Randomness Analyses of the Secure Hash Algorithms, SHA-1, SHA-2 and Modified SHA

Zeyad Al-Odat*, Assad Abbas†, Samee U. Khan*
*North Dakota State University
Fargo, ND, USA
†COMSATS Institute of Information Technology
Abbottabad, Pakistan
Emails: *zeyad.alodat@ndsu.edu, *samee.khan@ndsu.edu, †assadabbas@comsats.edu.pk

***Abstract*—This paper introduces a security analysis scheme for the most famous secure hash algorithms SHA-1 and SHA-2. Both algorithms follow Merkle Damgård structure to compute the corresponding hash function. The randomness of the output hash reflects the strength and security of the generated hash. Therefore, the randomness of the internal rounds of the SHA-1 and SHA-2 hash functions is analyzed using Bayesian and odd ratio tests. Moreover, a proper replacement for both algorithms is proposed, which produces a hash output with more randomness level. The experiments were conducted using a high performance computing testbed and CUDA parallel computing platform.**

***Keywords*—Hash; Secure hash algorithm; Bayesian; Randomness.**

I. Introduction

A powerful tool is needed for message authentication and verification. Secure Hash Algorithm (SHA) is the most popular tool to ensure integrity properties. SHA was developed and standardized by the National Institute of Standards and Technology (NIST) [1]. SHA was designed for compression purpose, where a message of any size is compressed to a fixed-length hash. To consider any cryptography hash function as a secure one, three challenges exist preimage, 2^{nd}preimage, and collision resistance. The preimage means that the SHA function is a one-way function. The 2^{nd}preimage means that there are no messages that have the same hash value. The collision resistance means that there is no possibility to generate the same hash from two or more different messages [2].

The primary hash algorithms follow two types of structures, Merkle Damgård and Sponge structures. The hash functions MD5, SHA-1, and SHA-2 follow the Merkle Damgård , while the official SHA-3 (Keccak) follows the Sponge structure. The MD5, and SHA-1 produce 128-bit, and 160-bit output hash, respectively. However, these three hash functions were exposed to collision attack by Wang *et al.* since 2005 [3]. Since then, the SHA-2 emerged to be the official hash standard with six flavors (224, 256, 384, 512, 512/224, and 512/256). Even with the existence of the SHA-2, some entities still using the SHA-1 and MD5 hash functions [4]. Lately, the NIST released the latest hash standard SHA-3 (Keccak) on 2015 after a competition that was held to select the winner among 64 candidates [5].

The designers of the hash functions try to generate a unique output hash for any input and make it random when compared to another output. To test the randomness property, the Pseudo-Random Number Generator (PRNG) is used as a statistical test. the NIST uses the statistical tests to verify the cryptography applications [6], [7]. However, these tests are general and they are only used to test the randomness of binary sequences from any source. Moreover, the SHAs are designed to have fixed input parameters, which are nonrandom, and fixed output length. Therefore, the NIST test suites may not give a fair result for the randomness analysis.

In this paper, randomness analyses are conducted to examine the secure hash algorithms (SAH-1 and SHA-2). Moreover, a modified design is proposed to improve the randomness of the internal rounds of the hash functions. The analyses are based on the *Bayes factor* and *odd ratio* tests. We utilized a large-scale experimental environment testbed and Compute Unified Device Architecture (CUDA) platform to test the design and compare it with existed hash standards.

The remainde of the paper is formulated as follows: A background information about the SHA-1, SHA-2, and *Bayesian odd ratio test* will be presented in Section 2; Section 3 presents the state of art in the related field; Section 4 elaborates the randomness test and all related analyses; Sections 5 presents the discussions and analyses; Conclusion will be presented in Section 6.

II. Preliminaries

To better understand the proposed randomness analyses scheme, brief descriptions about SHA-1 and collision attack needs to be addressed. The threat model is also included to describe the possible security issues that are related to our work.

A. SHA-1 standard

SHA-1 was published by the NIST in 1995 after undisclosed security concerns with the SHA-0 [8]. The SHA-1 maintains 160-bit hash output through 80 steps of compression function evaluations. The SHA-1 is considered as a part of the Merkle Damgård functions, where the input message is divided into number of blocks and processed sequentially.

The SHA-1 takes a message of a predefined size and process it using the SHA-1 round function F. This includes the preprocessing phase where the message is padded to make its

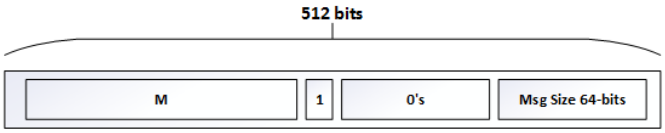

Fig. 1. Padding the message before processing

size congruent to 512. The padding is performed by appending "1", least number of zeros, and 64-bit message size, as shown in Fig.1.

After padding, the message is divided into equal size blocks (B_i) each of 512-bit. Each block is divided into 16 32-bit words and expanded into 80 32-bit words using Equation (1), which produces 2560-bit. These 80 words are represented by the W_t in functional block diagram of the SHA-1, as shown in Fig.2. The block words are assigned to the first 16 words of the W_t and the rest of the expanded words are calculated using these values and the message expansion equation. The figure shows that five working state variables (A, B, C, D, and E) are used to evaluate the intermediate hash value after each step. These variables are initialized with fixed 32-bit hexadecimal values (H_0=`67452301`, H_1=`EFCDAB89`, H_2=`98BADCFE`, H_3=`10325476`, and H_4=`C3D2E1F0`).

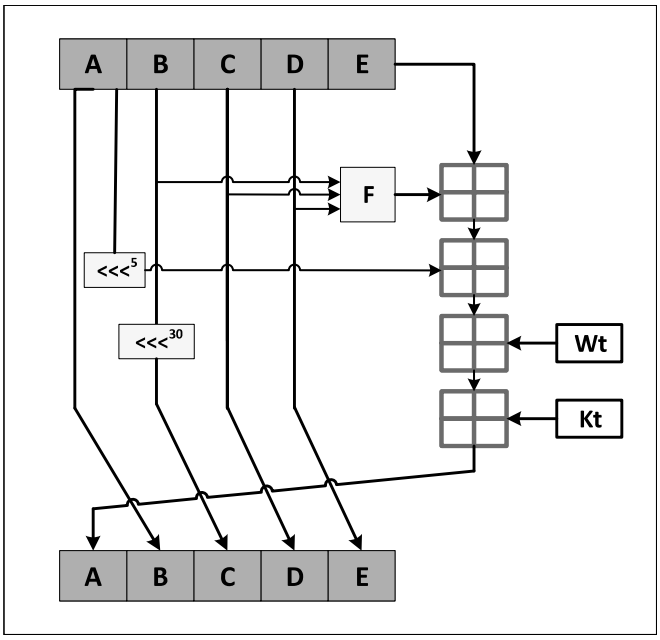

Fig. 2. Function block of the SHA-1

$$W_t = \begin{cases} B_i^t, & 0 \le t \le 15 \\ (W_{t-3} \oplus W_{t-8} \oplus W_{t-14} \oplus W_{t-16}) \lll 1, & 16 \le t \le 79, \end{cases} \quad (1)$$

where $\oplus$ is the logical XOR operation, and $\lll 1$ is the left rotate by 1-bit. Noting that the block B_i is divided into 16 words and these words are assigned to the first 16 values of the W_t.

The function block of the SHA-1 is processed 80 times (steps), where each 20 steps represent a group. Each group has a distinct round function f and constant K_t, as represented in Table I.

The expanded message words are processed as four groups. Each group consists of 20-steps, round constant k_t, and compression function f, as illustrated in Table I.

TABLE I
THE GROUPS FUNCTIONS (f) AND CONSTANTS (K_t)

Round(steps)	Function f(B,C,D)	Constant (K_t)
1 (0-19)	$(B \wedge C) \vee (\neg B \wedge D)$	`0x5A827999`
2 (20-39)	$(B \bigoplus C \bigoplus D)$	`0x6ED9EBA1`
3 (40-59)	$(B \wedge C) \vee (B \wedge D) \vee (C \wedge D)$	`0x8F1BBCDC`
4 (60-79)	$(B \bigoplus C \bigoplus D)$	`0xCA62C1D6`

The intermediate hash values (A,B, C, D, and E) change cyclically every step using (2), (3), (4), (5), (6).
For $i = 1$ to 80,

$$A_i = ROTL^5(A_{i-1}) + F_i(B_{i-1}, C_{i-1}, D_{i-1}) + E_{i-1} + W_i + K_i, \quad (2)$$

$$B_i = A_{i-1}, \quad (3)$$

$$C_i = RL^{30}(B_{i-1}), \quad (4)$$

$$D_i = C_{i-1}, \quad (5)$$

$$E_i = D_{i-1}. \quad (6)$$

The output hash is produced by taking the concatenation of the five working state variables ($A||B||C||D||E$), at the end of processing the last block. .

B. SHA-2 standard

SHA-2 follows Merkle Damgård structure model. Unlike SHA-1, SHA-2 maintains four different flavors (SHA-224, SHA-256, SHA-384, and SHA-512) that produce different outputs. Each flavor of the SHA-2 has a unique compression function and round constants. The SHA-2 uses eight variables to process the intermediate hash value (A, B, C, D, E, F, G, and H). The size of the variables depends on the desired flavour. For SHA-224 and SHA-256 flavors, the variable size is equal to 32-bit and 64-bit for the SHA-384 and SHA-512 flavors. SHA-2 maintains 64 steps for 224 and 256 versions and 80 steps for the others.

Like the SHA-1, the SHA-2 compression function evaluation requires a preprocessing phase. A message is padded and divided into blocks, as described in the SHA-1 definition. Each block is expanded using (7), then, the expanded values are assigned to W_t.

$$W_t = W_{t-16} + \sigma_0 + W_{t-7} + \sigma_1 \qquad 16 \le t \le n, \quad (7)$$

where σ_0 and σ_1 are represented by (8) and (9), respectively.

$$\sigma_0 = RR^{r1}(W_{t-15}) \oplus RR^{r2}(W_{t-15}) \oplus SR^{r3}(W_{t-15}), \quad (8)$$

$$\sigma_1 = RR^{q1}(W_{t-2}) \oplus RR^{q2}(W_{t-2}) \oplus SR^{q3}(W_{t-2}), \quad (9)$$

where $RR^n(X)$ rotates word X to the right by n bits, and $SR^n(X)$ shift right word X by n bits. For SHA-224 and

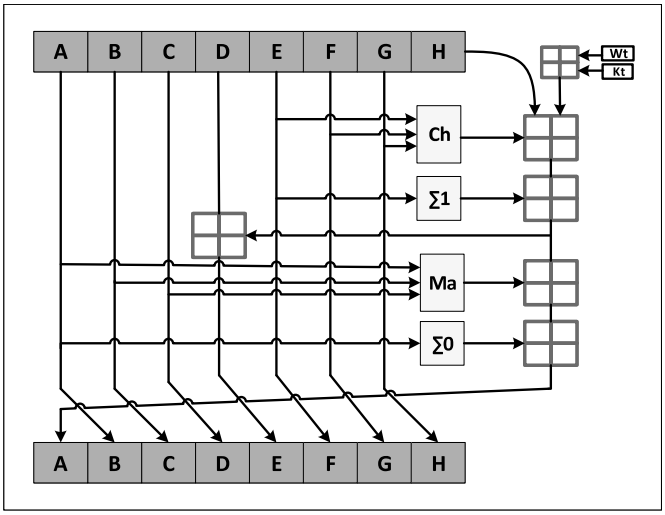

Fig. 3. Function block of the SHA-2

SHA-256 $n = 63$, $r1 = 7$, $r2 = 18$, $r3 = 3$, $q1 = 7$, $q2 = 19$, and $q3 = 10$, while $n = 79$, $r1 = 1$, $r2 = 8$, $r3 = 7$, $q1 = 19$, $q2 = 61$, and $q3 = 6$ for SHA-384 and SHA-512.

Fig.3 shows the functional block of the SHA-2 hash function. The intermediate hash values change, cyclically, after each step according to (10), (11), (12), (13), (14), (15), (16), (17), (18), (19)

$$T1 = H_{t-1} + W_t + K_t + Ch(E, F, G) + \sum\nolimits_1 \tag{10}$$

$$T2 = Maj(A, B, C) \sum\nolimits_0 \tag{11}$$

$$H = G, \tag{12}$$
$$G = F, \tag{13}$$
$$F = E, \tag{14}$$
$$E = D + T1, \tag{15}$$
$$D = C, \tag{16}$$
$$C = B, \tag{17}$$
$$B = A, \tag{18}$$
$$A = T1 + T2, \tag{19}$$

where Ch, Maj, $\sum_0$, and $\sum_1$ are represented by (20), (21), (22), and (23), respectively.

$$Ch(E, F, G) = (E \wedge F) \oplus (\neg E \wedge G), \tag{20}$$
$$Maj(A, B, C) = (A \wedge B) \oplus (A \wedge C) \oplus (B \wedge C), \tag{21}$$
$$\sum\nolimits_0 (V) = RR^{r1}(V) \oplus RR^{r2}(V) \oplus RR^{r3}(V), \tag{22}$$
$$\sum\nolimits_1 (V) = RR^{q1}(V) \oplus RR^{q2}(V) \oplus RR^{q3}(V), \tag{23}$$

where $\sum_0$ and $\sum_1$ are the message manipulators that are used inside the compression function. The values $r1 = 2$, $r2 = 13$, $r3 = 22$, $q1 = 6$, $q2 = 11$, and $q3 = 25$ are for the SHA-224 and SHA-256, while $r1 = 28$, $r2 = 34$, $r3 = 39$, $q1 = 14$, $q2 = 18$, and $q3 = 41$ are for the SHA-384 and SHA-512.

After processing all blocks and the corresponding working state variables, the final hash is produced by the concatenation of part or all of the working state variables (A, B, C, D, E, F, G, and H), as depicted by (24), (25), (26), and (27).

$$SHA_{224} \rightarrow A \| B \| C \| D \| E \| F \| G, \tag{24}$$
$$SHA_{256} \rightarrow A \| B \| C \| D \| E \| F \| G \| H, \tag{25}$$
$$SHA_{384} \rightarrow A \| B \| C \| D \| E \| F, \tag{26}$$
$$SHA_{512} \rightarrow A \| B \| C \| D \| E \| F \| G \| H, \tag{27}$$

noting that, the size of each working state variable is 32-bit for the SHA-224 and SHA-256 and 64-bit for the SHA-384 and SHA-512 hash functions.

C. Baysian Odd Ratio

The Bayesian theory is used to establish an evidence of the scientific theory that is applied to binomial distribution [9]. Let Θ denotes a *hypothesis*, D denotes the observed sample after running the model, $Pr(\Theta)$ the model's probability, and $Pr(D)$ the sample's probability. Then the Bayes's test states that the conditional probability of the model given the sample ($Pr(\Theta|D)$) is represented by (28)

$$Pr(\Theta|D) = \frac{Pr(D|\Theta)Pr(\Theta)}{Pr(D)}, \tag{28}$$

where it represents the conditional probability of $\Theta|D$.

The odd ratio test assumes that there are two hypotheses Θ_1 and Θ_2, where Θ_1 represents the model of random population and Θ_2 represents the model of nonrandom population [10]. Then the *odd ratio* test is calculated using Equation (29).

$$\frac{Pr(\Theta_1|D)}{Pr(\Theta_2|D)} = \frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} . \frac{Pr(\Theta_1)}{Pr(\Theta_2)}, \tag{29}$$

where $Pr(\Theta_1|D)/Pr(\Theta_2|D)$ is the posterior odd ratio, $Pr(D|\Theta_1)/Pr(D|\Theta_2)$ is the *Bayes factor*, and $Pr(\Theta_1)/Pr(\Theta_2)$ is the *prior odd ratio*.

III. Related Work

The randomness of cryptography functions was studied by different researchers. The studies used the NIST test suites to analyze the randomness of the bit sequences of certain cryptography functions, e.g., Advanced Encryption Standard (AES) and SHA. Doganaksoy *et al.* proposed a cryptographic randomness testing of block ciphers and SHA [11]. The proposed work, employed Strict Avalanche Criterion (SAC), Linear Span, and Coverage tests. These tests were used to evaluate the internal blocks of the block ciphers and hash functions. Other researchers employed the NIST test suite to test the randomness of block cipher functions.

Hellekalk and Wegenkittl proposed a model to test the block cipher applications. They use the NIST test suite by applying the PRNG [12]. The block ciphers were turned into PRNG and tested using the NIST suite. The empirical results showed that the AES gave interesting results regarding nonlinear RNG.

Kaminsky in [13] proposed a Bayesian statistical test for block cipher and message authentication code (MAC). The

proposed design employed the Bayesian test factor and odd ratios to measure the randomness level of block cipher functions. The proposed design was applied to the PRESENT and IDEA block cipher functions, and SipHash and SQUASH MAC functions. The randomness test of the functions showed the number of nonrandom rounds of each function and the ration of nonrandom to the random rounds.

The NIST test suite also employed to test the randomness of the reduced round SHA-3 [14]. The proposed work turned the internal blocks of the SHA-3 reduced round into PRNG then applied the NIST test suite on them. For the reduced round SHA-3, the randomness ratio was 21.3% for 3 rounds of the Keccak hash function. However, this result implies no significance since it was applied on 3 rounds instead of full rounds.

The ALgebric Normal Form (ANF) of random boolean functions to test symmetric ciphers and hash functions was presented in [15]. The proposed work presents new statistical testing for AES, SHA-0, MD4, MD5, Ripmid, and SHA-1 functions. The work employed Pseudo-Random Bit Generator (PRBG) to turn the internal blocks of the functions.

In our work, we analyze the randomness of the SHA functions directly, without turning the blocks into PRNG. Our work is performed on different SHA standards and a suggested modification is proposed. The proposed modification produces more randomness to the output hash than any other flavors of the SHAs.

In the subsequent section, the randomness test, logarithmic *Baeyian factor* test, and *odd ratio* test will be elaborated in details.

IV. The Randomness Test

A. Definition of The Randomness Test

The randomness is calculated after each round of the SHA-1 and SHA-2 algorithms. Figure 4 shows the block diagram of the randomness test. An input message (M) is fed to the compression function (SHA-1 or SHA-2). After each round of the processing phase, the intermediate hash value is divided into small bit groups (G). Then the randomness test is applied by XORing a certain bit group of the current output (H_i) with the same bit group of the previous round (H_{i-1}). Then, the odds ratio is computed by checking whether the output of the XORing is uniformly distributed. The results of all rounds and bit group categories are stored to apply the odd ratio equation on them. This will be applied for four different bit group sizes (1-bit, 2-bit, 4-bit, and 8-bit groups), where each bit group category has its distinct calculations.

B. Logarithmic Bayes Factor

The *Bayes factor* test is performed using the Bernoulli model, where n Bernoulli trials are performed. The success probability is denoted by χ, which represents the successes number of trials that follows the binomial distribution.

In our model, the *Bayes factor test* is performed on two models (Θ_1 and Θ_2). The probability of success for the Θ_1 and Θ_2 models are χ_1 and χ_2, respectively. Considering that

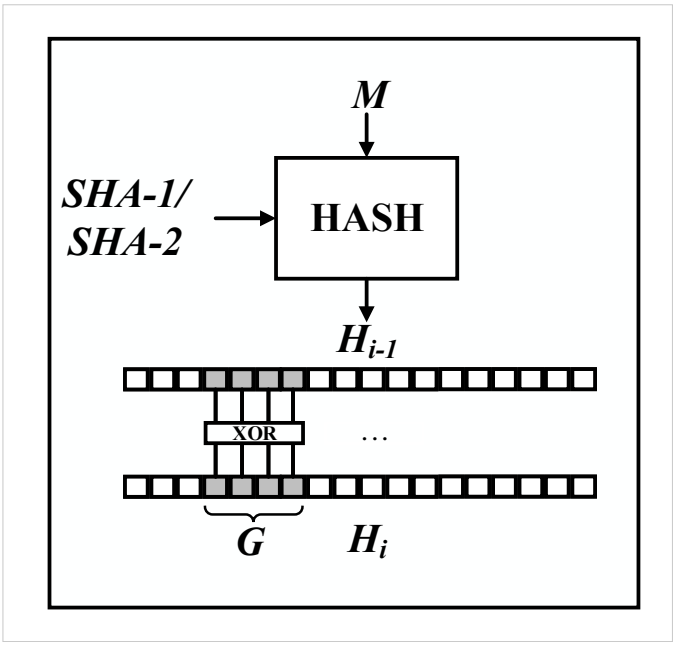

Fig. 4. The randomness test on the SHA-1 and SHA-2

the number of success of n trials is m. Then the *Bayes factor* equations for Θ_1 and Θ_2 are represented by a binomial distribution model, as seen in Equation (30).

$$\frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} = \frac{n!}{m!\,(n-m)!}\,p^m(1-p)^{n-m}(n+1), \quad (30)$$

where p represents χ_1 or χ_2.

To avoid the problem of floating point overflow, the logarithm of the *Bayes factor* is computed instead of the *Bayes factor* [16]. Moreover, the factorial ($n!$) is replaced by a gamma function. According to [17] the gamma function (Υ) of the factorial ($n!$) is represented by (31).

$$n! = \Upsilon(n+1)! \quad (31)$$

After substituting the Υ function the *Bayes factor* equation becomes:

$$\frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} = \frac{\Upsilon(n+1)}{\Upsilon(m+1)\,\Upsilon(n-m+1)}\,p^m(1-p)^{n-m}(n+1). \quad (32)$$

Then after applying the logarithm on Equation 32, the logarithmic *Bayes factor* becomes:

$$\begin{aligned}\log\frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} &= \log\Upsilon(n+1) - \log\Upsilon(m+1)\\ -\log\Upsilon(n-m+1) + m\log p + (n-m)\log(1-p)&\\ +\log(n+1).&\end{aligned} \quad (33)$$

The efficient computing of the *Logarithmic Bayes test* using computer programming is the reason for using the logarithmic form [16].

C. Odd Ratio

The *odd ratio* represents the ratio between the two hypotheses (Θ_1 and Θ_2) in the presence and absence of each one of them toward the other [18]. The *odd ratio test* is performed according to the procedure described in Algorithm 1. The algorithm accepts the input message (*M*) and the set of bit group sizes (*G*); in our work, we set G to have four different bit groups (1, 2, 4, and 8-bit groups). The algorithm works for the SHA-1 and SHA-2 hash standards, which must be determined before the beginning of the test. The bit group categories are processed one after the other, where for each bit group the corresponding SHA calculations are performed through all rounds (t). The value of t is determined according to the definition of SHA, as discussed in Section 2. The Intermediate Hash Value (IHV_{i-1}) of the previous round is assigned to the H_{i-1} and the IHV_i is assigned to the H_i.

Algorithm 1: Odd Ratio

Input: Message (M); `//One Block`
Input: G =[`1, 2, 4, 8`]
Output: Odd Ratio

$Determine(SHA)$; `//SHA-1 or SHA-2`
while $g \in G$ **do**
  for $i \leftarrow 1$ ***to*** $t-1$ **do**
    $H_i = IHV_i$
    H_{i-1} = IHV_{i-1}
    for $j \leftarrow 1$ ***to*** $t \mod G$ **do**
      $Res_i^{G_j}$ = $H_i^j \oplus H_{i-1}^j$
$\log \frac{Pr(D|\Theta_1)}{Pr(D|\Theta_2)} = \log[\frac{\Upsilon(n+2)}{\Upsilon(k+1)\Upsilon(n-k+1)} p^k (1-p)^{n-k}]$
Odd ratio

The IHV value is divided into groups, which are multiple of G. Then the corresponding results of XORing the current and previous analogous bit group position are stored in $Res_i^{G_j}$ array. After processing all bit groups and all internal rounds of the SHA function, the random selection of data (D) is performed by performing n Bernoulli trials considering round (i), bit group (g), and success (k). For each bit group (G), the success probability (p) is equal to 2^{-g}, where g is the corresponding bit group size. For each bit group, all k successes are counted and the corresponding *logarithmic Bayes factor* is computed for each round (i). Consequently, the *odd ratio* value for each round is the accumulated values of *logarithmic Bayes factor* of the corresponding round and bit group.

D. Modified SHA

The modified SHA design that we suggest works by employing the function of the SHA-1 and the function manipulators of the SHA-2. The proposed design works as follows:

1) **Padding**: The message padding technique in our design is different. We follow the $10*1$ technique where "1" is appended to the end of a message then followed by zeros and "1". In this padding method, the size of the message is not included in the padding, which protects the message from any targeted attack.
2) **Process**: After padding, the message is divided into blocks of 64-bit. Then, each block is expanded using the expansion equation of the SHA-512. The design works on 80 steps. These steps are divided into four stages, each stage comprises 20 steps. For each stage, the compression function of the SHA-1 is used, as shown in Table I. Each step maintains a distinct constant K. The constant values are generated by taking the least 64-bit of the fractional part of the $\sqrt[3]{Z_p}$ in hexadecimal, where Z_p represent the first 80 prime numbers.
3) **Output**: The output hash is produced by taking the concatenation of the eight working state variables to form 512-bit hash. For hash standard with shorter length, e.g., the SHA-1, the hash value is taken from the least significant 160-bit of the produced output.

V. Results and Discussions

The *odd ratio test* is performed on the SHA-1, SHA-2, and A modified SHA function. The modified SHA hash function performs different padding technique by using $10*1$ method. In this method, we excludes the message size from being processed during hash computations.

As the test performs a massive number of compression function calculations, the execution program was tested using GPGPU unit. The experiments were conducted on Chameleon, which is a configurable experimental environment for large-scale cloud research [19]. On Chameleon, we reserved a cluster lease with 2-Intel Xeon processors 3.2GHz with 56 threads, 128GB of RAM, and Tesla P100 GPU with 3584 cores.

TABLE II
Odd Ratio results for the SHA-1, SHA-512 and Modified SHA

Round	Odd Ratio		
	SHA-1	SHA-512	Modified
1	-1.78×10^8	-1.177×10^7	-1.214×10^8
2	-1.62×10^8	-1.187×10^7	-1.186×10^7
3	-1.55×10^8	-1.168×10^7	-1.159×10^7
4	-1.58×10^8	-1.59×10^7	-1.143×10^7
5	-1.43×10^8	-1.53×10^7	-6.32×10^6
17	-8.68×10^7	-520	-425
20	-7.43×10^6	-65.66	630
21	-6.95×10^5	478	648
27	-100.68	520	645
28	-15.68	680	745
29	200.68	598	635
78	578	633	788
79	598	812	596
80	620	599	623

Table II shows the results of the *odd ratio* test of the SHA-1, SHA-512, and modified SHA. For space limitation, we omit the results of some rounds. According to the *Bayesian odd ratio*, the number of nonrandom rounds for the SHA-1 is 28. This is because the *odd ratio* values of the first 28 rounds of the SHA-1 give negative values. The SHA-512 performs better than the SHA-1 with 20 nonrandom rounds. The modified version performs the best and produces 17 nonrandom rounds.

TABLE III
ALL TEST RESULTS

Hash Function	#Non-Random Rounds	Total #Rounds	Non-Random%
SHA-1	28	80	35
SHA-224	19	64	29.68
SHA-256	20	64	31.25
SHA-384	19	80	23.75
SHA-512	20	80	25
Modified SHA	17	80	21.25

Table III shows the results of performing the *odd ratio* test on the SHA-1, SHA-2, and Modified SHA functions. The SHA-1 produces 28 nonrandom rounds among 80 rounds with non-random percentage of 35%. The SHA-224 and SHA-256 produce approximately the same number of nonrandom rounds with 19 and 20, respectively. The modified SHA function shows the best results in case of nonrandom rounds with 17 non-random rounds.

VI. CONCLUSIONS

This paper presented a model that examines the randomness of the internal rounds of the SHA-1 and SHA-2 standards. A modified SHA design was proposed to improve the randomness of the internal rounds. The *Bayesian factor* and *odd ratio* tests were used to evaluate the hash functions. The CUDA platform was utilized to perform the experimental analyses, which was implemented with the help of Chameleon large-scale experiments testbed. The results showed that the Modified SHA produced more random rounds other than the other functions.

In the future, the same analyses will be applied to the SHA-3 hash function (Keccak). Moreover, more experiments will be conducted to test the Avalanche and Difference analyses for the SHA functions.

ACKNOWLEDGMENTS

The work of Samee U. Khan was supported by (while serving at) the National Science Foundation. Any opinion, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the National Science Foundation.

Results presented in this paper were obtained using the Chameleon testbed supported by the National Science Foundation.

REFERENCES

[1] FIPS PUB. Secure hash standard (shs). *FIPS PUB 180*, 4, 2012.
[2] Zeyad Al-Odat, Mazhar Ali, and Samee U Khan. Mitigation and improving sha-1 standard using collision detection approach. In *2018 International Conference on Frontiers of Information Technology (FIT)*, pages 333–338. IEEE, 2018.
[3] Xiaoyun Wang and Hongbo Yu. How to break md5 and other hash functions. In Ronald Cramer, editor, *Advances in Cryptology – EUROCRYPT 2005*, pages 19–35, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.
[4] Zeyad Al-Odat and Samee Khan. The sponge structure modulation application to overcome the security breaches for the md5 and sha-1 hash functions. In *2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)*, volume 1, pages 811–816. IEEE, 2019.
[5] NIST DRAFT FIPS PUB and PUB FIPS. 202. *SHA-3 Standard: Permutation-Based Hash and Extendable-Output Functions*, 2014.
[6] Juan Soto. Statistical testing of random number generators. In *Proceedings of the 22nd National Information Systems Security Conference*, volume 10/99, page 12. NIST Gaithersburg, MD, 1999.
[7] Lawrence Bassham, Andrew Rukhin, Juan Soto, James Nechvatal, Miles Smid, Elaine Barker, Stefan Leigh, Mark Levenson, Mark Vangel, David Banks, N. Heckert, and James Dray. A Statistical Test Suite for Random and Pseudorandom Number Generators for Cryptographic Applications. *CSRC | NIST*, Apr 2010.
[8] Secure Hash Standard. Fips pub 180-1. *National Institute of Standards and Technology*, 17:15, 1995.
[9] Robert E Kass and Adrian E Raftery. Bayes factors. *Journal of the american statistical association*, 90(430):773–795, 1995.
[10] Jeffrey N Rouder, Paul L Speckman, Dongchu Sun, Richard D Morey, and Geoffrey Iverson. Bayesian t tests for accepting and rejecting the null hypothesis. *Psychonomic bulletin & review*, 16(2):225–237, 2009.
[11] Ali Doganaksoy, Baris Ege, Onur Koçak, and Fatih Sulak. Cryptographic randomness testing of block ciphers and hash functions. *IACR Cryptology ePrint Archive*, 2010:564, 2010.
[12] Peter Hellekalek and Stefan Wegenkittl. Empirical evidence concerning aes. *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, 13(4):322–333, 2003.
[13] Alan Kaminsky. The coincidence test: a bayesian statistical test for block ciphers and macs, 2013.
[14] Ali Doganaksoy, Baris Ege, Onur Koçak, and Fatih Sulak. Statistical analysis of reduced round compression functions of sha-3 second round candidates. *IACR Cryptology ePrint Archive*, 2010:611, 2010.
[15] Eric Filiol. A new statistical testing for symmetric ciphers and hash functions. In *International Conference on Information and Communications Security*, pages 342–353. Springer, 2002.
[16] William H Press, Saul A Teukolsky, William T Vetterling, and Brian P Flannery. *Numerical recipes 3rd edition: The art of scientific computing*. Cambridge university press, 2007.
[17] Eric W. Weisstein. Gamma Function, 2019 (accessed 20. Jul). "http://mathworld.wolfram.com/GammaFunction.html".
[18] Magdalena Szumilas. Explaining odds ratios. *Journal of the Canadian academy of child and adolescent psychiatry*, 19(3):227, 2010.
[19] Kate Keahey, Pierre Riteau, Dan Stanzione, Tim Cockerill, Joe Mambretti, Paul Rad, and Paul Ruth. Chameleon: a scalable production testbed for computer science research. In Jeffrey Vetter, editor, *Contemporary High Performance Computing: From Petascale toward Exascale*, volume 3 of *Chapman & Hall/CRC Computational Science*, chapter 5. CRC Press, Boca Raton, FL, 1 edition, 2018.