatic tool STP is exploited, which constructs binary expressions for the word-based operations in SHA-256 compression function and then invokes a SAT solver to solve the equations. The simulation of the new attack needs about 65 fault injections to recover the chaining value and the input message block with about 200 seconds on average. Moreover, based on the attack on SHA-256 compression function, an almost universal forgery attack on HMAC-SHA-256 is presented. Our algebraic fault analysis is generic, automatic and can be applied to other ARX-based primitives.*

Keywords: *Algebraic Fault Analysis, HMAC, SHA-256 Compression Function, SAT solver, STP.*

I. INTRODUCTION

As seen in the last decade, the cryptographic community begins to investigate the security of the hardware implementation of a cryptographic algorithm. As a common way of Side Channel Attack (SCA), Differential Fault Analysis (DFA) induces faults into the calculation of the hardware device and takes the faulty output values as side-channel. Then the relations between the correct and faulty outputs are exploited to extract secret information within the device of the target cryptographic algorithm.

The concept of fault attacks was first introduced by Boneh, Demillo and Lipton in 1996 [6, 7]. Then Biham and Shamir proposed the DFA attack in 1997, which processes the right and faulty outputs with differential cryptanalysis [8]. After that, DFA has been successfully applied to many block ciphers and stream ciphers, such as AES [12, 33, 34], SHACAL1 [11], LED [35, 36], Piccolo [37], PRINCE [24], Trivium [9, 23], RC4 [13, 14]. Besides attacks against block and stream ciphers, the DFA attack on the compression function of a hash function has also been studied. At FDTC 2011, Hemme L. et al. proposed a DFA attack on SHA-1 compression function under the word-oriented random fault model [5]. Its basic principle is to construct single-variable equations by exploiting the differences between the correct output and faulty outputs to retrieve the internal state and the input message block. Based on this attack, similar DFA attacks on the HAS-160 and MD5 compression functions have also been discussed in [21, 22]. At FDTC 2012, Fischer et al. presented a DFA attack on Grøstl-256 [39], whose structure is similar to AES.

Instead of combining fault attack with differential cryptanalysis, Courtois et al. proposed an algebraic fault analysis (AFA) on DES in eSmart 2010 [10], which combines fault attack with algebraic techniques [32]. The AFA attack first constructs algebraic equations for the cipher and the faults, and then invokes the automatic tools to solve the equations and recover the secret information of the cipher. An important advantage of AFA over DFA is that AFA does not rely on the manual analysis of differential propagations. By applying AFA, the previous DFA attacks on block and stream ciphers have been improved, such as AES [30], LED [26, 31], Piccolo [4], Trivium [29].

The SHA-2 hash family was proposed as a successor of SHA-1 in 2000 and was standardized by NIST in 2002. SHA-256 is a member of the SHA-2 family which outputs a 256-bit digest. There are few researches on SHA-256 against fault attacks. In [38], Jeong et al. recovered the secret key of HMAC/NMAC-SHA-2 by reducing the number of steps of SHA-2 compression function via fault injections during the calculation of HMAC/NMAC [1].

Their fault model assumes that the adversary could exactly modify the number of steps of the target compression function and therefore is very restrictive. In this paper the security of SHA-256 compression function against fault attacks is investigated under a more relaxed and realistic fault model, i.e., the 32-bit-word-oriented random fault model.

Although SHA-256 shares the similar design principles of SHA-1, the concrete structure of SHA-256 prevents the attack in [5] from being directly applied to SHA-256 compression function under the word-oriented random fault model. However, SHA-256 compression function is found to be vulnerable to AFA, and thus an AFA attack is proposed against it in this paper. During the attack, the automatic toolkit STP [25] is applied, which can be used to construct binary expressions for every simple operation, i.e., addition, Boolean function, rotation and XOR in the compression function, and then to invoke a SAT solver to solve the algebraic equations. By injecting about 65 faults, the secret inputs of SHA-256 compression function could be revealed with about 200 seconds on average. Based on this attack, an almost universal forgery attack on HMAC-SHA-256 is proposed. Our attack is generic and can be easily extended to evaluate the security properties of other ARX-based primitives against AFA.

The rest of this paper is organized as follows. Section II briefly describes the SHA-256 hash function and the HMAC algorithm. The reason why the DFA attack in [5] cannot be applied to SHA-256 compression function is also discussed in Section II. Section III proposes an AFA attack on SHA-256 compression function using STP [25] under the word-oriented random fault model. Section IV presents an almost universal forgery attack on HAMC-SHA-256 based on the AFA attack on SHA-256 compression function. Section V briefly discusses how the concrete structure of SHA-256 affects AFA and the potential application of AFA to improve DFA on SHA-1 compression function. Finally the last section concludes the paper.

II. PRELIMINARIES

A. SHA-256 hash function

The SHA-2 hash family consists of four hash functions with digests of length 224, 256, 384 and 512 bits, i.e., SHA-224, SHA-256, SHA-384 and SHA-512. In the following, a brief description of the SHA-256 algorithm is given. For more details, please refer to [2]. First, the original message M is padded to be a multiple of 512 bits according to Merkle-Damgård strengthening, i.e., a single bit "1", a variable number of 0s, and the 64-bit binary representation of the length of M are appended at the end. Then the padded message is parsed into 512-bit message blocks $M_0, M_1, \cdots, M_{L-1}$. The hash value is computed as follows:

$$H_0 = IV, H_{l+1} = CF(M_l, H_l), 0 \le l < L$$

where H_l is a 256-bit chaining value which consists of 8 32-bit words, + denotes addition modulo 2^{32}. The last chaining value H_L is the output of the hash function. The compression function $CF(M_l, H_l)$ consists of two parts: the message expansion and the state update transformation.

1. Message expansion

The message schedule of SHA-256 splits M_l into 16 32-bit message words $m_0, m_1, \cdots, m_{15}$ and expands them into 64 32-bit words W_i for $0 \le i < 63$, according to the following equation,

$$W_i \leftarrow \begin{cases} m_i & 0 \le i < 16, \\ \sigma_1(W_{i-2}) + W_{i-7} + \sigma_0(W_{i-15}) + W_{i-16} & 16 \le i < 63. \end{cases}$$

The functions $\sigma_0(x)$ and $\sigma_1(x)$ are defined as follows:

$$\sigma_0(x) \leftarrow (x \ggg 7) \oplus (x \ggg 18) \oplus (x \gg 3),$$

$$\sigma_1(x) \leftarrow (x \ggg 17) \oplus (x \ggg 19) \oplus (x \gg 10),$$

where $\oplus$ denotes XOR, $\ggg$ and $\gg$ represent left rotation and left shift respectively.

2. State update transformation

The 256-bit chaining value H_l is updated by applying the step function 64 times. Let $p_i = a_i \| b_i \| c_i \| d_i \| e_i \| f_i \| g_i \| h_i$ be the input of step i ($i = 0, \cdots, 63$), i.e., $p_0 = H_l$ and p_{64} is the output value of step 63. Figure 1 illustrates the transformation of step i. Note that K_i denotes the round constant and the functions T_i, *Ch*, *Maj*, $\sum_0$ and $\sum_1$ are given by

$$T_i = h_i + \sum_1(e_i) + Ch(e_i, f_i, g_i) + K_i + W_i,$$

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

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

$$\sum_0(x) = (x \ggg 2) \oplus (x \ggg 13) \oplus (x \ggg 22),$$

$$\sum_1(x) = (x \ggg 6) \oplus (x \ggg 11) \oplus (x \ggg 25).$$

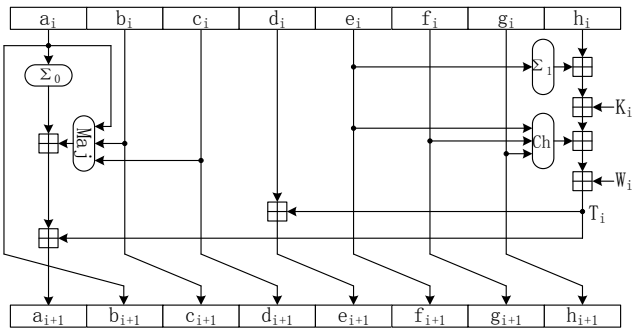

Figure 1: Step function of SHA-256

After 64 expanded message words W_i have been processed, $H_{l+1} = p_{64} + H_l$ is set as the chaining value for the next message block M_{l+1}.

B. HMAC

Hash-based MAC (HMAC) is one of the most widely used MAC constructions, which was first introduced by Bellare et al. in 1996 [1] and then standardized by NIST in 2002 [3]. In the HMAC algorithm, the input message M is authenticated by invoking a hash function $\mathcal{H}$ twice with a secret key K. The MAC is computed as follows:

$$HMAC_K(M) = \mathcal{H}\big(IV, (K \oplus opad) \parallel \mathcal{H}(IV, (K \oplus ipad) \parallel M)\big),$$

where IV represents the initial value of $\mathcal{H}$, *opad* and *ipad* are two different padding constants, and || denotes concatenation. Note that before processing M, K must be padded with zeros to the length of a single message block of $\mathcal{H}$.

While HMAC can work with any cryptographic hash function, $\mathcal{H}$ is suggested to be a NIST-proved hash function [3]. Hence, SHA-256 is selected to instantiate $\mathcal{H}$, and HMAC instantiated with SHA-256 is depicted with HMAC-SHA-256. As depicted in Figure 2, the output H_{in} of the inner call to SHA-256 is 256 bits, thus it needs to be padded to 512 bits. Consequently, the outer call to SHA-256 only invokes its underlying compression function *CF* twice, i.e., CF_1 and CF_2. Note that *pad* is a 256-bit padding constant.

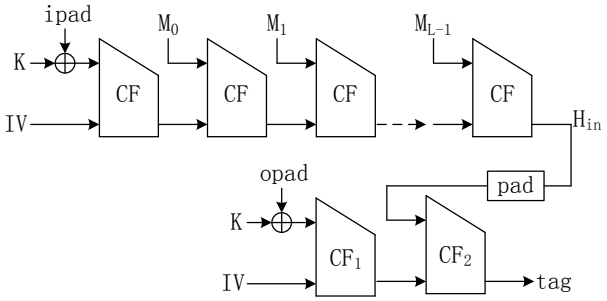

Figure 2: The HMAC-SHA-256 construction

C. DFA on SHA-1 compression function

By injecting 1,002 random word faults, the DFA attack [5] on SHA-1 compression function can recover the chaining value and the input message block with the success rate 73.7%. The fault model used in [5] is described first. The attack uses a **word-oriented random fault model**, i.e., **the fault with a random unknown 32-bit value is induced into the particular intermediate 32-bit state register of a specified step**.

The attack procedure in [5] contains two phases. In the first phase, the final addition is eliminated, and the input value $p_{79} = a_{79} \parallel b_{79} \parallel c_{79} \parallel d_{79} \parallel e_{79}$ of step 79 is computed as well. The second phase is similar to the DFA attack on SHACAL1 [11], and aims to extract the input message block M_l. The success of the attack mainly attributes to the constructions of the following two single-variable equations:

$$((X + \Phi) \ggg t) - (X \ggg t) = \Psi \quad (1)$$

$$(x \oplus \delta) - x = \Delta \quad (2)$$

where $0 \le X, \Phi, \Psi, x, \delta, \Delta < 2^{32}$ be 32-bit integers, and $0 < t < 32$. If the value of (Φ, Ψ) (resp. (δ, Δ)) can be computed via fault injections, partial information on X (resp. x) will be obtained. Moreover, if more faulty compression values can be obtained, more information will be gained, and consequently the number of candidates for X and x can be reduced.

D. Extension to SHA-256 compression function

Due to the very similar design principles of SHA-2 and SHA-1, it is natural to apply the DFA attack in [5] to SHA-256 compression function *CF* under the word-oriented random fault model. Let $Y = Y_a \parallel Y_b \parallel Y_c \parallel Y_d \parallel Y_e \parallel Y_f \parallel Y_g \parallel Y_h$ be the right output of *CF*. Suppose that a random word fault is induced during another computation of *CF* with the same inputs, then the corresponding faulty compression value $Y^* = Y_a^* \parallel Y_b^* \parallel Y_c^* \parallel Y_d^* \parallel Y_e^* \parallel Y_f^* \parallel Y_g^* \parallel Y_h^*$ is obtained. Moreover, assume that $p_i^* = a_i^* \parallel b_i^* \parallel c_i^* \parallel d_i^* \parallel e_i^* \parallel f_i^* \parallel g_i^* \parallel h_i^*$ is the faulty input of step i *(i = 0, ⋯ ,63)*, where the x_i^* denotes the faulty intermediate state register, $x \in \{a, b, c, d, e, f, g, h\}$. And p_{64}^* is the faulty output of step 63.

Since DFA relies on the property of the difference propagation, the concrete structure of the target algorithm analyzed has a big impact on DFA. The structure of SHA-256 is more complex than SHA-1. Besides the larger state size in SHA-256, there are two other key issues that prevent the adversary from constructing single-variable equations like (1) or (2) via fault injections during the procedure of DFA on SHA-256 compression function.

Firstly, the state variable a_i simultaneously participates in two operations: $\sum_0(a_i)$ and $Maj(a_i, b_i, c_i)$. Hence, if the value of a_{63} is changed to a_{63}^* by inducing one fault, the following system of equations can be obtained:

$$a_{63}^* - a_{63} = Y_b^* - Y_b,$$
$$f_0(a_{63}^*, b_{63}, c_{63}) - f_0(a_{63}, b_{63}, c_{63}) = Y_a^* - Y_a. \quad (3)$$

where $f_0(x, y, z) = \sum_0(x) + Maj(x, y, z)$. Note that the equation system (3) contains three variables a_{63}, b_{63}, c_{63}. Thus partial information on a_{63} cannot be achieved without knowing the values of b_{63} and c_{63}. The similar case also holds for e_i.

Secondly, SHA-256 adopts two non-linear Boolean functions *Ch* and *Maj*, as compared to the linear Boolean function $f(x, y, z) = x \oplus y \oplus z$ used in the last 20 steps of SHA-1. Without loss of generality, assume that the value of b_{63} is changed to b_{63}^* via one fault injection. Consequently, the following three-variable system of equations is obtained:

$$b_{63}^* - b_{63} = Y_c^* - Y_c,$$

$$Maj(a_{63}, b_{63}^*, c_{63}) - Maj(a_{63}, b_{63}, c_{63}) = Y_a^* - Y_a. \quad (4)$$

Since other relationships between $Maj(a_{63}, b_{63}^*, c_{63})$ and $Maj(a_{63}, b_{63}, c_{63})$ cannot be generated, such as the $\oplus$-relationship in (2), the value of $Maj(a_{63}, b_{63}, c_{63})$ cannot be determined. Moreover, the value of b_{63} cannot be calculated as well, since the values of a_{63} and c_{63} are unknown. Hence, partial information on $Maj(a_{63}, b_{63}, c_{63})$ or b_{63} could not be obtained by computing the system (4). The similar case also holds for c_i, f_i, g_i.

Although the (1)- or (2)-like equations could not be constructed to calculate single variables, the values of multiple variables can be simultaneously computed by solving the set of multiple-variable equation systems like (3) or (4) efficiently with algebraic techniques [32]. So attempts are made in this paper to launch a fault attack on SHA-256 compression function using AFA instead of DFA under the word-oriented random fault model.

III. AFA ON SHA-256 COMPRESSION FUNCTION

E. The framework of AFA using STP

Traditional DFA mainly relies on the manual analysis on the propagation of the injected faults. When faults are injected in deeper steps of the compression function, the fault propagation path may become very complicated and the manual analysis procedure is difficult for the adversary. Compared to DFA, AFA treats the fault analysis as an algebraic problem and solves it with some automatic tools. It is worth trying to apply AFA [4] on SHA-256 compression function, and the experiments conducted in this work show positive results. The generic framework of AFA is as follows:

- *Injecting the faults*: the location and the number of random faults to be injected need to be carefully determined. There are many methods to inject a fault, refer to [15, 16, 17] for details.

- *Constructing the equation system for the compression function*: the remaining step functions starting from the location of the injected faults are represented as an equation system. The binary expressions for the bit-oriented operations $\sum_0$, $\sum_1$, *Maj* and *Ch* and the non-linear operation addition modulo 2^{32} need to be constructed carefully.

- *Constructing the equation systems for the faults*: the step functions affected by the induced faults are represented as equation systems as well.

- *Solving the set of equation systems*: the equation systems for the right and faulty computations of the compression function share some common variables, including the chaining value and the extended message words used in the steps affected by the injected faults. Computing their values is equivalent to solving the set of merged equation systems by invoking the automatic tools, such as the SAT solvers [19, 28].

The typical SAT solver takes the conjunctive-normal form (CNF) formulas as inputs, but the Boolean functions and additions in SHA-256 are operations on 32-bit words. Generating the corresponding binary CNF formulas for these word based operations seems to be cumbersome for the adversary. Luckily the tool STP [25] can be applied to overcome this problem.

As a decision procedure for quantifier-free formulas with the data types of bit-vector and array, STP employs a series of word-level pre-processing algorithms to convert the original problem to a CNF formula and then solves it by invoking the default SAT solver CryptoMiniSat2 [28]. In order to fully exploit the speed of the SAT solver, optimizing transformations are employed to reduce the size and difficulty of the transformed problem.

STP has been used to search for optimal differential characteristics for ARX cipher Salsa20 [27]. In this paper, STP is exploited to solve the set of emerged multiple-variable equation systems. To use the tool, the adversary just needs to rewrite the expressions for every addition, rotation, XOR and Boolean function in the compression function according to the input requirements of STP. STP takes these 32-bit word based equation systems as input and converts them into a CNF formula, which is then solved by the invoked CryptoMiniSat2. If a satisfying solution to the CNF formula exits, the binary solution is converted to a word based solution for the original 32-bit variables by STP. If no solution is found, STP outputs “Valid”.

F. AFA on SHA-256 compression function

Since the chaining value H_l of *CF* cannot be computed directly, our attack is divided into two phases, which is similar to the attack procedure in [5]. Phase 1 aims to evaluate the input value p_{63} of step 63. In phase 2, the input message block M_l is revealed.

The experimental results of AFA on SHA-256 compression function are also presented in this section. The SHA-256 algorithm is implemented with the C language and the fault injections are simulated in software. STP with the default SAT solver CryptoMiniSat2 is running on a PC with Intel(R) Core(TM)2 Quad CPU Q8400 @2.66 GHZ, 2.67 GHZ, 2G memory, Ubuntu 13.10 32-bit desktop OS. During the attack procedure, the set of equation systems conforming to the input requirements of STP is automatically generated for the AFA attack using a program. This program has two parameters: the intermediate state register in which the faults are

injected and the number of injected faults. Hence, the adversary can adjust these two parameters to launch AFA on SHA-256 compression function. Note that in our simulations, when the state register in which the faults are injected is determined, the number of faults to be injected is adaptively chosen depending on the output of the AFA attack. Actually, it is a trade-off between the number of injected faults and the solving time.

Phase 1: Revealing p_{63}

To recover p_{63}, several random word faults are injected in a particular intermediate state register at a certain step of *CF*. In our attack, 14 faults are injected in c_{60} at step 60 respectively and the corresponding faulty compression values are obtained. After building the set of equation systems of last 4 steps (including the feed-forward operation) for the correct and faulty compression values, STP is invoked to solve the set of equation systems and outputs the correct solution for p_{63} within one minute. Furthermore, 100 different instances are tested with random word fault injections in order to evaluate the success rate of our attack. The correct value of p_{63} can be revealed in all the 100 instances. Figure 3 illustrates the statistical result of the solving time, where the execution time varies from 4.5 seconds to 626 seconds and more than 80% instances can output the solution within 100 seconds on average.

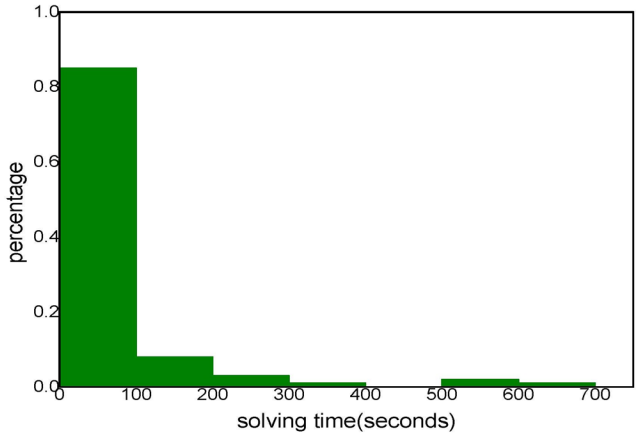

Figure 3: Statistics of the solving time with 14 faults in c_{60}

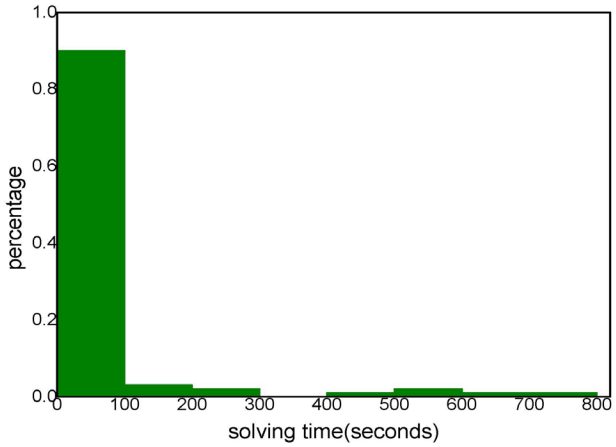

Figure 4: Statistics of the solving time with 13 faults in c_{59}

Extend to Deeper Fault Positions. Instead of injecting faults in c_{60} at step 60, faults can be induced in the deeper steps. As a direct instance, 13 random faults are first injected in c_{59} at step 59 to launch AFA. After building the set of equation systems of last 5 steps (59-63), STP is invoked and then outputs the correct value of p_{63}. Again, by testing 100 different instances, the attack could compute the correct solution with 58 seconds on average, and the maximal time is 786.6 seconds. The corresponding statistics of the execution time is presented in Figure 4. The attack can also be extended to step 58 by inducing 13 random faults in c_{58}. After testing 100 instances, the results show that our AFA still works though it would require a longer solving time. It takes an average of 8.58 hours to output a correct solution, and the exact solving time varies from 0.19 hours to 29.31 hours.

Phase 2: Revealing M_l

After phase 1, the right value of p_{63} is revealed. Combined with the right compression value Y and any faulty compression value Y^*, the corresponding faulty input p_{63}^* of step 63 can be computed as follows:

$$\begin{aligned}
a_{63}^* &= Y_b^* - Y_b + a_{63},\\
b_{63}^* &= Y_c^* - Y_c + b_{63},\\
c_{63}^* &= Y_d^* - Y_d + c_{63},\\
e_{63}^* &= Y_f^* - Y_f + e_{63},\\
f_{63}^* &= Y_g^* - Y_g + f_{63},\\
g_{63}^* &= Y_h^* - Y_h + g_{63},\\
h_{63}^* &= T_{63}^* - T_{63} + f_1(e_{63}^*, f_{63}^*, g_{63}^*)\\
&\quad - f_1(e_{63}, f_{63}, g_{63}) + h_{63},\\
d_{63}^* &= Y_e^* - Y_e - (T_{63}^* - T_{63}) + d_{63},
\end{aligned} \tag{5}$$

where $T_{63}^* = h_{63}^* + \sum_1(e_{63}^*) + Ch(e_{63}^*, f_{63}^*, g_{63}^*) + K_{63} + W_{63}$, $f_1(x,y,z) = \sum_1(x) + Ch(x,y,z)$.

Based on the input values of step 63, phase 2 can recover the input message block M_l of *CF*. The attack procedure is as follows:

- *Subphase 1*: 13 random word faults are injected in c_{56} at step 56 and the corresponding faulty compression values Y^* is obtained. Then the input value p_{63}^* is calculated for each Y^* as described above. Based on these input values, the set of equation systems of 7 steps (56-62) is constructed automatically. In our experiment, STP could output the correct values of four extended message words W_{59}, W_{60}, W_{61}, and W_{62}. 100 different instances have been tested and all of them output the correct values for these extended message words with an average of 34 seconds. The statistical result is shown in Figure 5.

- *Subphase 2*: after revealing the extended message words used in step 59 to step 62, another 13 random word faults are injected in c_{52} at step 52, and the 13 faulty compression values are obtained. Using the equation system (5), each faulty value p_{63}^{*} is calculated. Then these faulty values p_{63}^{*} and the right value p_{63} are decrypted with the recovered message words W_{59}, W_{60}, W_{61}, and W_{62} to obtain the output values of step 58. Thus the steps from 52 to 58 can be attacked according to the process in subphase 1, and the correct values of W_{55}, W_{56}, W_{57}, and W_{58} are recovered.

- *Subphase 3*: Repeat the above procedure by inducing faults in c_{48} and c_{44} sequentially. 16 extended message words $W_{47}, W_{48}, \cdots, W_{62}$ in total are recovered, which are then used to deduce M_l according to the message expansion algorithm. Therefore, the input message block M_l of *CF* is revealed.

With the value of M_l, the extended message word W_{63} of the last step can be computed. Based on the revealed p_{63} and W_{63}, p_{64} is calculated and $H_l = Y - p_{64}$ is the chaining value. By injecting about 65 random word faults, the secret inputs of SHA-256 compression function are extracted within about 200 seconds on average**.**

Based on the experimental results, it can be seen that SHA-256 compression function is very vulnerable to AFA. SHA-512 basically shares the same structure as SHA-256, except the 64-bit word size, more steps and the different rotation constants in σ_0, σ_1, $\sum_0$, and $\sum_1$. Hence, our AFA attack can be directly applied to SHA-512 compression function without the influence of these minor differences. SHA-224/384 is the truncated version of SHA-256/512. Since only partial compression values are obtained, AFA on SHA-224/384 compression function may need more faults injected and longer solving time to recover the secret inputs.

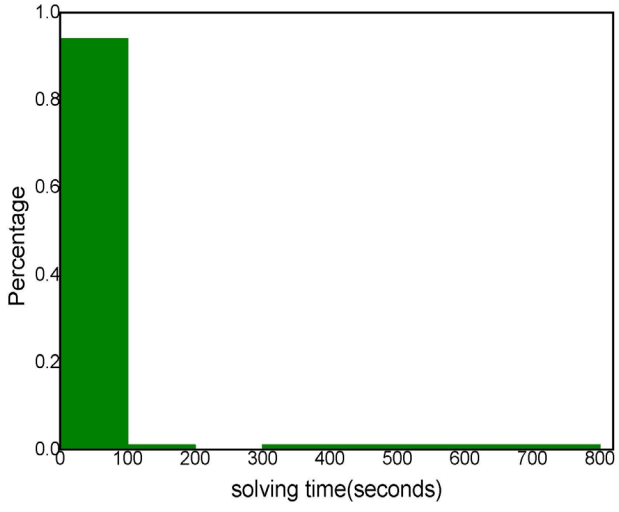

Figure 5: Statistics of the time with 13 faults in c_{56}

V. DISCUSSION

IV. ALMOST UNIVERSAL FORGERY ATTACK ON HMAC-SHA-256

In [18], Dunkelman et al. introduced the concept of an almost universal forgery attack, which is a very strong form of attack and is originally explained by [18] as follows.

"we can find in linear time and space the tag of essentially any desired message m chosen in advance, after performing a onetime precomputation in which we query the MAC on $2^{n/2}$ messages which are completely unrelated to m. The only sense in which this is not a universal forgery attack is that we need the ability to modify one message block in an easy to compute way."

In this attack, the adversary is free to modify partial information in *M*, i.e., at least one message block. Later, Y. Sasaki presented an almost universal forgery attack on LPMAC [20].

In our almost universal forgery attack, the adversary first chooses a random 447-bit message string. Before computing its tag, the string is padded to a 512-bit message block M_0 with one bit of "1" and 64-bit binary expression of the string length. Then the adversary accesses the device to compute the tag of M_0. During the calculation, he launches the above AFA attack on SHA-256 compression function CF_2 to reveal K_{out} and $H_{in} \parallel pad$, as shown in Figure 6. At the online stage, given any message *M* whose first block is equal to M_0, the corresponding tag for *M* can be computed with the values of K_{out} and H_{in}, i.e., an almost universal forgery on HMAC-SHA-256 is constructed.

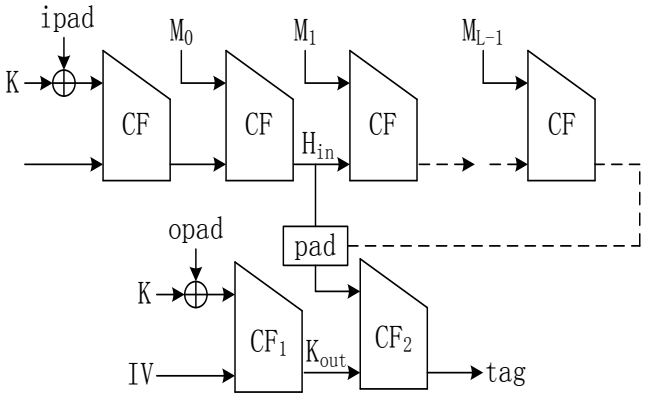

Figure 6: Almost universal forgery on HMAC-SHA-256

Remark. In order to recover the secret key *K* of HMAC-SHA-256, the adversary needs to launch AFA on SHA-256 compression function CF_1. This attack requires the ability of the adversary to reveal the faulty input chaining value K_{out}^{*} of CF_2 for each fault injection in advance, but this condition cannot be fulfilled in our attack model. Thus the secret key *K* cannot be recovered by exploiting the above AFA attack on SHA-256 compression function.

A. The effect on AFA by the structure of SHA-256

Similar to DFA, AFA is also affected by the property of difference propagation, especially for the generalized unbalanced Feistel structure in SHA-256. In order to examine how the injected faults in different positions affect the efficiency of AFA, each time 13 random word faults are induced in one state register from $\{a_{59}, b_{59}, \cdots, h_{59}\}$ at step 59, and the goal is to recover p_{63}. For each case, the recovery procedure is tested 100 times with different random fault injections. The output for each case is listed in Table 1. The results show that when injecting 13 faults in a_{59} and b_{59} respectively, the attack cannot output the correct solution for p_{63}. While the faults are induced in e_{59}, 4 sequent states with the corresponding message words can be revealed correctly.

Table 1: The output with 13 faults for each case

fault position	Nr. of faults	output states	output subwords
a_{59}	13	-	-
b_{59}	13	-	-
c_{59}	13	p_{63}	-
d_{59}	13	p_{63}, p_{62}, p_{61}	w_{62}, w_{61}
e_{59}	13	$p_{63}, p_{62}, p_{61}, p_{60}$	w_{62}, w_{61}, w_{60}
f_{59}	13	p_{63}, p_{62}	w_{62}
g_{59}	13	p_{63}, p_{62}	w_{62}
h_{59}	13	p_{63}, p_{62}, p_{61}	w_{62}, w_{61}

The statistics of solving time is illustrated in Figure 7, where the 4 colors varying from blue to green denote the percentage of the execution time less than 200 seconds, less than 400 seconds, less than 600 seconds and larger than 600 seconds respectively. As depicted in Figure 7, the percentage of the solving time larger than 200 seconds may be positively related to the number of computed states, i.e., the more states solved correctly, the more instances with execution time larger than 200 seconds. When launching AFA attack, injecting faults in the right 4 branches of SHA-256 step function will be more efficient according to the number of correct states obtained. That is, a even smaller number of faults in these positions is needed if the adversary only aims to reveal p_{63}.

B. AFA on SHA-1 compression function

The original DFA on SHA-1 compression function [5] consists of two phases: the computational elimination of the final addition phase and SHACAL1-phase. The AFA attack presented in this paper may be applied to improve this attack as well.

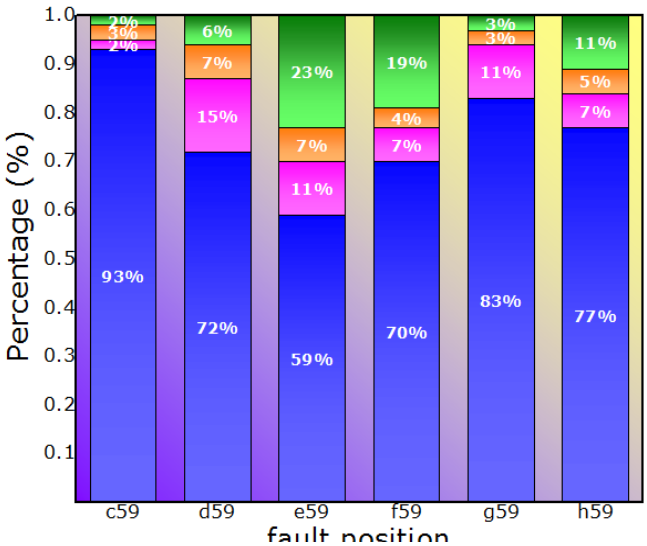

Figure 7: Statistics of the time with 13 faults for 6 cases in step 59

Firstly, the final addition may be eliminated by using AFA with less injected faults. Instead of compute the values of a_{79}, a_{78} and $b_{79} \oplus c_{79} \oplus d_{79}$, a_{77} and $b_{78} \oplus c_{78} \oplus d_{78}$ by injecting faults in different positions, AFA can take full advantage of the information induced by the injected faults in a deeper step to simultaneously calculate the values of b_{79}, c_{79} ,e_{79}, d_{79} and at least the highest five bits of a_{79}. Since the faults in the phase of eliminating the final addition in [5] accounts for a larger proportion in all faults needed, the total number of faults for the fault attack on SHA-1 compression function may be reduced significantly. Secondly, based on the known p_{79} and p_{79}^{*}, it is very probable that less fault injections are needed to improve the procedure of solving several sequent message words. Therefore, the DFA attack on SHA-1 compression function may be improved with AFA by reducing the number of faults injected and injecting the faults in deeper steps at the same time.

VI. CONCLUSION

In this paper, an efficient AFA attack on SHA-256 compression function is proposed. Due to the special structure of SHA-256, the fault attack on SHA-256 compression function will be more efficient and feasible by using algebraic techniques instead of differential analyses. Furthermore, the tool STP is used to launch the AFA attack automatically and efficiently by carefully modeling the operations in the compression function and fully exploiting the ability of the SAT solvers.

In order to show the feasibility of our attack, the AFA attack is simulated under a word-oriented random fault model. By only injecting about 65 faults, the secret inputs including the chaining value and the input message block of SHA-256 compression function could be revealed within minutes.

Based on the AFA attack on SHA-256 compression function, an almost universal forgery on HMAC-SHA-256 is also presented. It shows that applications using SHA-256 compression function should be protected against the threat of fault injections.

The AFA attack presented in this paper is generic, automatic and easy to be extended to other ARX-based primitives, such as other SHA-2 variants, MD5, SHA-1, HAS-160, SM3, Skein, Blake et al. These are work in progress.

Acknowledgements. We would like to thank the anonymous reviewers for their valuable comments and suggestions. This work was supported by the National Basic Research Program of China (973 Project, No.2013CB338002), the National High Technology Research and Development Program of China (863 Program, No.2013AA014002), the National Natural Science Foundation of China (No.61379137), the IIE's Cryptography Research Project (No.Y3Z0027103), and the Strategic Priority Research Program of Chinese Academy of Sciences under Grant XDA06010702.

II. REFERENCES

[1] M. Bellare, R. Canetti, H. Krawczy, "Keying Hash Functions for Message Authentication", 16th Annual International Cryptology Conference (CRYPTO'96), Springer Berlin Heidelberg 1996, LNCS 1109, pp. 1-15. doi: 10.1007/3-540-68697-5_1

[2] U.S. Department of Commerce, National Institute of Standards and Technology (2008). Announcing the SECURE HASH STANDARD (Federal Information Processing Standards Publication 180-3). [Online]. Available: http://csrc.nist.gov/publications/fips/fips180-3/fips180-3_final.pdf

[3] National Institute of Standards and Technology (March 2002). FIPS PUB 198. The Keyed-Hash Message Authentication Code (HMAC)

[4] F. Zhang, X. Zhao, S. Guo, T. Wang, Z. Shi, "Improved Algebraic Fault Analysis: A Case Study on Piccolo and Applications to Other Lightweight Block Ciphers", 4th International Conference on Constructive Side-Channel Analysis and Secure Design (COSADE'13), Springer Berlin Heidelberg 2013, LNCS 7864, pp. 62-79. doi: 10.1007/978-3-642-40026-1_5

[5] L. Hemme and L. Hoffman, "Differential Fault Analysis on the SHA1 Compression Function", 2011 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2011, pp 54-62. doi: 10.1109/FDTC.2011.16

[6] D. Boneh, R.A. DeMillo, R.J. Lipton, "On the Importance of Checking Cryptographic Protocols for Faults", International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'97), Springer Berlin Heidelberg 1997, LNCS 1233, pp. 37-51. doi: 10.1007/3-540-69053-0_4

[7] D. Boneh, R.A. DeMillo, R.J. Lipton, "On the Importance of Eliminating Errors in Cryptgraphic Computations", *Journal of Cryptography*, 14(2): pp. 101-119, 2001. doi: 10.1.1.42.7009

[8] E. Biham and A. Shamir, "Differential fault analysis of secret key cryptosystems", 17th International Cryptology Conference (CRYPTO '97), Springer Berlin Heidelberg 1997, LNCS 1294, pp. 513-525. doi: 10.1007/BFb0052259

[9] M. Hojsik and B. Rudolf, "Differential fault analysis of Trivium", 15th International Workshop of Fast Software Encryption (FSE'08). Springer Berlin Heidelberg 2008, LNCS 5086, pp. 158-172. doi: 10.1007/978-3-540-71039-4_10

[10] N. Courtois, D. Ware, K. Jackson, "Fault-Algebraic Attacks on Inner Rounds of DES", European Smart Card Security Conference (eSmart'10), 2010, pp. 22-24.

[11] R. Li, C. Li, C. Gong, "Differential fault analysis on SHACAL-1", 2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2009, pp 120-126. doi: 10.1109/FDTC.2009.41

[12] P. Dusart, G. Letourneux, O. Vivolo, "Differential Fault Analysis on A.E.S.", 1st International Conference of Applied Cryptography and Network Security (ACNS'03), Springer Berlin Heidelberg 2003, LNCS 2846, pp. 293-306.doi: 10.1007/978-3-540-45203-4_23

[13] J. Hoch and A. Shamir, "Fault Analysis of Stream Ciphers", 6th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'04), Springer Berlin Heidelberg 2004, LNCS 3156, pp. 41-51. doi: 10.1007/978-3-540-28632-5_18

[14] E. Biham, L. Granboulan, P. Q. Nguyen, "Impossible Fault Analysis of RC4 and Differential Fault Analysis of RC4," 12th International Workshop of Fast Software Encryption (FSE'05), Springer Berlin Heidelberg 2005, LNCS 3557, pp. 359-367. doi: 10.1007/11502760_24

[15] M. Agoyan, J.-M. Dutertre, D. Naccache, B. Robisson, A. Tria, "When clocks fail: On critical paths and clock faults", the ninth Smart Card Research and Advanced Application IFIP Conference (CARDIS'10), Springer Berlin Heidelberg 2010, LNCS 6035, pp. 182-193. doi: 10.1007/978-3-642-12510-2_13

[16] H. Bar-El, H. Choukri, D. Naccache, M. Tunstall, C. Whelan, "The Sorcerer's Apprentice Guide to Fault Attacks", Proceedings of the IEEE, vol.94, issue.2, pp. 370-382, doi: 10.1109/JPROC.2005.862424

[17] A. Barenghi, L. Breveglieri, I. Koren, D. Naccache, "Fault injection attacks on cryptographic devices: Theory, practice and countermeasures", Proceedings of the IEEE, vol.100, issue.11, pp. 3056-3076, doi: 10.1109/JPROC.2012.2188769

[18] O. Dunkelman, N. Keller, A. Shamir, "ALRED blues: New attacks on AES-based MACs", Cryptology ePrint Archive, Report 2011/095. [Online]. Available: http://eprint.iacr.org/2011/095

[19] M. Soos, K. Nohl, C. Castelluccia, "Extending SAT Solvers to Cryptographic Problems", Twelfth International Conference on Theory and Applications of Satisfiability Testing (SAT'09), Springer Berlin Heidelberg 2009, LNCS 5584, pp. 244-257. doi: 10.1007/978-3-642-02777-2_24

[20] Y. Sasaki, "Cryptanalyses on a Merkle- Damgård Based MAC—Almost Universal Forgery and Distinguishing-H Attacks", 32nd Annual International Conference on the Theory and Applications of Cryptographic Techniques (EUROCRYPT'12), Springer Berlin Heidelberg 2012,

LNCS 7237, pp. 411-427. doi: 10.1007/978-3-642-29011-4_25

[21] J. Kang, K. Jeong, J. Sung, S. Hong, "Differential Fault Analysis on HAS-160 Compression Function", The 4th FTRA International Conference on Computer Science and its Applications (CSA'12), Springer Berlin Heidelberg 2012, LNEE 203, PP. 97-105. doi: 10.1007/978-94-007-5699-1_11

[22] L. Wei, T. Zhi, G. Dawu, W. Yi, L. Zhiqiang, L. Ya, "Differential Fault Analysis on the MD5 Compression Function", *Journal of Computers*, 8(11): pp. 2888-2894, Nov 2013. doi:10.4304/jcp.8.11.2888-2894

[23] M. Hojsik and B. Rudolf, "Floating fault analysis of Trivium", 9th International Conference on Cryptology in India (INDOCRYPT'08), Springer Berlin Heidelberg 2008, LNCS 5365, pp. 239-250. doi: 10.1007/978-3-540-89754-5_19

[24] L. Song, L. Hu, "Differential Fault Attack on the PRINCE Block Cipher", Second International Workshop on Lightweight Cryptography for Security and Privacy (LightSec'13), Springer Berlin Heidelberg 2013, LNCS 8162, pp. 43-54. doi: 10.1007/978-3-642-40392-7_4

[25] V. Ganesh and D.L. Dill, "A Decision Procedure for Bit-Vectors and Arrays", 19th International Conference on Computer Aided Verification (CAV'07), Springer Berlin Heidelberg 2007, LNCS 4590, pp. 519-531. doi: 10.1007/978-3-540-73368-3_52

[26] X. Zhao, S. Guo, F. Zhang, Z. Shi, C. Ma, T. Wang, "Improving and Evaluating Differential Fault Analysis on LED with Algebraic Techniques", 2013 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2013, pp 120-126. doi: 10.1109/FDTC.2013.14

[27] N. Mouha, and B. Preneel, "Towards Finding Optimal Differential Characteristics for ARX: Application to Salsa20", Cryptology ePrint Archive, Report 2013/359. [Online]. Available: http://eprint.iacr.org/2013/359.pdf

[28] M. Soos (2013), "CryptoMiniSat2". [Online]. Available: http://www.msoos.org/cryptominisat2/

[29] M. Mohamed, S. Bulygin, J. Buchmann, "Improved Differential Fault Analysis of Trivium", Second International Workshop on Constructive Side-Channel Analysis and Secure Design (COSADE'11), 2011, pp. 147-158.

[30] C. Bouillaguet, P. Derbez, P.-A. Fouque, "Automatic Search of Attacks on Round-Reduced AES and Applications", 31st International Cryptology Conference (CRYPTO'11), Springer Berlin Heidelberg 2011, LNCS 6841, pp. 169-187. doi: 10.1007/978-3-642-22792-9_10

[31] P. Jovanovic, M. Kreuzer, I. Polian, "An Algebraic Fault Attack on the LED Block Cipher", Cryptology ePrint Archive, Report 2012/400. [Online]. Available: http://eprint.iacr.org/2012/400.pdf

[32] N.T. Courtois, J. Pieprzyk, "Cryptanalysis of Block Ciphers with Overdefined Systems of Equations", 8th International Conference on the Theory and Application of Cryptology and Information Security (ASIACRYPT'02), Springer Berlin Heidelberg 2002, LNCS 2501, pp. 267-287. doi: 10.1007/3-540-36178-2_17

[33] G. Piret, J.-J. Quisquater, "A Differential Fault Attack Technique against SPN Structures, with Application to the AES and KHAZAD", 5th International Workshop on Cryptographic Hardware and Embedded Systems (CHES'03), Springer Berlin Heidelberg 2003, LNCS 2779, pp. 77-88. doi: 10.1007/978-3-540-45238-6_7

[34] M. Tunstall, D. Mukhopadhyay, S. Ali, "Differential Fault Analysis of the Advanced Encryption Standard Using a Single Fault", 5th IFIP WG 11.2 International Workshop on Security and Privacy of Mobile Devices in Wireless Communication (WISTP'11), Springer Berlin Heidelberg 2011, LNCS 6633, pp. 224-233. doi: 10.1007/978-3-642-21040-2_15

[35] K. Jeong and C. Lee, "Differential Fault Analysis on Block Cipher LED-64", 7th FTRA International Conference on Future Information Technology (FutureTech'12), Springer Berlin Heidelberg 2012, LNEE 164, pp. 747-755. doi: 10.1007/978-94-007-4516-2_79

[36] P. Jovanovic, M. Kreuzer, I. Polian, "A Fault Attack on the LED Block Cipher", 3rd International Conference on Constructive Side-Channel Analysis and Secure Design (COSADE'12), Springer Berlin Heidelberg 2012, LNCS 7275, pp. 120-134. doi: 10.1007/978-3-642-29912-4_10

[37] Jeong, K.: Differential Fault Analysis on Block Cipher Piccolo. Cryptology ePrint Archive, Report 2012/399. [Online]. Available: http://eprint.iacr.org/2012/399.pdf

[38] K. Jeong, Y. Lee, J. Sung, S. Hong, "Security Analysis of HMAC/NMAC by Using Fault Injection", *Journal of Applied Mathematics*, 2013(17). doi: 10.1155/2013/101907

[39] W. Fischer and C.A. Reuter, "Differential Fault Analysis on Grøstl", 2012 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC), 2012, pp. 44-54. doi: 10.1109/FDTC.2012.14

How to cite

Ronglin Hao, Bao Li, Bingke Ma, Ling Song, "*Algebraic Fault Attack on the SHA-256 Compression Function*". *International Journal of Research in Computer Science*, 4 (2): pp. 1-9, March 2014. doi: 10.7815/ijorcs.42.2014.079