The Insecurity of SHA2 under the Differential Fault Characteristic of Boolean Functions

Weiqiong Cao[1], Hua Chen[1], Hongsong Shi[2], Haoyuan Li[3], Jian Wang[1,4]

[1] Trusted Computing and Information Assurance Laboratory, Institute of Software, Chinese Academy of Sciences, Beijing 100190, China.
[2] China Information Technology Security Evaluation Center, Beijing 100085, China.
[3] Zhongguancun Laboratory, Beijing, China.
[4] University of Chinese Academy of Sciences, Beijing 100049, China.
Email: caoweiqiong@iscas.ac.cn, chenhua@iscas.ac.cn

Abstract.
SHA2 is widely used in various traditional public key cryptosystems, post-quantum cryptography, personal identification, and network communication protocols. Therefore, ensuring its robust security is of critical importance. Several differential fault attacks based on random word fault have targeted SHA1 and SHACAL-2. However, extending such random word-based fault attacks to SHA2 proves to be much more difficult due to the increased complexity of the Boolean functions in SHA2.
In this paper, assuming random word fault, we identify distinctive differential properties within the Boolean functions of SHA2. Based on these findings, we propose a novel differential fault attack methodology that can be effectively used to recover the final message block and its corresponding initial vector in SHA2, forge HMAC-SHA2 messages, extract the key of SHACAL-2, and extend our analysis to similar algorithms such as SM3. The efficacy of these attacks is validated through rigorous simulations and theoretical deductions, illustrating that they represent a considerable threat to the security of SHA2. In simulations, our approach only requires guessing T bits of a register, where T is at most 5. Moreover, the probability of successfully recovering a register (excluding the guessed bits) approaches 100% when introducing 15 faults (in 1000 instances), and the approximate probability is at least 95% when $T = 1$. Consequently, approximately 928 random faults are necessary to successfully execute the attack on the compression function. Furthermore, we discuss potential countermeasures, including verification and infection detection, and propose methods to determine the time and location of fault injection in practical experiments.

Keywords: SHA2 · Differential Fault Attack · Boolean Function · HMAC · Hash Function

1 Introduction

1.1 Background

SHA2 (Secure Hash Algorithm 2) family [Nat02], which serves as an advanced successor to SHA1 algorithm, was officially introduced by the National Institute of Standards and Technology (NIST) in 2001. This suite comprises four variants: SHA224, SHA256, SHA384 and SHA512, all of which share a common compression function, while differing in terms of data block length. The compression function of SHA2 plays a pivotal role within the SHACAL block cipher algorithm [BJ01] and HMAC (Hash-based Message Authentication Code) [BCK96]. Meanwhile, SHA2 has been widely used in various public key cryptosystems, personal identification and network communication protocols. In particular, SHA256 and SHA512 have been widely adopted as key derivation functions in both traditional public key cryptography (such as ECDSA and EdDSA) and emerging post-quantum cryptography (such as Dilithium and Kyber). Moreover, SHA2 has become a standard for authentication mechanisms within numerous secure communication protocols, including TLS/SSL, IPsec, SSH (Secure Shell), PGP, PIN verification, OTP (one-time password) systems, and others.

Despite the limited public information about its design, SHA2 is considered a more secure alternative to MD5 and SHA1 in terms of resisting various attacks, including collision attacks [WYYD05] and differential attacks [SKP13]. Nevertheless, the computational complexity renders many of these theoretical attacks impractical in real scenarios. Over the past decade, research has increasingly focused on the implementation security of hash algorithms, with particular attention paid to side-channel and fault attacks.

Side-channel attacks on HMAC implementations have been identified. McEvoy et al. proposed a Differential Power Analysis (DPA) on HMAC-SHA2 [MTMM07], which uses the leakage of the modular *ADDs* and *XORs* to reveal the secret initial state in HMAC-SHA2 and perform a forgery attack. Meanwhile, Fouque etc. proposed a template attack on HMAC [FLDV09], which can recover the secret key k by measuring a single execution of HMAC-SHA1. These attacks are practical and have been demonstrated many times in real products [Osw16, GWM16, SBB^{+}18]. As another category of physical attacks, fault attacks [LLG09, HH11, Sho13] have also been demonstrated to be effective against hash algorithms. In general, the adversary intends to perturb the registers and the corresponding Boolean functions in the compression function (by means of voltage glitches, laser or electro-magnetic injection, etc.) in order to generate faulty results and exploit them to perform input recovery.

1.2 Previous Works

The known fault attacks against hash algorithms and their derivatives are primarily based on difference methodologies, including Differential Fault Attacks (DFA) and Algebraic Fault Attacks (AFA). These attacks typically assume a random fault model.

The first fault attack (i.e., DFA) based on the 32-bit random word fault model, was introduced in FDTC 2009 [LLG09], which employs the arithmetical differential relation between the faulty and correct Boolean function $f(x, y, z) = x \oplus y \oplus z$ (with the same message) to manually recover the key in SHACAL-1. It is noteworthy that the attack relies on the adversary's knowledge of the final round output rH, which differs from the output H of SHA1. Consequently, this attack is not feasible for hash algorithms that include a final addition operation. After that, Hemme and Hoffmann improved upon the aforementioned attack in FDTC 2011 [HH11]. They have developed a novel approach targeting the cyclic shifts in the compression function of SHA1, thereby enabling the recovery of the input to the compression function. However, due to the increased complexity of the Boolean functions in SHA2, the DFA techniques presented in [LLG09, HH11] cannot be directly applied to SHA2. Analogous DFA techniques based on random word faults have been applied to SHACAL-2 [WLLL10, She14] as well. Such attacks not only require knowledge of both the correct and faulty round outputs in the final round of SHACAL-2 (i.e., knowing every Boolean differential value and the initial vector), but they also fundamentally involve searching for the key by exploiting Boolean and arithmetic difference pairs. These types of attacks are also not feasible for SHA2. In addition, Bagheri et al. first introduced a DFA against SHA3 in INDOCRYPT 2015 [BGS15]. With only 80 faulty outputs, they were able to recover 1592 bits (out of the total 1600 bits) of the internal state. Subsequently, Luo et al. [LFZ^{+}16] improved upon this method in FDTC 2016 by extending the bit fault model to byte fault model. However, due to the fundamentally different compression structures between SHA2 and SHA3 (sponge construction), the DFAs against SHA3 are not directly applicable to the analysis of SHA2. To the best of our knowledge, the round-counter fault attacks [Sho13, JLSH13] are the only currently feasible fault attacks on SHA2. These attacks reduce the number of rounds or bypass the round counter by fault injection to make the total rounds fewer than 16, which can recover message blocks directly. However, such attacks require extremely precise timing and positioning for fault injection. Furthermore, a low-cost countermeasure, such as checking the round counter, can effectively prevent these attacks.

Since then, researchers have turned their attention to the study of other fault attacks, in particular AFA. AFA does not require the theoretical analysis of differential paths within hash algorithms as DFA does. Instead, it formulates algebraic equations based on both faulty and correct outputs, which are then processed by tools such as SAT solvers to uncover the message block or the initial state of the compression function. The results of AFA are obtained experimentally rather than derived strictly theoretically. The AFAs presented in [HLMS14, NHGG18] are capable of forging the initial state of HMAC-SHA2. In addition, Luo et al. in DATE 2017 [LAFW17] first proposed an AFA on SHA3. They subsequently extended this work in [LAFW18] to accommodate more extensive fault models with multi-bit faults. Since AFA utilizes all the correct and faulty hash outputs, it requires significantly fewer faults than DFA (which typically uses only one or two hash outputs at each step). However, it typically exhibits a lower success rate due to constraints on the search efficiency of SAT solvers and the complexity of the constructed algebraic equations. Additionally, it is considerably more time-consuming due to the absence of a theoretical framework that clarifies the differential properties intrinsic to the compression function. Furthermore, if any part of the hash output is unknown, the success rate will be drastically reduced or even rendered ineffective, and the computational complexity will be multiplied accordingly. Therefore, AFA often serves as a complementary approach to DFA and becomes advantageous when specific differential paths cannot be derived manually by DFA.

To sum up, no reported DFA exploits vulnerabilities in the compression function of SHA2, particularly in view of its final addition step. Therefore, the security of SHA2 under the random word fault model remains an open question and requires further investigation.

1.3 Our Contributions

In this paper, based on the random word fault model, we established several differential properties of the Boolean functions ($Ch(x, y, z) = (x \wedge y) \oplus (x\prime \wedge y)$ and $Maj(x, y, z) = (a \wedge b) \oplus (a \wedge c) \oplus (c \wedge b)$) in the compression function of SHA2. By exploiting these properties, we proposed a novel differential fault attack strategy targeting the compression function of SHA2 based on the random word fault model. We successfully performed such an attack on SHA2, HMAC-SHA2, SHACAL-2 and SM3 [Sta16] (which have similar Boolean functions) and demonstrated its effectiveness. The main advantages of our proposed differential approach can be summarised as follows:

- The proposed approach exploits previously undiscovered differential properties of Boolean functions and has a lower computational complexity. The Boolean functions $Ch(x, y, z)$ and $Maj(x, y, z)$ have posed a challenge to differential analysis on SHA2 due to their complex operations. We introduce novel transformations of Boolean functions and thereby uncover some differential properties, which can be employed to ascertain the intermediate state registers in the compression function. In contrast to previous DFAs on SHA1, which required guessing 16 bits of the targeted state register, our approach only requires guessing T bits, where $T \leq 5$. A detailed explanation is given in Section 3.

- Our methodology is broadly applicable to numerous hash functions and cryptographic algorithms with similar Boolean functions, such as SHA2 (SHA224/256/384/512), HMAC-SHA2, SHACAL-2 and SM3. The methodology was successfully employed to recover final message blocks and initial vectors in SHA2, forge HMAC-SHA2 messages, and retrieve SHACAL-2 secret keys. Specially, we have confirmed that SM3's Boolean functions are indeed functionally equivalent to $Ch(x, y, z)$ and $Maj(x, y, z)$. See section 4 for detailed case studies.

- Our attacks exhibit remarkably high success rates and exceptionally low time consumption, supported by rigorous theoretical analysis and experimental evidence. Experiments show that inducing 15 random word faults ($N = 15$) achieves a success rate of nearly 100% (across 1000 instances) in recovering one register (or the XOR value of two registers). Additionally, there is a success rate of approximately 99% (and at least 95% when $T = 1$) for recovering the final round input with $14N$ faults, with a time consumption of less than 0.001 seconds, when $T \leq 5$. Tables 1 and 2 provide more detailed experimental results and comparisons.

The comparison of our attacks with previous DFAs [LLG09, HH11, WLLL10, She14, Sho13, JLSH13] is shown in Table 1. The column "Number of Faults" indicates the minimum number of faults required to achieve a success probability of at least 98% (over 1000 attack instances), excluding the guessing step. In addition, Table 2 provides a comparison between our DFA and previous AFAs on SHA256 [HLMS14, NHGG18]. Although a much larger number of faults are required, our attack exhibits a obvious advantage in terms of time complexity and success rate in recovering the message block and initial vector (in 100 instances). Furthermore, our attack remains feasible even when only the left half of the hash values are output (see column 6 in Table 2).

Table 1: Comparison with previous DFAs

Attacks	Fault model	Complexity of guessing	Number of faults	Algorithms of analysis
Our attack	Random word fault	$2^T (T \approx 1)$	928	SHA2,SM3 and HMAC-SHA2
		–	720	SHACAL-2
[LLG09]	Random word fault	2^{16}	120	SHACAL-1
[HH11]	Random word fault	2^{16}	1000	SHA1, SHACAL-1
[WLLL10]	Random word fault	–	128	SHACAL-2
[She14]			240	
[Sho13] [JLSH13]	Round counter fault	–	16	SHA2 SHACAL-2

Table 2: Comparison with previous AFAs on SHA256

Attacks	Time consumption(s)	Success rate	Needed faults	Solving Method	All hash values required
Our attack	0.016	98%	928	Deduction	×
[NHGG18]	43200	–	48	Search	✓
[HLMS14]	200	80%	65	Search	✓

The remainder of this paper is organized as follows: In Section 2, we present the fundamental theory of Boolean operations and the architectural design of both SHA2 and HMAC-SHA2. Section 3 presents the core concepts of the differential theory applied to Boolean functions, along with the associated proofs. Section 4 provides concrete examples of differential attacks on SHA2, HMAC-SHA2, SHACAL-2, and SM3, as well as a discussion of potential countermeasures. Finally, in Section 5, we present the experimental results that validate the effectiveness of the proposed key recovery method.

2 Preliminaries

2.1 Notations

Throughout the paper, we define the notations listed in Table 3.

Table 3: Notations

Symbols	Definition
n	The bit length of one state register
N	The Number of faults
T	The number of bits required to guess when recovering one register
$\mathbb{Z}_2^n$	The set of integers with bit length n
$\oplus$	Exclusive-OR (XOR)
$\cdot$ or &	Logical bitwise AND, i.e., boolean AND
$'$	Logical bitwise NOT
$+$	Addition modulo 2^n when the operators belong to $\mathbb{Z}_2^n$; Logical bitwise OR when the operators belong to $\in \mathbb{Z}_2$
$\vee$	32-bit logical bitwise OR
xy	Abbreviation of $x \cdot y$. Higher priority than XOR and OR
x_i	The i-th bit value of n-bit x
x^j	The faulty value ($\in \mathbb{Z}_2^n$) derived by j-th fault injection
$\sum\limits_{j=0}^{N-1} x_i^j$	$x_i^0 + \ldots + x_i^{N-1}$. Equals to the sum of logical OR ($x_i^j \in \mathbb{Z}_2$)
$-$	Subtraction modulo 2^n
$\ggg t$	Circular right shift by t positions
$x \parallel y$	Concatenation of two operators x and y
$Ch(x,y,z)$	$(x\&y) \oplus (x'\&z)$, where $x,y,z \in \mathbb{Z}_2^n$.
$Maj(x,y,z)$	$(x\&y) \oplus (x\&z) \oplus (y\&z)$, where $x,y,z \in \mathbb{Z}_2^n$
$\sum_0(x)$, $\sum_1(x)$, $\sigma_0(x)$, $\sigma_1(x)$	$(x \ggg i) \oplus (x \ggg j) \oplus (x \ggg k)$, where $x \in \mathbb{Z}_2^n$ is n-bit operator, and i, j and k depend on n
$H(M, IV)$	Hash function with message M and initial vector IV
$F(V_i, M^i)$	The i-th compression function in $H(M, IV)$ with initial vector V_i and message block M^i
r	The number of message blocks
R	The round number of compression function
$f(V_{i,j}, W_j)$	The j-th round of transformation in the i-th $F(V_i, M^i)$
$P(X)$	The probability of event X
Random number X	Each bit of X ($\in \mathbb{Z}_2^n$) follows uniform bernoulli distribution, i.e., $P(X_i = 1) = \frac{1}{2}$ and $P(X_i = 0) = \frac{1}{2}$
P_i	The average probability of successfully recovering the i-th bit of the register x in Lemma 4

2.2 SHA2 family

The following is a brief overview of the SHA2 hash algorithms (SHA224, SHA256, SHA384 and SHA512), which all share the same compression function design. SHA224/SHA256 use a 32-bit word size (i.e., $n = 32$) and 64 rounds (i.e., $R = 64$), while SHA384/SHA512 use a 64-bit word size and 80 rounds. The full hash function $H(M, IV)$, which takes a padded message $M = M^0||M^1||...||M^{r-1}$ and an initial vector IV as input, is shown in Figure 1. Here M^i ($i \in \mathbb{Z}_r$) is the i-th message block, F is the compression function of SHA2 and V_i is the input to the i-th function $F(V_i, M^i)$ (where $V_0 = IV$).

2.2.1 Message Extension

As illustrated in Figure 1, the message block M^i is firstly divided into sixteen n-bit words $W_0, ..., W_{15}$ (i.e., $M^i = W_0||...||W_{15}$), and then extended to R words W_j using the following equation:

$$W_j = \sigma_1(W_{j-2}) + W_{j-7} + \sigma_0(W_{j-15}) + W_{j-16}\,(16 \le j < R)\,. \tag{1}$$

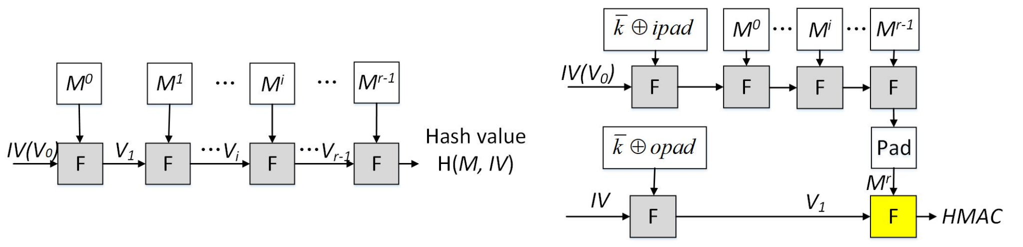

Figure 2: Structure of HMAC

2.2.2 Compression function

As illustrated in Figure 3, the compression function F employs an R-round iterative transformation, denoted as $f(V_{i,j}, W_j)(j = 0, ..., R-1)$, which operates on the internal state for every round j from 0 to $R-1$. The internal state $V_{i,j}$ $(i = 0, ..., r-1)$ is composed of eight concatenated n-bit word registers: $a_j, b_j, c_j, d_j, e_j, f_j, g_j, h_j (j = 0, ..., R)$, such that $V_{i,j} = a_j||...||h_j$. The initial state is defined as a constant initial vector, denoted by V_0. That is, $V_0 = V_{0,0} = a_0||...||h_0$. The transformation function f comprises several Boolean functions, including $Maj(x, y, z)$, $Ch(x, y, z)$, $\sum_0(x)$, $\sum_1(x)$ and the additions modulo 2^n defined in Table 3. IK_j represents a predefined constant for each round. After running R rounds of the transformation function f, a final modulo 2^n addition is performed between the initial input vector V_i and the R-th round output, denoted as $V_{i,R}$, to derive V_{i+1}, which serves as the input for the next compression function. Finally, the final state V_r is the hash result of $H(M, IV)$.

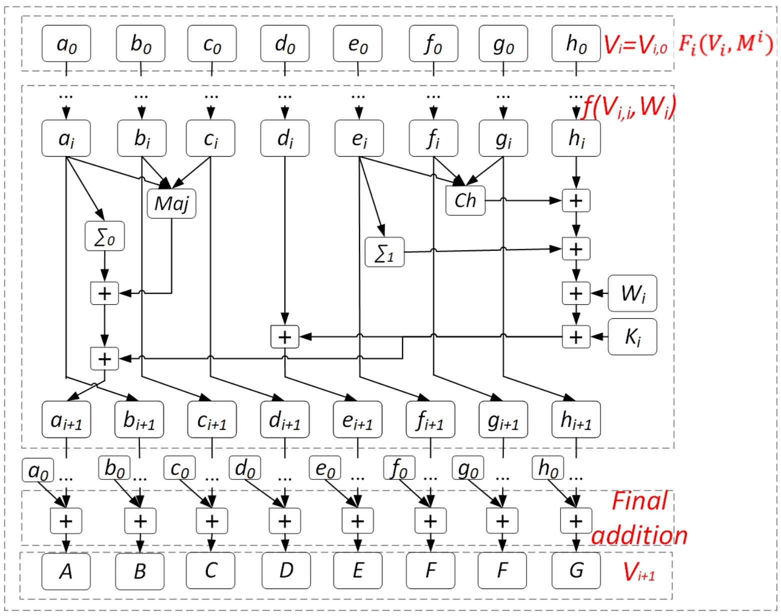
a_0 b_0 c_0 d_0 e_0 f_0 g_0 h_0
V_i=$V_{i,0}$
$F_i(V_i, M^i)$
$f(V_{i,i}, W_i)$
a_i b_i c_i d_i e_i f_i g_i h_i
Maj
Ch
Σ_0
Σ_1
W_i
K_i
a_{i+1} b_{i+1} c_{i+1} d_{i+1} e_{i+1} f_{i+1} g_{i+1} h_{i+1}
Final addition
A B C D E F F G
V_{i+1}

Figure 3: Compression function of SHA2

In our subsequent analysis, we will focus on two specific Boolean functions within the round transformation $f(V_{i,j}, W_j)$, i.e., $Ch(e_j, f_j, g_j)$ and $Maj(a_j, b_j, c_j)$.

2.3 HMAC of SHA2

HMAC is formulated as follows:

$$HMAC(M,k,IV) = H(\bar{k} \oplus opad \| H(\bar{k} \oplus ipad \| M, IV), IV),$$

where M represents the message requiring authentication, k denotes the secret key, $\bar{k}$ denotes the padded form of k (padded with zeros to a specified length), and *ipad* are distinct $8n$-bit padding strings for the HMAC steps. []As shown in Fig. 2, there are two hash functions. First, an inner hash function is processed by inputting $\bar{k}$, *ipad*, the message M and IV. The output of this hash is subsequently hashed again along with $\bar{k}$, *opad* and IV. The utilization of dual hash functions serves to ensure the integrity and authenticity of the message.

2.4 Boolean Theorems

The Boolean operations, which are denoted by the symbols "$\oplus$", "$\cdot$" , "$\prime$" and "$+$" (as defined in Table 3) exhibit the following distinctive properties when their operands are single-bit.

Theorem 1 (Operational rules)**.** *$x, y, z \in \mathbb{Z}_2$ and satisfy:*

1. *Distributive law.*
   (a) $(x \oplus y) \cdot z = (x \cdot z) \oplus (y \cdot z)$.
   (b) $(x + y) \cdot z = (x \cdot z) + (y \cdot z)$.
2. *Absorption law.* $x + xy = x$, $x + x\prime y = x + y$.
3. $(x \cdot y)\prime = x\prime + y\prime$, $(x + y)\prime = x\prime \cdot y\prime$, $(x \oplus y)\prime = x\prime y + y\prime x$.

Theorem 2 (Transformation between boolean and arithmetic operations)**.** *Given $x, y \in \mathbb{Z}_2^n$, there exists a transformation defined as follows:*

$$x + y = (x \oplus y) + ((x \cdot y) \lll 1)\,.$$

3 Methodology of Attack

This section presents some differential fault properties of the Boolean functions in SHA2, which are used to determine the intermediate state registers. Subsequently, we present a differential fault attack methodology on SHA2 that can effectively recover the initial vector V_{r-1} and message block M^{r-1} in the final compression function.

3.1 Differential Fault Properties of Boolean Functions

Lemma 1. *If $x \oplus \sigma = x + \Delta$, where $x, \sigma, \Delta \in \mathbb{Z}_2^n$, then the bits of x, σ and Δ satisfy*

$$\sigma_i \cdot (\Delta_i \oplus x_i) = \sigma_{i+1} \oplus \Delta_i \oplus \Delta_{i+1} \quad \textit{for} \quad i \in \mathbb{Z}_{n-1}$$

and $\sigma_0 = \Delta_0$.

Proof. Let c be the carry generated during the addition modulo 2^n of x and Δ. Then, the bits of c satisfy

$$c_i = \begin{cases} 0 \quad i = 0 \\ (c_{i-1}\Delta_{i-1}) \oplus (c_{i-1} \oplus \Delta_{i-1})\, x_{i-1} \quad 0 < i < n \end{cases}$$

and

$$x_i \oplus \sigma_i = x_i \oplus \Delta_i \oplus c_i \text{ for } 0 \leq i < n.$$

Hence,

$$c_i = \sigma_i \oplus \Delta_i \text{ (where } \Delta_0 = \sigma_0\text{)}$$

and

$$c_{i+1} = \Delta_{i+1} \oplus \sigma_{i+1} = c_i \Delta_i \oplus (c_i \oplus \Delta_i) x_i.$$

Replacing c_i with $\sigma_i \oplus \Delta_i$ in the equation above, we obtain

$$\sigma_i (\Delta_i \oplus x_i) = \sigma_{i+1} \oplus \Delta_i \oplus \Delta_{i+1} \tag{2}$$

□

Lemma 2. *Given* $\begin{cases} x \oplus \sigma^j = x + \Delta^j \\ x \oplus e\sigma^j = x + \Lambda^j \end{cases}$ *for* $j \in \mathbb{Z}_N$*, where* $\sigma^j \in \mathbb{Z}_2^n$ *is an unknown random number,* $\Delta^j, \Lambda^j \in \mathbb{Z}_2^n$ *are known values and* $x, e \in \mathbb{Z}_2^n$ *are unknown values, each bit* e_i *of* e *for* $i \in \mathbb{Z}_n$ *can be determined with probability* $1 - (\frac{1}{2})^N$*.*

Proof. From *Lemma 1*, we have the following equations (3) and (4) for $j \in \mathbb{Z}_N$ and $i \in \mathbb{Z}_n$, where $A_i^j = \Delta_{i+1}^j \oplus \Delta_i^j$ and $B_i^j = \Lambda_{i+1}^j \oplus \Lambda_i^j$.

$$\sigma_i^j \left(\Delta_i^j \oplus x_i\right) = \sigma_{i+1}^j \oplus A_i^j \tag{3}$$

$$\sigma_i^j e_i \left(\Lambda_i^j \oplus x_i\right) = \sigma_{i+1}^j e_{i+1} \oplus B_i^j \tag{4}$$

Hence, we can deduce

$$(\sigma_i^j)\prime\sigma_{i+1}^j = (\sigma_i^j)\prime A_i^j, \tag{5}$$

$$\sigma_{i+1}^j e_i\prime e_{i+1} = B_i^j e_i\prime, \tag{6}$$

$$\sigma_{i+1}^j e_i e_{i+1}\prime = e_i \alpha_{i+1}^j \oplus e_i (\sigma_i^j)\prime \alpha_i^j, \tag{7}$$

where $\alpha_i^j = \Delta_i^j \oplus \Lambda_i^j$ and $\alpha_i^j \oplus \alpha_{i+1}^j = A_i^j \oplus B_i^j$. Equation (7) is obtained by multiplying Equation (3) and Equation (4) individually by e_i, and then eliminating x_i.

By the equations (5), (6) and (7), the following relations hold.

1. **Case 1: Determine e_0.**
   From *Lemma 1*, we have $\sigma_0^j = \Delta_0^j$ and $\sigma_0^j e_0 = \Lambda_0^j$ for $j = 0, ..., N-1$. Hence, $\left(\sum\limits_{j=0}^{N-1} \sigma_0^j\right) e_0 = \sum\limits_{j=0}^{N-1} \Lambda_0^j$. Due to the randomness of σ_0^j, $e_0 = \sum_{j=0}^{N-1} \Lambda_0^j$ holds with probability $1 - (\frac{1}{2})^N$.

2. **Case 2: Determine e_{i+1} when $e_i = 0$.**
   If $e_i = 0$, then $\sigma_{i+1}^j e_{i+1} = B_i^j$(by equation (6)). Hence, $\left(\sum\limits_{j=0}^{N-1} \sigma_{i+1}^j\right) e_{i+1} = \sum\limits_{j=0}^{N-1} B_i^j$.
   Similarly, $e_{i+1} = \sum\limits_{j=0}^{N-1} B_i^j$ holds with probability $1 - (\frac{1}{2})^N$ when $e_i = 0$.

3. **Case 3: Determine e_{i+1} when $e_i = 1$.**
   When $e_i = 1$, we have $\sigma_{i+1}^j e_{i+1}\prime = \alpha_{i+1}^j \oplus \sigma_i^j\prime \alpha_i^j$ (from equation (7)).
   If $i = 0$, then $\sigma_0^j = \Delta_0^j$ and $\sigma_1^j e_1\prime = \alpha_1^j \oplus \Delta_0^j\prime \alpha_0^j$.
   If $i > 0$, there are two cases:

(a) If $e_{i-1} = 0$, then $\sigma_i^j = B_{i-1}^j$ (from equation (6));

(b) If $e_{i-1} = 1$, then $\alpha_i^j = \sigma_{i-1}^j{}'\alpha_{i-1}^j$. Hence,

  i. When $\alpha_i^j = 0$: $\sigma_{i+1}^j e_{i+1}' = \alpha_{i+1}^j$;

  ii. When $\alpha_i^j = 1$: $\sigma_{i-1}^j = 0$ and $\sigma_i^j = A_{i-1}^j$ (from equation (5)).

Consequently,

$$e_{i+1} = \begin{cases} \left(\sum\limits_{j=0}^{N-1} \left(\alpha_{i+1}^j \oplus B_{i-1}^j{}'\alpha_i^j\right)\right)' & e_{i-1} = 0, e_i = 1, i > 0 \\ \left(\sum\limits_{j=0,\alpha_i^j=0}^{N-1} \left(\alpha_{i+1}^j\right) + \sum\limits_{j=0,\alpha_i^j=1}^{N-1} \left(\alpha_{i+1}^j \oplus A_{i-1}^j{}'\alpha_i^j\right)\right)' & e_{i-1} = 1, e_i = 1, i > 0 \\ \left(\sum\limits_{j=0}^{N-1} \left(\alpha_1^j \oplus \Delta_0^j{}'\alpha_0^j\right)\right)' & e_i = 1, i = 0 \end{cases}$$

holds with probability $1 - (\frac{1}{2})^N$ when $e_i = 1$.

To sum up above, with probability $1 - (\frac{1}{2})^N$ we can get each bit e_i of e from **Cases 1-3**.

□

The lemma above suggests that, if all bits of x have been flipped at least once during N fault injections, the value of e can be uniquely determined with probability $(1 - (\frac{1}{2})^N)^n$.

Lemma 3. *Given* $\begin{cases} x \oplus e\sigma^j = x + \Lambda^j \\ x \oplus e'\varepsilon^j = x + \Psi^j \end{cases}$ *for* $j = 0, \ldots, N-1$, *where* $\sigma^j, \varepsilon^j \in \mathbb{Z}_2^n$ *are unknown random numbers,* $\Lambda^j, \Psi^j \in \mathbb{Z}_2^n$ *are known values, and* $x, e \in \mathbb{Z}_2^n$ *are unknown values, each bit* e_i *of* e *for* $i = 0, \ldots, n-1$ *can be determined with probability* $1 - (\frac{1}{2})^N$.

Proof. From *Lemma 1* and equation (6), we have

$$\begin{array}{c} \sigma_{i+1}^j e_i' e_{i+1} = B_i^j e_i' \\ \varepsilon_{i+1}^j e_i e_{i+1}' = C_i^j e_i, \end{array} \tag{8}$$

where $B_i^j = \Lambda_{i+1}^j \oplus \Lambda_i^j$ and $C_i^j = \Psi_{i+1}^j \oplus \Psi_i^j$.

Hence,

$$e_{i+1} = \begin{cases} \sum\limits_{j=0}^{N-1} B_i^j & e_i = 0 \\ \left(\sum\limits_{j=0}^{N-1} C_i^j\right)' & e_i = 1 \end{cases}$$

holds with probability $1 - (\frac{1}{2})^N$, where e_0 is equal to $\sum\limits_{j=0}^{N-1} \Lambda_0^j$ as defined in **Case 1** of Lemma 2.

□

Lemma 4. *Suppose that* $\begin{cases} x \oplus \sigma^j = x + \Delta^j \\ x \oplus e\sigma^j = x + \Lambda^j \end{cases}$ *for* $j = 0, \ldots, N-1$, *where* $\sigma^j \in \mathbb{Z}_2^n$ *is unknown random number,* $\Delta^j, \Lambda^j \in \mathbb{Z}_2^n$ *are known values,* $e \in \mathbb{Z}_2^n$ *is a known random number, and* $x \in \mathbb{Z}_2^n$ *is an unknown value. Each bit* x_i *of* $x \in \mathbb{Z}_2^n$ *for* $i = 0, \ldots, n-2$ *can be determined with an average probability* P_i, *where*

$$P_i = \begin{cases} \frac{1}{4}\left(1 - (\frac{1}{2})^{E_{i+1}}\right)\left(1 - (\frac{1}{4})^{E_i}\right) + \frac{1}{2}\left(1 - (\frac{1}{2})^{E_i}\right) & i \in \{1, \ldots, n-3\} \\ \frac{1}{2}\left(1 - (\frac{1}{2})^{E_i}\right) & i = n-2 \end{cases}$$

and $E_i = \begin{cases} N & i=0 \\ \frac{N}{2^i}+\frac{N}{4} & i \in \{1,\ldots,n-1\} \end{cases}$.

Proof. For $j \in \mathbb{Z}_N$ and $i \in \mathbb{Z}_n$, let $\sigma_i^j = \mathcal{F}\left(\sigma_{i-1}^j, x_{i-1}\right)$, then $\mathcal{F}\left(\sigma_{i-1}^j, x_{i-1}\right)$ is defined as

$$\mathcal{F}\left(\sigma_{i-1}^j, x_{i-1}\right) = \begin{cases} \Delta_i^j \oplus x_{i-1} & \sigma_{i-1}^j = 1 \\ A_{i-1}^j & \sigma_{i-1}^j = 0 \\ B_{i-1}^j & e_i = 1, e_{i-1} = 0 \\ \alpha_i^j \oplus \sigma_{i-1}^j{'}\alpha_{i-1}^j & e_i = 0, e_{i-1} = 1 \end{cases}, \tag{9}$$

which is derived by the equations (3) and (5)-(7) in Lemma 2 (having the similar conditions), where $\sigma_0^j = \Delta_0^j$ (see Lemma 1), $A_i^j = \Delta_{i+1}^j \oplus \Delta_i^j$, $B_i^j = \Lambda_{i+1}^j \oplus \Lambda_i^j$ and $\alpha_i^j = \Delta_i^j \oplus \Lambda_i^j$.

Let E_i be the expected value of the number of known σ_i^j for all $j \in \mathbb{Z}_N$. Given the randomness of e and σ^j, E_i satisfies the following equation (10), where $\frac{E_{i-1}}{4}$, $\frac{N}{4}$, and $\frac{E_{i-1}}{4}$ correspond respectively to the expected values under the following cases:
1) $e_{i-1} \oplus e_i = 0$ and $\sigma_{i-1}^j = 0$, with probability $\frac{1}{4}$;
2) $\{e_i, e_{i-1}\} = \{1, 0\}$, with probability $\frac{1}{4}$;
3) $\{e_i, e_{i-1}\} = \{0, 1\}$, with probability $\frac{1}{4}$.

$$E_i = \begin{cases} N & i=0 \\ \frac{E_{i-1}}{4}+\frac{N}{4}+\frac{E_{i-1}}{4} = \frac{E_{i-1}}{2}+\frac{N}{4} & i \in \{1,\ldots,n-1\} \end{cases} \tag{10}$$

Furthermore, the equations (ref. equations (3) and (4)) in Lemma 2 can be used to derive the following equation

$$\begin{aligned} &\sigma_i^j\left(e_{i+1} \oplus e_i\right) x_i = \sigma_i^j\left(\Delta_i^j e_{i+1} \oplus \Lambda_i^j e_i\right) \oplus A_i^j e_{i+1} \oplus B_i^j \\ &\left(e_{i+1} \oplus e_i\right) \sigma_{i+1}^j = B_i^j \oplus \left(A_i^j \oplus \sigma_i^j \alpha_i^j\right) e_i \end{aligned}, \tag{11}$$

From equation (3), the value of x_i can be determined provided that $\sigma_i^j = 1$ and σ_{i+1}^j is known for a specific j. The following cases must be considered in order to determine x_i.

1. **Case 1:** $\{e_{i+1}, e_i\} = \{1, 0\}$
   (a) When $\{e_{i+1}, e_i\} = \{1, 0\}$, we have $\sigma_i^j\left(x_i \oplus \Delta_i^j\right) = A_i^j \oplus B_i^j$ and $\sigma_{i+1}^j = B_i^j$ by equation (11).
   For any $j \in \mathbb{Z}_N$, if σ_i^j is known and $\sigma_i^j = 1$ (obtained from $\mathcal{F}\left(\sigma_{i-1}^j, x_{i-1}\right)$), then $x_i = \Delta_{i+1}^j \oplus B_i^j$; Otherwise, if there exist $A_i^j \oplus B_i^j = 1$, then $x_i = \Delta_i^j{'}$.
   (b) Let $P(\text{recovering } x_i | \{e_{i+1}, e_i\} = \{1, 0\})$ (P_{10} for short) be defined as the average probability of recovering x_i when $\{e_{i+1}, e_i\} = \{1, 0\}$. It is known that $\sigma_{i+1}^j = B_i^j$ for any $j \in \mathbb{Z}_N$. Consequently, P_{10} is primarily influenced by the condition that σ_i^j is not only known, but also equal to 1. From equation (10), $P_{10} = 1 - (\frac{1}{2})^{E_i}$.
2. **Case 2:** $\{e_{i+1}, e_i\} = \{0, 1\}$
   (a) When $\{e_{i+1}, e_i\} = \{0, 1\}$, we have $\sigma_i^j\left(x_i \oplus \Lambda_i^j\right) = B_i^j$ and $\sigma_{i+1}^j = \alpha_{i+1}^j \oplus \sigma_i^j{'}\alpha_i^j$.
   For any $j \in \mathbb{Z}_N$, if $\sigma_i^j = 1$ (obtained from $\mathcal{F}(\sigma_{i-1}^j, x_{i-1})$), then $x_i = \Lambda_{i+1}^j$, Otherwise, if there exists $B_i^j = 1$, then $x_i = \Lambda_i^j{'}$.

(b) Similarly, the average probability of recovering x_i when $\{e_{i+1}, e_i\} = \{0, 1\}$, denoted by $P(\text{recovering } x_i | \{e_{i+1}, e_i\} = \{0, 1\})$ (P_{01} for short), depends on whether σ_i^j is known and equal to 1, i.e., $P_{01} = 1 - (\frac{1}{2})^{E_i}$.

3. **Case 3:** $\{e_{i+1}, e_i\} = \{1, 1\}$. There are also two sub-phases:

   (a) If $\{e_{i+2}, e_{i+1}, e_i\} = \{1, 1, 1\}$, substituting $\{e_{i+2}, e_{i+1}, e_i\} = \{1, 1, 1\}$ into equation (11), we have $\sigma_i^j{'}\alpha_i^j = \alpha_{i+1}^j$ and $\sigma_{i+1}^j{'}\alpha_{i+1}^j = \alpha_{i+2}^j$. If $\sigma_i^j = 1$, then $\alpha_{i+1}^j = \alpha_{i+2}^j = 0$. Consequently, σ_{i+1}^j and x_i cannot be determined when $\sigma_i^j = 1$.

   (b) If $\{e_{i+2}, e_{i+1}, e_i\} = \{0, 1, 1\}$, then $\sigma_{i+1}^j \left(x_{i+1} \oplus \Lambda_{i+1}^j\right) = B_{i+1}^j$.

   For any $j \in \mathbb{Z}_N$ and $i > 0$, if x_{i+1} has been known, $\sigma_i^j = 1$ and $\left(x_{i+1} \oplus \Lambda_{i+1}^j\right) = 1$, then $\sigma_{i+1}^j = B_{i+1}^j$ and $x_i = \Delta_{i+1}^j \oplus B_{i+1}^j$ (by equation (3)); Otherwise, if there exist $\sigma_i^j = 1$ and $B_{i+1}^j = 1$, then $\sigma_{i+1}^j = 1$, $x_{i+1} = \Lambda_{i+1}^j$ and $x_i = \Delta_{i+1}^j{'}$.

   (c) The key conditions for the recovery of x_i are as follows: 1) x_{i+1} is known, with a probability of $1 - (\frac{1}{2})^{E_{i+1}}$ (demonstrated in Case 2); 2) There exists an index j, such that σ_i^j is known to equal 1 and $\left(x_{i+1} \oplus \Lambda_{i+1}^j\right) = 1$, with a probability of $1 - \left(\frac{1}{4}\right)^{E_i}$.Hence,

$$P(\text{recovering } x_i | \{e_{i+2}, e_{i+1}, e_i\} = \{0, 1, 1\}) = \left(1 - \left(\frac{1}{2}\right)^{E_{i+1}}\right)\left(1 - \left(\frac{1}{4}\right)^{E_i}\right).$$

4. **Case 4 :** $\{e_{i+1}, e_i\} = \{0, 0\}$**.** There are two sub-phases:

   (a) If $\{e_{i+2}, e_{i+1}, e_i\} = \{0, 0, 0\}$, then $B_i^j = B_{i+1}^j = 0$, and thus no information is available to determine x_i.

   (b) If $\{e_{i+2}, e_{i+1}, e_i\} = \{1, 0, 0\}$, then $\sigma_{i+1}^j \left(x_{i+1} \oplus \Delta_{i+1}^j\right) = A_{i+1}^j \oplus B_{i+1}^j$ and $\sigma_{i+2}^j = B_{i+1}^j$.
   If x_{i+1} is known, $\sigma_i^j = 1$ and $x_{i+1} \oplus \Delta_{i+1}^j = 1$, then $\sigma_{i+1}^j = A_{i+1}^j \oplus B_{i+1}^j$ and $x_i = \Delta_{i+2}^j \oplus B_{i+1}^j$ (by equation (3)); Otherwise, for any $j \in \mathbb{Z}_N$ and $i > 0$, if there exist $\sigma_i^j = 1$ and $A_{i+1}^j \oplus B_{i+1}^j = 1$, then $\sigma_{i+1}^j = 1$ and $x_i = \Delta_{i+1}^j{'}$.

   (c) Similar to Case 3,

$$P(\text{recovering } x_i | \{e_{i+2}, e_{i+1}, e_i\} = \{1, 0, 0\}) = \left(1 - \left(\frac{1}{2}\right)^{E_{i+1}}\right)\left(1 - \left(\frac{1}{4}\right)^{E_i}\right).$$

In summary, the average probability of successfully recovering x_i for $j \in \{0, ..., n-2\}$, which may be denoted henceforth as $P(\text{recovering } x_i)$ (P_i for short), is subject to the condition that

$$P_i = \begin{cases} \frac{\left(1-\left(\frac{1}{2}\right)^{E_{i+1}}\right)\left(1-\left(\frac{1}{4}\right)^{E_i}\right)}{4} + \frac{1-\left(\frac{1}{2}\right)^{E_i}}{2} & i \in \{0, \ldots, n-3\} \\ \frac{1-\left(\frac{1}{2}\right)^{E_i}}{2} & i = n-2 \end{cases}. \tag{12}$$

□

If E_i (at least greater than $N/4$) is sufficiently large, P_i approximates to $\frac{3}{4}$ for $i \in \{0, \ldots, n-3\}$ and $\frac{1}{2}$ for $i = n-2$, respectively. In particular, when $\{e_{i+2}, e_{i+1}, e_i\} = \{0, 0, 0\}$ and $\{e_{i+2}, e_{i+1}, e_i\} = \{1, 1, 1\}$ with a probability of $\frac{1}{4}$, the value of x_i remains undeterminable. As an alternative approach, in Subsection 3.2, eight distinct instances of e (as in Algorithm 2) are employed to ascertain the remaining bits of x. Additionally, the most significant bit, x_{n-1}, remains inherently unrecoverable.

It should be noted that the average probability is defined in this context, rather than an exact probability. This is due to the fact that the quantity of known σ_i^j is represented by the expected value E_i, which is contingent upon the random distribution of e_i. Consequently, there may be a slight discrepancy between the average probability and the exact probability when the value of e assumes specific real values. The experiments in Section 5 also illustrate this case. Conversely, it serves as a theoretical approximation intended to provide an estimate of the exact probability.

Lemma 5. *Given the function $Maj\left(a,b,c\right) = ab \oplus bc \oplus ac$, the following equations hold:*

$$Maj\left(a, b \oplus \sigma, c\right) - Maj\left(a,b,c\right) = \left(b \oplus \left(a \oplus c\right)\sigma\right) - b,$$

$$Maj\left(a, b, c \oplus \sigma\right) - Maj\left(a,b,c\right) = \left(c \oplus \left(a \oplus b\right)\sigma\right) - c,$$

where $a, b, c, \sigma \in \mathbb{Z}_2^n$.

Proof. From *Theorem 2*, we have

$$\begin{array}{l} Maj\left(a, b \oplus \sigma, c\right) = Maj\left(a,b,c\right) \oplus \left(a \oplus c\right)\sigma \\ = Maj\left(a,b,c\right) + \left(a \oplus c\right)\sigma - \left(Maj\left(a,b,c\right)\left(a \oplus c\right)\sigma\right) \ll 1 \\ = Maj\left(a,b,c\right) + \left(a \oplus c\right)\sigma - \left(b\left(a \oplus c\right)\sigma\right) \ll 1 \\ = Maj\left(a,b,c\right) + \left(b \oplus \left(a \oplus c\right)\sigma\right) - b \end{array}.$$

Similarly, the equation $Maj\left(a,b,c \oplus \sigma\right) - Maj\left(a,b,c\right) = \left(c \oplus \left(a \oplus b\right)\sigma\right) - c$ can also be demonstrated. □

Lemma 6. *Given the function $Ch\left(e,f,g\right) = ef \oplus e\prime g$, the following equations hold:*

$$Ch\left(e, f \oplus \sigma, g\right) - Ch\left(e,f,g\right) = \left(f \oplus e\sigma\right) - f,$$

$$Ch\left(e, f, g \oplus \sigma\right) - Ch\left(e,f,g\right) = \left(f \oplus e\prime\sigma\right) - f,$$

where $e, f, g, \sigma \in \mathbb{Z}_2^n$.

Proof. From *Theorem 2*, we have

$$\begin{array}{l} ch\left(e, f \oplus \sigma, g\right) = ch\left(e,f,g\right) \oplus e\sigma \\ = ch\left(e,f,g\right) + e\sigma - \left(ch\left(e,f,g\right)e\sigma\right) \ll 1 \\ = ch\left(e,f,g\right) + e\sigma - \left(ef\sigma \ll 1\right) \\ = ch\left(e,f,g\right) + \left(f \oplus e\sigma\right) - f \end{array}$$

Similarly, the equation $Ch\left(e,f,g \oplus \sigma\right) - Ch\left(e,f,g\right) = \left(f \oplus e\prime\sigma\right) - f$ can also be demonstrated. □

3.2 Recovery of Initial Input and Message Block in Final Compression Function

By employing the lemmas above and the random word fault model, it is feasible to retrieve the initial vector and the message in SHA2's final compression function (see Figure 1). The model assumes that a single state register, denoted as x, is subject to a random fault, resulting in its transition into the j-th faulty state register, x^j, which is equal to $x \oplus \sigma^j$ for each $j \in \mathbb{Z}_N$. Each σ^j is a unique random number. After injecting faults into various registers, the following propositions can be drawn.

Proposition 1. *By introducing N random faults for each of the 14 special state registers in SHA2, knowing the correct hash value $\{A, ..., H\}$ and all the faulty hash values $\{A^j, ..., H^j\}$ for any $j \in \mathbb{Z}_N$, it is possible to recover the $(R-1)$-th round's input registers $\{a_{R-1}, ..., h_{R-1}\}$ and their faulty forms $\{a^j_{R-1}, ..., h^j_{R-1}\}$ with probability $P_r \cdot P_l$, where $P_r = \left(1 - (\frac{1}{2})^N\right)^{4n}$ and $P_l = \left(1 - (\frac{1}{2})^N\right)^{8n} \prod_{i=0}^{n-2} \left(1 - (1 - P_i)^8\right)$.*

Proof. **Step.1 Recover e_{R-1}, f_R, f_0 by introducing N faults into f_{R-1}.**

As shown in Figure 4, if N random faults are introduced into f_{R-1}, then $f_{R-1} \to f^j_{R-1}$, where $f^j_{R-1} = f_{R-1} \oplus \sigma^j$ for each $j \in \mathbb{Z}_N$. Consequently, the final outputs, namely A, E and G, are transformed into A^j, E^j and G^j, respectively.

From Lemma 6, we have

$$\begin{aligned} G^j - G &= \left(f_{R-1} \oplus \sigma^j\right) - f_{R-1} \\ A^j - A &= \left(f_{R-1} \oplus e_{R-1}\sigma^j\right) - f_{R-1} \end{aligned} \tag{13}$$

Let $G^j - G = \Delta^j$ and $A^j - A = \Lambda^j$. Then, e_{R-1} can be determined from Lemma 2 with probability $\left(1 - (\frac{1}{2})^N\right)^n$. Consequently, the values of f_R and f_0 can also be determined.

Step.2 Recover f_{R-1}, g_R, g_0 by introducing N faults into f_{R-2}.

Similarly, if N random faults are induced into f_{R-2}, then f_{R-2}, H, and B are transformed into f^j_{R-2} ($=f_{R-2} \oplus \sigma^j$), H^j, and B^j for any $j \in \mathbb{Z}_N$, respectively.

$$\begin{aligned} H^j - H &= \left(f_{R-2} \oplus \sigma^j\right) - f_{R-2} \\ B^j - B &= \left(f_{R-2} \oplus e_{R-2}\sigma^j\right) - f_{R-2} \end{aligned} \tag{14}$$

From Lemma 2, e_{R-2} (i.e.,f_{R-1} and g_R) can be recovered with probability $\left(1 - (\frac{1}{2})^N\right)^n$.

Step.3 Recover g_{R-1}, h_R, h_0 by introducing N faults into f_{R-3} and g_{R-3} respectively.

Inducing N faults into f_{R-3} results in $f^j_{R-3} = f_{R-3} \oplus \sigma^j$, altering output C to C^j for all $j \in \mathbb{Z}_N$. Unlike the scenario above, the hash output does not reveal the value of $f^j_{R-3} - f_{R-3}$. Consequently, a further N faults in g_{R-3} are required, resulting in the transformation of $g_{R-3} \to g^j_{R-3}(= g_{R-3} \oplus \varepsilon^j)$ and $C \to C^{N+j}$, respectively. The random number ε^j is analogous to σ^j for all $j \in \mathbb{Z}_N$.

Hence, from Lemma 6, we have the following equations

$$\begin{aligned} C^j - C &= \left(f_{R-3} \oplus e_{R-3}\sigma^j\right) - f_{R-3} \\ C^{j+N} - C &= \left(g_{R-3} \oplus e_{R-3}\prime\varepsilon^j\right) - g_{R-3} \end{aligned} . \tag{15}$$

From Lemma 3, let $C^j - C = \Lambda^j, C^{j+N} - C = \Psi^j$, then e_{R-3}, h_R and h_0 can be recovered with probability $\left(1 - (\frac{1}{2})^N\right)^n$.

Step.4 Recover h_{R-1} by introducing N faults into f_{R-4} and g_{R-4} respectively.

Similarly, N faults are introduced into f_{R-4} and g_{R-4}, respectively, resulting in $f^j_{R-4}(= f_{R-4} \oplus \sigma^j)$, faulty output D^j, $g^j_{R-4}(= g_{R-4} \oplus \varepsilon^j)$ and faulty output D^{j+N} for all $j \in \mathbb{Z}_N$. we have

$$\begin{aligned} D^j - D &= \left(f_{R-4} \oplus e_{R-4}\sigma^j\right) - f_{R-4} \\ D^{j+N} - D &= \left(g_{R-4} \oplus e_{R-4}\prime\varepsilon^j\right) - g_{R-4} \end{aligned} . \tag{16}$$

From Lemma 3, let $D^j - D = \Lambda^j$ and $D^{j+N} - D = \Psi^j$, then e_{R-4}, i.e., h_{R-1} can be recovered with probability $\left(1 - (\frac{1}{2})^N\right)^n$.

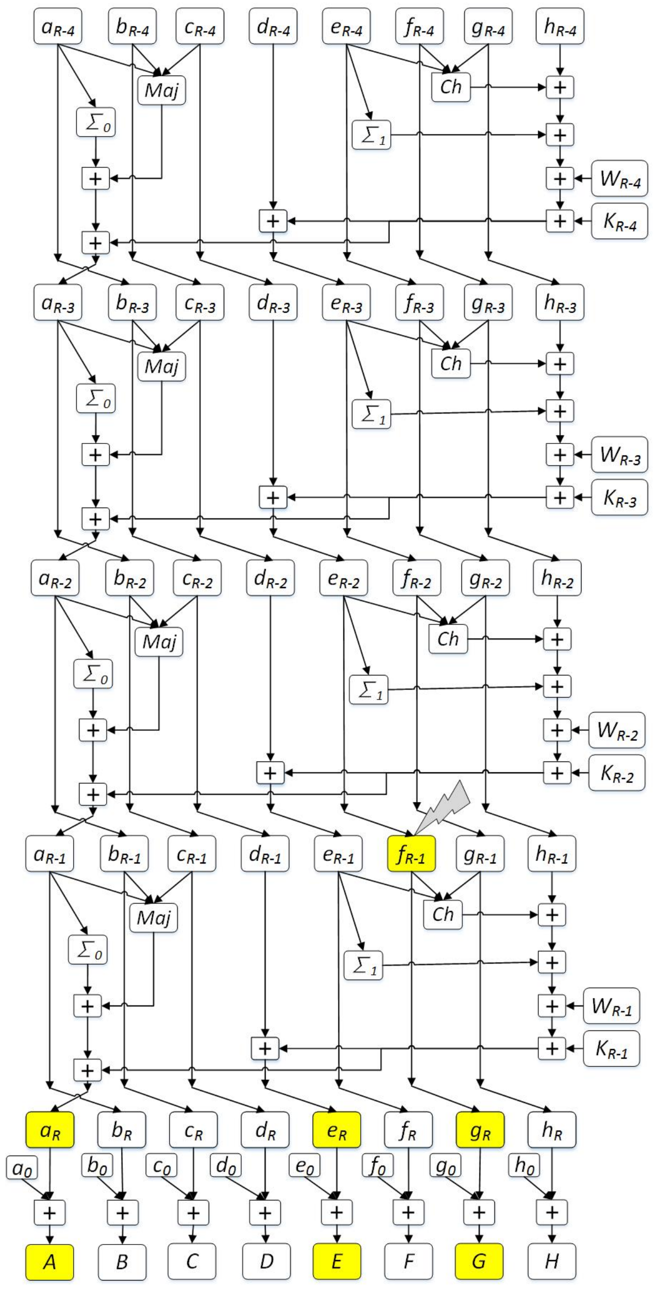

Figure 4: Fault injection on f_{R-1}

In summary, the four-step process above recovers the registers $\{e_{R-1}, f_{R-1}, g_{R-1}, h_{R-1}\}$, $\{f_R, g_R, h_R\}$, and $\{f_0, g_0, h_0\}$ with a probability P_r that satisfies the following equation:

$$P_r = \left(1 - \left(\frac{1}{2}\right)^N\right)^{4n}. \tag{17}$$

Meanwhile, the recovery process requires a total of $6N$ random faults.

It should be noted that, due to the lack of knowledge about G^j, G, H^j, and H, steps 1-2 for SHA224 and SHA384 will adopt the same strategy as steps 3-4. N random faults are introduced separately into f_{R-1}, g_{R-1} in step 1 (f_{R-2}, g_{R-2} in step 2). The faulty values A^j, A^{N+j} (B^j, B^{N+j} for step 2) are obtained. From Lemma 3, e_{R-1} (f_{R-1}) can also be recovered.

Step.5 Recover $a_{R-1} \oplus c_{R-1}$ and partial bits of b_{R-1}(denoted as $part(b_{R-1})$) by introducing N faults into b_{R-1}.

As shown in Figure 4, if N random faults are induced into b_{R-1}, then $b_{R-1} \to b_{R-1} \oplus \sigma^j$ (denoted by b^j_{R-1}), $A \to A^j$ and $C \to C^j$ for all $j \in \mathbb{Z}_N$.

From Lemma 5, we have

$$\begin{array}{l} C^j - C = \left(b_{R-1} \oplus \sigma^j\right) - b_{R-1} \\ A^j - A = \left(b_{R-1} \oplus (a_{R-1} \oplus c_{R-1})\sigma^j\right) - b_{R-1} \end{array}. \tag{18}$$

From Lemma 2, let $C^j - C = \Delta^j$ and $A^j - A = \Lambda^j$, then $a_{R-1} \oplus c_{R-1}$ can be recovered with probability $\left(1 - (\frac{1}{2})^N\right)^n$.

Furthermore, the recovered $a_{R-1} \oplus c_{R-1}$ from equation (18) enables the recovery of $n-1$ bits of b_{R-1} (denoted by $part(b_{R-1})$) with a probability given by $\prod_{i=0}^{n-2} P_i$, where P_i is the i-th bit recovery probability as defined in equation (12) and Table 3.

Step.6 Recover $a_{R-1} \oplus b_{R-1}$ and partial bits of c_{R-1}(denoted as $part(c_{R-1})$) by introducing N faults into c_{R-1}.

Similarly, based on Lemma 5 and Lemma 2, the value of $a_{R-1} \oplus c_{R-1}$ can be recovered with probability $\left(1 - (\frac{1}{2})^N\right)^n$. Based on Lemma 4, $n-1$ bits of $part(c_{R-1})$ is recovered with a probability $\prod_{i=0}^{n-2} P_i$.

It should be noted that due to the single random value of $a_{R-1} \oplus c_{R-1}$ (and $a_{R-1} \oplus b_{R-1}$), not all $n-1$ bits of $part(b_{R-1})$ (and $part(c_{R-1})$) are recovered.

Step.7 Recover more bits of a_{R-1}, b_{R-1} and c_{R-1}.

Given the known values of $a_{R-1} \oplus b_{R-1}$ and $a_{R-1} \oplus c_{R-1}$, the value of $b_{R-1} \oplus c_{R-1}$ can be determined. From Algorithm 1, the values of $part(b_{R-1}), part(c_{R-1})$ and subsequently $part(a_{R-1})$ are updated.

In view of the random nature of $a_{R-1} \oplus b_{R-1}$ and $a_{R-1} \oplus c_{R-1}$, the probability of recovery for each bit (except the MSB) of b_{R-1}, c_{R-1}, and a_{R-1} is given by $1 - (1 - P_i)^2$.

Step.8 Repeat steps 5 to 7 in order to update more bits of $part(c_{R-1})$ (and $part(a_{R-1})$, $part(b_{R-1})$, $part(d_{R-1})$) by introducing N faults into b_{R-2}, c_{R-2}, b_{R-3}, c_{R-3}, b_{R-4} and c_{R-4} respectively.

As outlined in Algorithm 2, by iterating steps 5 to 7, it is possible to recover $n - T$ bits of the set $\{a_{R-1}, b_{R-1}, c_{R-1}, d_{R-1}\}$, where T is the number of undetermined bits and is typically equal to 1.

From Algorithm 2, the probability of undetermined bits in $part(a_{R-1})$, $part(b_{R-1})$, $part(c_{R-1})$ and $part(d_{R-1})$ depends on the randomness of eight values: $\{a_{R-1} \oplus b_{R-1}, a_{R-1} \oplus c_{R-1}, ..., a_{R-4} \oplus c_{R-4}\}$ and the faulty value σ_j for each $j \in \mathbb{Z}_N$. Excluding the most significant bit, the probability of determining any bit in b_{R-1} is about $1 - (1 - P_i)^8$, when $b_{R-1} \oplus c_{R-1}$, $a_{R-1} \oplus b_{R-1}$, and $b_{R-1} \oplus d_{R-1}$ are known (by Lemma 2). Consequently, the

Algorithm 1 Algorithm for Updating target registers

```
Require: $u(= x \oplus y)$, $part(x)$ and $part(y)$, where $u, x, y \in \mathbb{Z}^n$
Ensure: Updated $part(x)$ and $part(y)$
  For $i = 0$ to $n - 2$
      If $x_i \in part(x)$ and $y_i \notin part(y)$
          $y_i = u_i \oplus x_i$ and incorporate $y_i$ into $part(y)$
      Else If $y_i \in part(y)$ and $x_i \notin part(x)$
          $x_i = u_i \oplus y_i$ and incorporate $x_i$ into $part(x)$
  return $part(x)$ and $part(y)$
```

Algorithm 2 Recovery of $n - T$ bits of a_{R-1}, b_{R-1}, c_{R-1} and d_{R-1}

```
Require: All faulty results denoted as $\{S_0^j, S_1^j, S_2^j, S_3^j, S_4^j, S_5^j, S_6^j, S_7^j\}$ (equivalent to
  $\{A^j, B^j, C^j, D^j, E^j, F^j, G^j, H^j\}$) for any $j \in \mathbb{Z}_N$, and all correct results denoted as
  $\{S_0, S_1, S_2, S_3, S_4, S_5, S_6, S_7\}$ (equivalent to$\{A, B, C, D, E, F, G, H\}$), obtained by intro-
  ducing $N$ faults into $b_i$ and $c_i$ (for each $i \in \{R-1, ..., R-4\}$) respectively.
Ensure: The least significant $n - T$ bits of $a_{R-1}$, $b_{R-1}$, $c_{R-1}$ and $d_{R-1}$, denoted as
  $part(a_{R-1})$, $part(b_{R-1})$, $part(c_{R-1})$ and $part(d_{R-1})$, respectively.
  For $i = R - 1$ to $R - 4$
      1. Introduce $N$ random faults into $b_i$ so that $b_i \to b_i^j$ $(= b_i \oplus \sigma^j)$, and thereby
  yielding $N$ sets of faulty hash outputs $\{S_0^j, S_1^j, S_2^j, S_3^j, S_4^j, S_5^j, S_6^j, S_7^j\}$ for each $j \in \mathbb{Z}_N$.
      2. Let $\Delta^j = S_{R-i+1}^j - S_{R-i+1} = (b_i \oplus \sigma^j) - b_i$, $\Lambda^j = S_{R-i-1}^j - S_{R-i-1} = (b_i \oplus$
  $(a_i \oplus c_i)\sigma^j) - b_i$.
      3. Recover $a_i \oplus c_i$ by Lemma 2 and $part(b_i)$ by Lemma 4.
      4. Repeat steps 1-3 for $c_i$ (where $\Delta^j = S_{R-i+2}^j - S_{R-i+2}$ and $\Lambda^j = S_{R-i-1}^j -$
  $S_{R-i-1}$), using lemmas 2 and 4 to recover $a_i \oplus b_i$ and $part(c_i)$, respectively.
      5. Update $part(a_{R-1})$, $part(b_{R-1})$, $part(c_{R-1})$, $part(d_{R-1})$ by Algorithm 1.
  return $part(a_{R-1})$, $part(b_{R-1})$, $part(c_{R-1})$ and $part(d_{R-1})$
```

probability of successfully recovering all $n - T$ bits of $\{a_{R-1}, b_{R-1}, c_{R-1}, d_{R-1}\}$ is given by the following equation:

$$P_l = \left(1 - \left(\frac{1}{2}\right)^N\right)^{8n} \prod_{i=0}^{n-2} \left(1 - (1 - P_i)^8\right). \tag{19}$$

It can be demonstrated that the value of P_l approaches 99% when $n = 32$ or 64 and N is sufficiently large, for example, $N = 15$. Consequently, T is typically equal to 1.

In summary, Algorithm 2 allows for the recovery of $\{a_{R-1}, b_{R-1}, c_{R-1}, d_{R-1}\}$ (with only T guessed bits). In addition, the recovery process requires $8N$ random faults.

Step.9 Recover the faulty registers $\{a_{R-1}^j, ..., h_{R-1}^j\}$ for each $j \in \mathbb{Z}_N$.

As shown in Figure 4, state registers $\{a_{R-1}, \ldots, h_{R-1}\}$ and $\{b_0, c_0, d_0, f_0, g_0, h_0\}$, along with correct and faulty hash values, $\{A, \ldots, H\}$ and $\{A^j, \ldots, H^j\}$ for each $j \in \mathbb{Z}_N$, are all known. It is evident that

$$a_{R-1}^j = B^j - b_0, b_{R-1}^j = C^j - c_0, c_{R-1}^j = D^j - d_0, and$$

$$e_{R-1}^j = F^j - f_0, f_{R-1}^j = G^j - g_0, g_{R-1}^j = H^j - h_0.$$

Let $T_i = Ch(e_i, f_i, g_i) + \sum_1 (e_i)$, $U_i = Maj(a_i, b_i, c_i) + \sum_0 (a_i)$, $T_i^j = Ch(e_i^j, f_i^j, g_i^j) + \sum_1 \left(e_i^j\right)$ and $U_i^j = Maj(a_i^j, b_i^j, c_i^j) + \sum_0 \left(a_i^j\right)$ (for $i \in \mathbb{Z}_R$ and $j \in \mathbb{Z}_N$), then

$$h_{R-1}^j = A^j - A - (T_{R-1}^j + U_{R-1}^j - T_{R-1} - U_{R-1}) + h_{R-1},$$

where A^j, A, T^j_{R-1}, U^j_{R-1}, U_{R-1} and T_{R-1} are known quantities.

Consequently, $d^j_{R-1} = E^j - E - (T^j_{R-1} + h^j_{R-1} - T_{R-1} - h_{R-1}) + d_{R-1}$.

□

Proposition 2. *By introducing N random faults into each of the 48 special target registers (before $R-1$-th round) in SHA2, and knowing the $(R-1)$-th round's input registers $\{a_{R-1}, ..., h_{R-1}\}$ and all their faulty forms $\{a^j_{R-1}, ..., h^j_{R-1}\}$ for any $j \in \mathbb{Z}_N$, it is possible to recover the message block M^{r-1} and initial vector V_{r-1} of the final compression function $F(V_{r-1}, M^{r-1})$ with a probability $\left(1 - \frac{1}{2^N}\right)^{32n}$.*

Proof. **Step 1. Recover the correct registers $\{a_{R-2}, b_{R-2}, ..., h_{R-2}\}$.**

Obviously, $a_{R-2} = b_{R-1}$, $b_{R-2} = c_{R-1}$, $c_{R-2} = d_{R-1}$, $e_{R-2} = f_{R-1}$, $f_{R-2} = g_{R-1}$, $g_{R-2} = h_{R-1}$.

As demonstrated in the Step 4 of Proposition 1, if N random faults are introduced into f_{R-5} and g_{R-5} respectively, then $f_{R-5} \to f^j_{R-5}$ (where $f^j_{R-5} = f_{R-5} \oplus \sigma^j$) and $g_{R-5} \to g^j_{R-5}$ (where $g^j_{R-5} = g_{R-5} \oplus \varepsilon^j$) for any $j \in \mathbb{Z}_N$. Subsequently, by Lemma 6, we have

$$\begin{array}{l} d^j_{R-1} - d_{R-1} = \left(f_{R-5} \oplus e_{R-5}\sigma^j\right) - f_{R-5} \\ d^{N+j}_{R-1} - d_{R-1} = \left(g_{R-5} \oplus e_{R-5}\prime\varepsilon^j\right) - g_{R-5} \end{array},$$

where d^j_{R-1} and d^{N+j}_{R-1} represents the j-th faulty value of d_{R-1} when a random fault is introduced into f_{R-5} and g_{R-5}, respectively.

Let $d^j_{R-1} - d_{R-1} = \Lambda^j, d^{N+j}_{R-1} - d_{R-1} = \Psi^j$, then h_{R-2} (i.e., e_{R-5}) can be recovered with probability $(1 - \frac{1}{2^N})^n$ from Lemma 3. A total of $2N$ faults are required.

As demonstrated in the Step 5 of proposition 1, if N faults are introduced into b_{R-3}, then $b_{R-3} \to b^j_{R-3}$ (where $b^j_{R-3} = b_{R-3} \oplus \sigma^j$). Consequently, from Lemma 5, we have

$$\begin{array}{l} d^j_{R-1} - d_{R-1} = \left(b_{R-3} \oplus \sigma^j\right) - b_{R-3} \\ b^j_{R-1} - b_{R-1} = \left(b_{R-3} \oplus (a_{R-3} \oplus c_{R-3})\sigma^j\right) - b_{R-3} \end{array}.$$

From Lemma 2, $a_{R-3} \oplus c_{R-3}$ can be recovered with probability $(1 - \frac{1}{2^N})^n$, where a_{R-3} is known to be equal to c_{R-1}. Consequently, d_{R-2} (i.e., c_{R-3}) can be recovered with N random faults.

Step 2. Recover message W_{R-2}.

Given the knowledge of T_{R-2} (i.e., $Ch(e_{R-2}, f_{R-2}, g_{R-2}) + \sum_1 (e_{R-2})$), d_{R-2}, K_{R-2} and e_{R-1}, the following equation is derived:

$$W_{R-2} = e_{R-1} - T_{R-2} - K_{R-2} - d_{R-2}.$$

Step 3. Recover the faulty registers $\{a^j_{R-2}, ..., h^j_{R-2}\}$ for any $j \in \mathbb{Z}_N$

In the set of values $\{a^j_{R-2}, ..., h^j_{R-2}\}$, only the values d^j_{R-2} and h^j_{R-2} remain unknown. Furthermore, the values of T^j_{R-2} (i.e., $Ch(e^j_{R-2}, f^j_{R-2}, g^j_{R-2}) + \sum_1 \left(e^j_{R-2}\right)$) and U^j_{R-2} (i.e., $Maj(a^j_{R-2}, b^j_{R-2}, c^j_{R-2}) + \sum_0 \left(a^j_{R-2}\right)$) for any $j \in \mathbb{Z}_N$ are known. Hence, the values of h^j_{R-2} and h^j_{R-2} can be obtained as follows:

$$h^j_{R-2} = a^j_{R-1} - T^j_{R-2} - W_{R-2} - K_{R-2} - U^j_{R-2},$$

and

$$d^j_{R-2} = e^j_{R-1} - T^j_{R-2} - W_{R-2} - K_{R-2} - h^j_{R-2}.$$

Following the completion of steps 1-3, the $R-2$-th correct and faulty round inputs and message W_{R-2} can be recovered with probability $\left(1 - \frac{1}{2^N}\right)^{2n}$ and $3N$ random faults.

Step 4. Recover the message block M^{r-1} and initial vector V_{r-1} in the compression function $F(V_{r-1}, M^{r-1})$.

Repeat steps 1-3 until all 16 intermediate round message sets, $\{W_{R-2}, ..., W_{R-17}\}$, have been recovered. Subsequently, the original message block, denoted by M^{r-1}, is inferred via equation (1). The initial vector V_{r-1} is then derived retrospectively using the recovered M^{r-1} and the known final stage inputs, namely $\{a_{R-1}, ..., h_{R-1}\}$.

In summary, the final message block M^{r-1} and its corresponding initial vector V_{r-1} can be recovered with probability $\left(1 - \frac{1}{2^N}\right)^{32n}$ and $48N$ random faults.

□

Finally, a practical differential fault attack can be achieved by applying Propositions 1 and 2. The specific steps for executing this attack are listed in Algorithm 3. The total number of random faults is $62N$. Note that the recovery of d_{R-2} and d_{R-3} as Step 1 of Proposition 2 may be omitted since the registers $\{a_{R-i}, \ldots, d_{R-i}\}$ for any $i \in \{2, 3\}$ have already been successfully recovered by Algorithm 2 (with T guessed bits) as prescribed in Proposition 1. This optimization reduces the total number of random faults to $62N - 2$.

Algorithm 3 Recovery of initial vector V_{r-1} and message block M^{r-1} in compression function $F(V_{r-1}, M^{r-1})$

```
Require: The correct and faulty hash values for each of the $62N - 2$ special targets,
  denoted as $\{A, ..., H\}$ and $\{A^j, ..., H^j\}$, respectively, where $j \in \mathbb{Z}_N$.
Ensure: $V_{r-1}$ (equivalent to the set of $\{a_0, ..., h_0\}$) and $M^{r-1}$
  1. Recover the last round of correct and faulty inputs $\{a_{R-1}, b_{R-1}, ..., h_{R-1}\}$ and
  $\{a^j_{R-1}, b^j_{R-1}, ..., h^j_{R-1}\}$ according to Proposition 1, where there are $T$ unknown bits
  in any one of $a_{R-1}, ..., b_{R-1}$.
  2. Guess the unknown $T$ bits of any one in the set $\{a_{R-1}, ..., d_{R-1}\}$, thereby yielding
  $2^T$ guessed values of the set $\{a_{R-1}, ..., d_{R-1}\}$.
  3. For each guessed value:
      3.1. Recover $V_{r-1}$ and $M^{r-1}$ according to Proposition 2.
      3.2. If $F(V_{r-1}, M^{r-1}) + V_{r-1} = \{A, ..., H\}$
              return $V_{r-1}$ and $M^{r-1}$.
  return Recovery failed.
```

4 Cases of Study

The theoretical foundations set out in the aforementioned propositions can now be applied to the analysis of SHA2, SHACAL-2, and HMAC-SHA2. Furthermore, the attack methodology is also applicable to other algorithms that use SHA2-like functions, such as SM3 and A5/1. The following sections will present the attacks against different modes and algorithms.

4.1 Attack on SHA2 and its application

As shown in Figure 1, the adversary is unable to obtain the faulty output (such as the faulty values of $V_1, ..., V_{r-1}$) of the compression function, except for the final hash output of $F(V_{r-1}, M^{r-1})$. In light of the above, the objective of our attack on SHA2 is the final compression function, $F(V_{r-1}, M^{r-1})$. In the context of the SHA2 family, which encompasses SHA224, SHA256, SHA384, and SHA512, it is only possible to recover the final message block, M^{r-1}, and the corresponding initial state, V_{r-1}, by Algorithm 3. Despite this limitation, several potential attack scenarios remain feasible, as detailed below.

- In dynamic token systems, the SHA2 algorithm is employed for the purpose of compressing the seed key and identifier, and subsequently outputting either the left or right half of the hash value. In this case, it is possible to apply Lemma 3 in place of Lemma 2, as with the attacks on SHA224 and SHA384. Consequently, our attack strategy allows for the recovery of the seed key when processing a single message block ($r = 1$).
- In identity authentication mechanisms, the user's password, ID, and salt are concatenated and hashed to generate a verification hash. Similarly, our attack method enables the recovery of the ID, salt, and hash within a single message block.
- SHA2 is critical for public key cryptography which use the private key and message as hash inputs, such as ECDSA and EdDSA. A compromise of SHA2 would also compromise the security of these signatures. It is also a core element in post-quantum systems, primarily used for functions such as random number generation and key compression, and its strength directly affects the reliability of post-quantum cryptography, particularly in hash-based schemes.

4.2 Forgery Attack on HMAC-SHA2

Based on our methodology of fault attacks, an almost universal forgery attack can be successfully performed as the same with in [HLMS14, NHGG18]. For ease of understanding, the entire attack process is still briefly described in Figure 5. The strategy involves inducing fault injections into the final compression function (highlighted in yellow) during the compression of the specific message block M^0, and utilizing Algorithm 3 to recover M^r and V_1. This capability enables the forging of HMACs for messages beginning with M^0 (as indicated by the right area enclosed by the dotted lines).

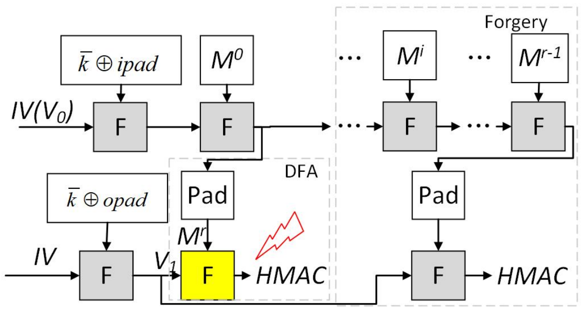

Figure 5: Almost universal forgery attack on HMAC

4.3 Differential fault attack on SHACAL-2

SHACAL-2 is a 256-bit block cipher supporting keys up to 512 bits, derived from the compression function of SHA256. Developed under the NESSIE project, it serves to secure electronic systems. As depicted in Figure 6, SHACAL-2 employs a modified F' function that mirrors SHA256's F function (illustrated in Figure 3), with the notable exception of omitting the "final addition" step. Here, the SHA256 message block M assumes the role of the encryption key, while V_0 takes on a position analogous to plaintext. Both are processed through F' to yield the ciphertext output.

Our approach enables an attack on SHACAL-2, exploiting the omission of the final addition step (Figure 3). This exposes the internal state registers $\{a_R, ..., h_R\}$ and their faulty values directly. Consequently, by applying Proposition 2, one can determine both

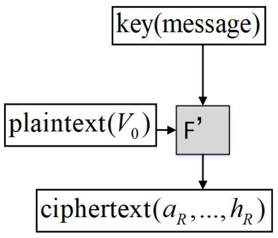

Figure 6: Structure of SHACAL-2

the plaintext (corresponding to V_0 in SHA2) and the key (corresponding to message block M in SHA2) using just $48N$ faults.

4.4 Extensions to Other Algorithms with Similar Boolean Functions

A number of algorithms, including SM3 [Sta16], A5/1, and A2U2 [DMRL11], employ similar Boolean functions, namely Maj and Ch. This analysis focuses on SM3's security against our proposed attack. For other stream ciphers, a comprehensive understanding of Boolean functions requires further research.

As depicted in Figure 7, SM3 mirrors the structural design of SHA256, incorporating boolean functions FF_j and GG_j that are equivalent to Maj and Ch, respectively, where

$$FF_j(X,Y,Z) = (X\&Y) \vee (X\&Z) \vee (Y\&Z) \quad 16 \leq j \leq 63$$

and

$$GG_j(X,Y,Z) = (X\&Y) \vee (X'\&Z) \quad 16 \leq j \leq 63.$$

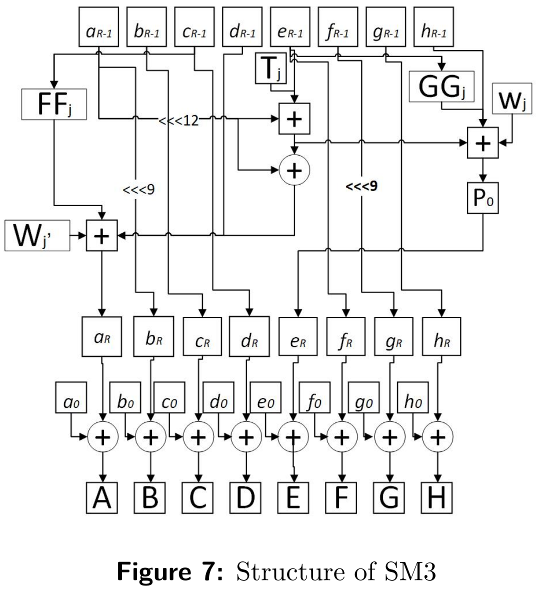

Figure 7: Structure of SM3

The principal difference between the functions FF_j and Maj (as well as GG_j and Ch) resides in their use of the bitwise OR operation ($\vee$) instead of the bitwise XOR operation ($\oplus$). Despite these operational differences, FF_j (GG_j) and Maj (Ch) are indeed functionally equivalent. The following section offers a detailed explanation of this equivalence.

$$\begin{aligned}
&(XY) \oplus (XZ) \oplus (YZ) \\
&= (XYZ' \vee XY'Z) \oplus (YZ) \\
&= (XYZ' \vee XY'Z)\,(YZ)' \vee (XYZ' \vee XY'Z)'\,(YZ) \\
&= (XYZ' \vee XY'Z)\,(Y' \vee Z') \vee (X' \vee (YZ' \vee Y'Z)')\,(YZ) \\
&= (XYZ' \vee XY'Z) \vee (X'YZ \vee YZ) \\
&= XYZ' \vee XY'Z \vee YZ \\
&= Y\,(XZ' \vee Z) \vee XY'Z \\
&= Y\,(Z \vee X) \vee XY'Z \\
&= XY \vee (Y \vee XY')\,Z \\
&= XY \vee XZ \vee YZ
\end{aligned} \tag{20}$$

$$\begin{aligned}
&(XY) \oplus (X'Z) \\
&= (XY)(X'Z)' \vee (XY)'(X'Z) \\
&= (XY \vee XYZ') \vee (X'Z \vee X'Y'Z) \\
&= XY \vee X'Z
\end{aligned} \tag{21}$$

The Equations (20) and (21) demonstrate that Maj is equivalent to FF_j, and Ch is equivalent to GG_j. Thus, as demonstrated in Sections 4.1 and 4.3, our fault attack is similarly effective against SM3 and its encryption variant. However, unlike SHA2 which incorporates an addition operation between initial inputs and final round outputs, SM3 employs XOR operations. This difference is of paramount importance for HMAC, especially considering the unknown the initial input to the compression function. As a result, the forgery attacks targeting SM3-HMAC become unfeasible under such conditions.

In conclusion, our proposed attack methodology reveals a multitude of potential attack instances, and thus poses a genuine threat to algorithms that incorporate the same Boolean functions, specifically Maj and Ch. It is necessary to re-evaluate the security of such cryptographic structures against fault attacks.

4.5 Discussion of Countermeasures

The countermeasure against our attacks comprises two primary strategies: verification and infection.

- **Verification.** The final four rounds must be recomputed, yielding two hash values that will be checked for equality. If they match, the hash value is returned; otherwise, an indication of a fault is returned.

- **Infection.** The $R-2$-th round of the compression function should be computed twice to obtain two sets of output values, which are denoted as $\{a_{R-1}, ..., h_{R-1}\}$ and $\{a^*_{R-1}, ..., h^*_{R-1}\}$, respectively. Subsequently, calculate

  $$(m_1 \& Maj(a_{R-1}, b_{R-1}, c_{R-1})) \vee Maj(a^*_{R-1}, b^*_{R-1}, c^*_{R-1})$$

  and

  $$(m_2 \& Ch(e_{R-1}, f_{R-1}, g_{R-1})) \vee Ch(e^*_{R-1}, f^*_{R-1}, g^*_{R-1}),$$

  and then substitute these results for $Maj(a_{R-1}, b_{R-1}, c_{R-1})$ and $Ch(e_{R-1}, f_{R-1}, g_{R-1})$, respectively. The variables m_1 and m_2 are represented by n-bit random numbers.

  Compared to verification, infection provides less exploitable information for our fault attacks. Moreover, repeated fault injection might potentially bypass the verification process, but such an approach is infeasible for infection.

5 Experiments

In this section, we simulated a random fault model and executed fault attacks to evaluate the validity and practicality of our method. For clarity and brevity, we focused on validating Lemma 2 and Algorithm 2, which are crucial for recovering the right and left halves of the final-round input registers for SHA256 and SHA512, respectively. It should be noted that similar attack strategies can be applied to other hash algorithms such as SHA224/384 and SM3, which share a similar operational structure. In addition, Table 2 provides the detailed results for recovering the final message block and initial vector in simulations. No further details will be given here.

First, we systematically simulated fault injection experiments N times, where N ranged from 1 to 30, specifically targeting register f_{R-1}, as shown in Figure 4. For each unique value of N (corresponding to the x-coordinate), we executed 1,000 attack instances based on Lemma 2 to attempt to recover e_{R-1}. We then computed and compared both the experimental and theoretical success rates (corresponding to the y-coordinate) of recovering e_{R-1}.

As shown in Figure 8, the red and blue lines represent the success rates of recovering e_{R-1} for SHA256 and SHA512, respectively. The dashed lines represent theoretical success rates, while the solid lines represent experimental success rates. The results show that the experimental success rate slightly exceeds its theoretical counterpart. In particular, the experimental success rate approaches an almost perfect 100% for both SHA256 and SHA512 when N reaches 15, which is achievable in real experiments.

Moreover, it is important to note that all attacks based on Lemma 2, such as those that recover $a_{R-1} \oplus c_{R-1}$ and $a_{R-1} \oplus b_{R-1}$, are basically identical. Moreover, except for the requirement of double fault targets, the attacks based on Lemma 3 have an equivalent success rate to those based on Lemma 2. Therefore, we will not go into the details here.

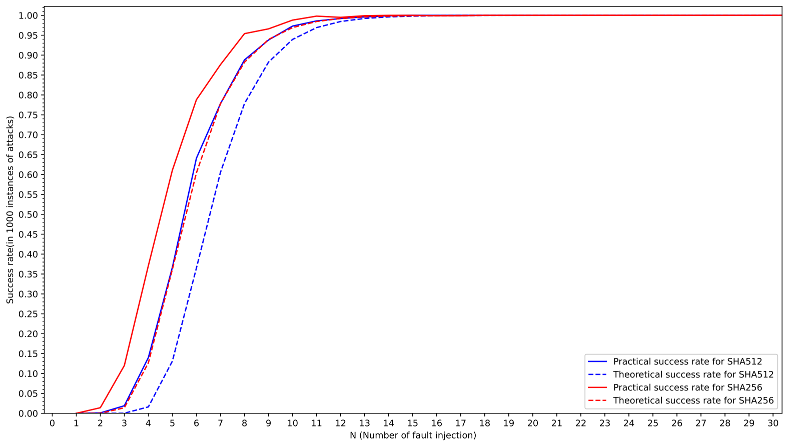

Figure 8: Success rate of recovering e in Lemma 2 when introducing N faults

Next, we validated the practical feasibility and success rate of Algorithm 2, which fundamentally relies on Lemma 2 and Lemma 4 to recover the left half of the final round input. We chose $\{b_{R-4}, c_{R-4}, ..., b_{R-1}, c_{R-1}\}$ as eight sequential targets and simulated 1 to 50 random faults (i.e., $N \in \{1, ..., 50\}$) for each target. We then executed 1,000 repeated instances of the attack in Algorithm 2 for each N.

The experimental and theoretical results are shown in Figure 9. The red and blue lines represent the respective success rates of SHA256 and SHA512. Dashed lines denote theoretical success rates, while solid lines denote experimental success rates for successfully recovering the lower $n-1$ ($T=1$) bits of the registers in Algorithm 2 for SHA256 and SHA512, depending on N and e (as specified in Lemma 4). Dotted lines indicate the experimental success rate for successfully recovering $n-T$ ($T \geq 1$) bits of the registers in Algorithm 2 for SHA256 and SHA512 (recovered success rate for short), without considering the cases of $\{e_{i+2}, e_{i+1}, e_i\} = \{0,0,0\}$ or $\{1,1,1\}$ in Lemma 4 and depending only on N.

We can see that the experimental success rates of recovering all the lower $n-1$ bits for SHA256 and SHA512 are consistently around 97% and 95%, respectively, when N is greater than 9. The theoretical average success rate reaches 99% when N is greater than 17/20 for SHA256/SHA512. The recovered success rates are close to 100% when N is greater than 15, which is slightly higher than the theoretical average success rates.

The above results show a slight divergence between the experimental and theoretical success rates. This divergence is primarily due to the fact that our theoretical success rates are based on the expected value of the number of known σ_i^j values, which depend on the randomness of e as stated in Lemma 4. However, in the actual experiments, we used only eight random numbers (i.e., $\{b_{R-4}, c_{R-4}, ..., b_{R-1}, c_{R-1}\}$), which may explain the observed discrepancy.

In addition, in cases where the recovered bits are correct but their number is less than $n-1$ (i.e., $T > 1$) with a probability of 3% for SHA256 (or 5% for SHA512) when $N \geq 15$, the average value of T (the average number of unrecoverable bits) remains at 3, and its maximum value is 5. This means that in the most unfavorable scenarios (with a probability of 3% or 5%), 3 bits will need to be guessed. In more common circumstances, it's only the most significant bit that needs to be guessed.

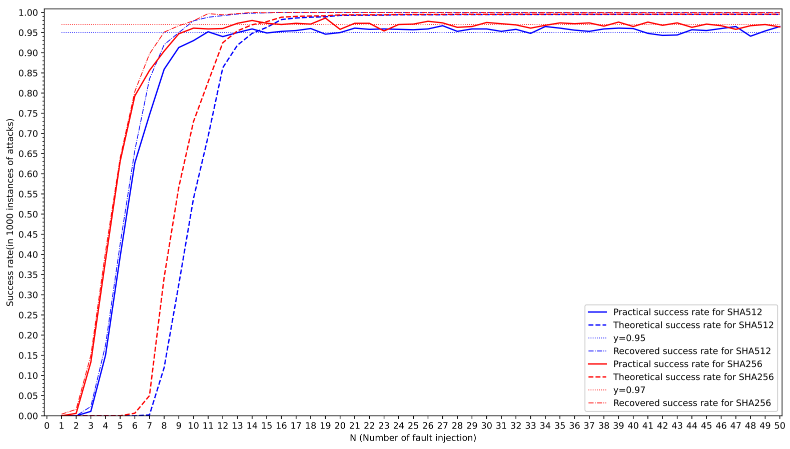

Figure 9: Success rate of recovering the left half of the final round

For the left half of the $R-1$-th input, as N approaches 15, the number of bits to guess, T, is generally 1 (with a probability of 95% for SHA512, 97% for SHA256) and rarely exceeds 5, achieving similarly high success rates (see Figure 9). In addition, 928 faults (i.e., $62N-2$) are required to recover the message block and the initial vector in the final compression function of SHA2 in the entire attack.

In conclusion, based on the random fault model, we achieve a near-perfect success rate of approximately 100% for the right half of the $R-1$-th input as N approaches 15 (see Figure 8). For the left half of the $R-1$-th input, as N approaches 15, the number of bits to guess, T, is generally 1 with a probability of 95% for SHA512, 97% for SHA256.

Moreover, T is less than 5 with a probability approaching 100%, achieving similarly high success rates (see the recovered success rate in Figure 9). In addition, 928 faults (i.e., $62N-2$) are required to recover the message block and the initial vector in the final compression function of SHA2 in the entire attack.

As the aforementioned results are derived from simulations, it is not necessary to distinguish between all potential faults. In order to identify the desired faults in practical experiments, it is recommended that the following steps be taken to determine the location and timing of fault injection.

- Observe the characteristics of the power traces from the oscilloscope in order to determine the time for each round.
- To identify the presence of a fault, it is necessary to introduce a fault during the scanning of the chip and subsequently observe the resulting hash values. As listed in Table 4, the faulty outputs resulting from introducing faults into intermediate registers are enumerated. For example, if the hash values $\{A, E, G\}$ differ from the correct values, it can be inferred that the fault injection occurred precisely at the location of f_{R-1}. Note that the faulty outputs are identical when both d_{R-1} and h_{R-1} are disturbed by fault injection.
- Fix the location of chip and return to the time of the $R-2$-th (beginning with 0) round, then introduce faults at this location. If $\{A, B, E, F, G\}$ are faulty, the corresponding location of fault injection is for d_{R-1}. If $\{A, B, E, F\}$ are faulty, the location is for h_{R-1}. By following these steps, the locations of all the registers can be determined.
- For each round of eight input registers, fix the location on the chip and adjust the time of the i-th ($i \in \mathbb{Z}_{\mathbb{R}}$) round (as determined by the aforementioned steps) to perform fault injection.

This method enables the precise locating and timing of the targeted register.

Table 4: The cases of faulty hash outputs when introducing faults into different targets

target	a_{R-1}	b_{R-1}	c_{R-1}	d_{R-1}	e_{R-1}	f_{R-1}	g_{R-1}	h_{R-1}
faulty Output	$\{A, B\}$	$\{A, C\}$	$\{A, D\}$	$\{A, E\}$	$\{A, E, F\}$	$\{A, E, G\}$	$\{A, E, H\}$	$\{A, E\}$

6 Conclusion

Our study has identified distinct differential fault properties of Boolean functions in SHA2, which enables us to propose a new differential fault attack that effectively targets SHA2, HMAC-SHA2, SHACAL-2, and even SM3. The effectiveness of our attacks is substantiated by rigorous theoretical proofs and experimental verifications, which indicate that they pose a significant threat to SHA2, with a success probability of at least 95% and requiring a guess of fewer than 5 bits. In particular, approximately 928 faults enables the recovery of the final message block and its initial vector.

Our findings reveal the essential attributes of Boolean functions in SHA2 and their potential threat for security. Moreover, they extend the scope of research on fault attacks and offer insights into traditional differential analysis. It can be reasonably inferred that other algorithms with analogous Boolean functions are similarly vulnerable to our attack. Future research will investigate the differential characteristics of Boolean functions in SHA3 under a 64-bit random word fault model, aiming to evaluate its resilience against potential fault attacks. Additionally, applying our approach to stream ciphers such as

A5/1 and A2U2 could yield valuable insights. Notwithstanding these contributions, our attack is not without limitations, including the lack of practical verification on devices implementing SHA2. This limitation necessitates further study.

References

[BCK96] Mihir Bellare, Ran Canetti, and Hugo Krawczyk. Keying Hash Functions for Message Authentication. *Journal of Cryptology*, 9(3):185–211, 07 1996.

[BGS15] Nasour Bagheri, Navid Ghaedi, and Somitra Kumar Sanadhya. Differential Fault Analysis of SHA-3. In *International Conference on Cryptology in India (INDOCRYPT)*, pages 253–269. Springer, Cham, 2015.

[BJ01] Eli Biham and Antoine Joux. SHACAL-2: A 256-Bit Block Cipher. In *Fast Software Encryption — FSE 2001*, volume 2040 of *Lecture Notes in Computer Science*, pages 30–45. Springer, 2001.

[DMRL11] Firstname David, Firstname Mathieu, Damith C. Ranasinghe, and Torben Larsen. A2U2: A Stream Cipher for Printed Electronics RFID Tags. In *2011 IEEE International Conference on RFID*. IEEE, 2011.

[FLDV09] Pierre-Alain Fouque, Gaëtan Leurent, Réal Denis, and Frédéric Valette. Practical Electromagnetic Template Attack on HMAC. In *International Workshop on Cryptographic Hardware and Embedded Systems (CHES)*. Springer-Verlag, 2009.

[GWM16] Christopher H. Gebotys, Brian A. White, and Estefania Mateos. Preaveraging and Carry Propagate Approaches to Side-channel Analysis of HMAC-SHA256. *ACM Transactions on Embedded Computing Systems (TECS)*, 15(1):1–19, 2016.

[HH11] Ludger Hemme and Lars Hoffmann. Differential Fault Analysis on the SHA1 Compression Function. In *2011 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pages 54–62. IEEE, 2011.

[HLMS14] Ronglin Hao, Bao Li, Bingke Ma, and Ling Song. Algebraic Fault Attack on the SHA-256 Compression Function. *International Journal of Research in Computer Science*, 4(2):1–9, 2014.

[JLSH13] Kitae Jeong, Yuseop Lee, Jaechul Sung, and Seokhie Hong. Security Analysis of HMAC/NMAC by Using Fault Injection. *Journal of Applied Mathematics*, 2013, 2013.

[LAFW17] Pei Luo, Konstantinos Athanasiou, Yunsi Fei, and Thomas Wahl. Algebraic Fault Analysis of SHA-3. In *Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017*, pages 151–156. IEEE, 2017.

[LAFW18] Pei Luo, Konstantinos Athanasiou, Yunsi Fei, and Thomas Wahl. Algebraic Fault Analysis of SHA-3 under Relaxed Fault Models. *IEEE Transactions on Information Forensics and Security*, 13(7):1752–1761, 2018.

[LFZ+16] Pei Luo, Yunsi Fei, Liwei Zhang, et al. Differential Fault Analysis of SHA3-224 and SHA3-256. In *Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*, pages 4–15. IEEE, 2016.

[LLG09] Ruilin Li, Chao Li, and Chunye Gong. Differential Fault Analysis on SHACAL-1. In *2009 Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC)*. IEEE, 2009.

[MTMM07] Robert McEvoy, Michael Tunstall, Colin C Murphy, and William P Marnane. Differential Power Analysis of HMAC based on SHA-2, and Countermeasures. In *Information Security Applications: 8th International Workshop, WISA 2007, Jeju Island, Korea, August 27-29, 2007, Revised Selected Papers 8*, pages 317–332. Springer, 2007.

[Nat02] National Institute of Standards and Technology (NIST). Secure Hash Standard (SHS). Federal Information Processing Standards Publication 180-2, U.S. Department of Commerce, National Institute of Standards and Technology, 2002. with Change Notice including SHA-224.

[NHGG18] Saeed Nejati, Jan Horáček, Catherine Gebotys, and Vijay Ganesh. 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, August 27-31, 2018, Proceedings*, volume 11015 of *Lecture Notes in Computer Science*, pages 737–754. Springer International Publishing, 2018.

[Osw16] David Oswald. Side-channel Attacks on SHA-1-based Product Authentication ICs. In *Smart Card Research and Advanced Applications: 14th International Conference, CARDIS 2015, Bochum, Germany, November 4-6, 2015. Revised Selected Papers*, volume 9958 of *Lecture Notes in Computer Science*, pages 13–29. Springer International Publishing, 2016.

[SBB$^+$18] Niels Samwel, Lejla Batina, Guido Bertoni, Joan Daemen, and Ruggero Susella. Breaking ed25519 in Wolfssl. In *Topics in Cryptology–CT-RSA 2018: The Cryptographers' Track at the RSA Conference 2018, San Francisco, CA, USA, April 16-20, 2018, Proceedings*, pages 1–20. Springer, 2018.

[She14] Xuan Shen. *Differential Fault Attacks on SHACAL-2 and the MD5 Encryption Modes.* PhD thesis, National University of Defense Technology, 2014.

[Sho13] Abdulhadi Shoufan. A Fault Attack on a Hardware-based Implementation of the Secure Hash Algorithm SHA-512. In *2013 International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, pages 1–7. IEEE, 2013.

[SKP13] Marc Stevens, Pierre Karpman, and Thomas Peyrin. Differential Collisions for SHA-1. In *Annual International Cryptology Conference (CRYPTO)*, volume 8042 of *Lecture Notes in Computer Science*, pages 343–365, 2013.

[Sta16] State Cryptography Administration of the People's Republic of China. Information Security Technology: SM3 Cryptographic Hash Algorithm. China National Standard GB/T 32907-2016, National Technical Committee for Information Security Standardization, 2016. In Chinese. English Title translated by Author.

[WLLL10] Yuechuan Wei, Lin Li, Ruilin Li, and Chao Li. Differential Fault Analysis on SHACAL-2. *Journal of Electronics & Information Technology*, 32(2):318–322, 2010.

[WYYD05] Xiaoyun Wang, Hongbo Yu, Yiqun Yin, and Xuejia Dai. Finding Collisions in the Full SHA-1. In *Advances in Cryptology - CRYPTO 2005*, volume 3621 of *Lecture Notes in Computer Science*, page 17ÍC36. Springer, 2005.