Journal of Applied Sciences

ISSN 1812-5654

science alert

ANSInet
an open access publisher
http://ansinet.com

Journal of Applied Sciences 13 (5): 673-682, 2013
ISSN 1812-5654 / DOI: 10.3923/jas.2013.673.682

Diffusion and Statistical Analysis of STITCH-256

[1]N. Jamil, [2]R. Mahmod, [3]M.R. Z'aba, [2]N.I. Udzir and [2]Z.A. Zukarnain
[1]College of Information and Technology, Universiti of Tenaga Nasional, KM7, Jalan IKRAM-UNITEN, Serdang, 43600 Selangor, Malaysia
[2]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, 43600 Seri Kembangan, Selangor, Malaysia
[3]Cryptography Lab., MIMOS Berhad, Technology Park Malaysia, 57000 Bukit Jalil, Kuala Lumpur, Malaysia

Abstract: Hash function is an important cryptographic primitive used in a wide range of applications, for example, for message authentication and in digital signatures. MD 4/5 and SHA-0/1/2 are examples of widely used hash functions, but except for SHA-2 (SHA-224, 256, 384, 512), they were all broken in 2005 after more than a decade of use. Since, then, the structure and components of cryptographic hash functions have been studied and revisited extensively by the cryptographic community. STITCH-256 was introduced to overcome problems faced by the MD- and SHA-family hash functions. STITCH-256 employs the Balanced Feistel network and its step operation runs in four parallel branches. The algorithm was claimed to produce good diffusion and its outputs were claimed to be random. To evaluate its suitability for such purposes, avalanche and empirical statisti- cal tests are commonly employed to show that there is empirical evidence supporting the claims. In this study, we report on the studies that were conducted on the 1000 sample of 256 bit of output from STITCH-256 algorithm. The studies include the study of diffusion and statistical properties of STITCH-256 using avalanche test and nine statistical tests. The results suggest that the claims were true where STITCH-256 produces good avalanche effect, thus good diffusion property and its outputs appear random.

Key words: Avalanche, cryptography, hash function, statistical analysis, STITCH-256

INTRODUCTION

Cryptographic hash function is a function f that accepts arbitrary length of input message m and produces a fixed length of output message h, f [0, 1]*→f [0, 1]^n. The length of the input message can be a single byte of an alphabet or hundreds megabytes of a video file. The input message m is sometimes called pre-image and the output message h(m) is called hash value. To be used in cryptographic applications such as digital signatures, the function has to satisfy the following:

- **Pre-image resistance:** Given a hash value h(m), it is infeasible to find any message m which hashes to h(m). This property is also known as one- wayness
- **Second pre-image resistance:** Given a hash value h(m) and its corresponding message m, it is infeasible to find another message m' such that h(m) = h(m')
- **Collision resistance:** Given a hash value h(m), it is infeasible to find any two distinct messages m and m', where m ≠ m' such that h(m) = h(m')

Finding a pre-image or a second pre-image by brute force attack would require 2^n operations and finding a collision by birthday attack would need approximately $2^{n/2}$. Damgard (1990) showed that the strongest property is collision resistance where preservation of collision resistance implies preservation of second pre-image resistance and preservation of second pre-image resistance implies preservation of pre-image resistance. Thus, collision resistance is the most important property for cryptographic hash function and most of the attacks were aimed at breaking this property.

The most popular and widely used hash functions are known as dedicated hash functions, whose design method is a serial successive iteration of step operation. These hash functions include MD5 (Rivest, 1992b), HAVAL (Zheng *et al*., 1993; FIPS 180, 1993) and SHA-1 (NIST, 1995). These hash functions are also known as MDx-class hash functions since they have some commonalities in their design, i.e., same design principle with MD4 (Rivest, 1992a). In principle, the design of MDx-class hash functions follow Merkle-Damgard

Corresponding Author: N. Jamil, College of Information of Technology, Universiti of Tenaga Nasional, KM7, Jalan IKRAM-UNITEN, Serdang, 43600 Selangor, Malaysia Tel: +603-8921-2020 Fax: +603-8921-2161

iterative construction (Merkle, 1989; Damgard, 1990). In this construction, the security of the hash function H is reduced to the security of the compression function of f. However, most of the compression functions of MDx-class hash functions are light and relatively simple. This has led to the failure of these hash functions to preserve a property of collision resistance. After more than a decade of use, all of these hash functions were broken in series of attacks by Van Rompay *et al.* (2003), Wang *et al.* (2004, 2005a, b), Wang and Yu (2005), Biham *et al.* (2005) and Yu *et al.* (2006), to name a few. The successful attacks against these hash functions are differential attacks. The attacks aimed at producing zero output difference with non-zero input difference in the input messages (Wang *et al.*, 2005c). The hash function is considered as insecure if one has successfully developed an algorithm that can vanishes such difference in its compression function. As the compression function of MDx-class hash functions is light and relatively simple, it is easy to construct differential characteristic in it that leads to collisions. This could probably due to poor diffusion in the step operation of MDx-class hash function, i.e., did not produce a good avalanche effect.

STITCH-256 (Jamil *et al.*, 2012) was introduced to recommend a new step operation permuted in a so-called stitching permutation. The description of the step operation in STITCH-256 is given in the following section. It was claimed that the outputs of STITCH-256 appear to be random and the step operation produces good avalanche effect. To evaluate its suitability for cryptographic purposes, avalanche and empirical statistical tests are commonly employed to show that there is empirical evidence supporting the claim. In this study, we report on the studies that were conducted on the output bits from STITCH-256 algorithm. The results suggest that the claims were true and STITCH-256 is suitable to be used for cryptographic purposes.

DESCRIPTION OF STITCH-256

Table 1 shows the basic notation used to describe STITCH-256. STITCH-256 employs Merkle-Damgard iterative construction in order to produce a fixed size of output from arbitrary length of input and a variant of Davies-Meyer mode of operation as depicted in Fig. 1. In this variant mode of operation, it takes a message block M and the previous hash value h_i. For the first iteration, the h_0 is the IV. The output from left most and right most branch, B1 and B4, are then added with the previous hash value h_i and output from B2 and B3. The output from this operation is then XORed together and added once again with h_i. The output is h_{i+1}. This process is repeated until all the message blocks M are processed. If the two

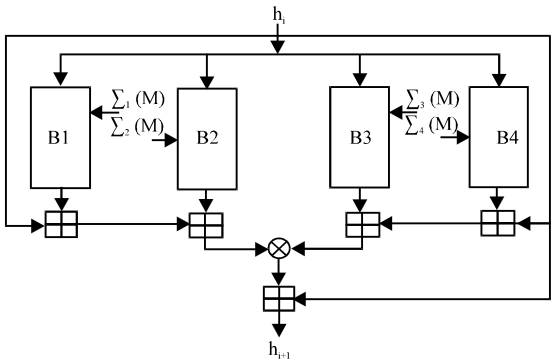

Fig. 1: Mode of operation of STITCH-256

Table 1: Notation and description of symbols used in algorithm equations

CA rules	Representation as boolean function
29	$x \oplus ((x \oplus \bar{z}) \vee y)$
39	$((x \oplus \bar{y}) \vee z) \oplus y$
27	$x \oplus ((x \oplus \bar{y}) \vee z$
46	$(x \vee y) \oplus (y \vee z)$
53	$(x \vee (y \oplus \bar{z})) \oplus y$
58	$(x \vee (y \oplus z)) \oplus y$
71	$((x \oplus \bar{z}) \vee y) \oplus r$
83	$(x \vee (y \oplus \bar{z})) \oplus r$

branches on the right are denoted as a big function E1 and the two branches on the left are denoted as a different big function E2, this variant of Davies-Meyer mode can be formulated as:

$$h_{i+1} = E1 \oplus E2 + h_i$$

The update can also be seen as the h_i is updated to h_{i+1} by computing:

$$h_{i+1} = [h_i + B1(h_i, \Sigma_{j,1}(M)) + B2(h_i, \Sigma_{j,2}(M))] \oplus [h_i + B4(h_i, \Sigma_{j,3}(M)) + B3(h_i, \Sigma_{j,4}(M))] + h_i$$

where, $\Sigma_{j,i}(M)$ for j = 1, 2, 3, 4 and 0 = i = 15, is the sum of sixteen 32 bit message words W_i that has been rearranged according to Fig. 2. Each line needs two $\Sigma_{j,i}(M)$ and they are different in each line. This gives a total of eight different orderings for the compression function.

STITCH-256 processes 512 bit blocks of input message and produces 256 bit block of hash value. The input message is first appended by a single bit 1 to the least significant bit of the input message, followed by as many zero as possible until the length of the message is 448 modulo 512 which is filled up by the 64 bit message length modulo 264.

COMPRESSION FUNCTION OF STITCH-256

The compression function of STITCH-256 runs in parallel of four branches, B1, B2, B3 and B4 as illustrated

in Fig. 1. There are three main components in the compression function of STITCH-256 namely, message expansion, step operation and Boolean functions.

Message expansion of STITCH-256: The message expansion of STITCH-256 follows these formulas:

$$W_t = M_t \text{ for } 0 \le t \le 15$$

$$W_t = \gg 11\ (s_0(W_i\text{-}16, W_i\text{-}15, W_i\text{-}14, W_i\text{-}13) + s_1(W_{i\text{-}12}, W_{i\text{-}11}, W_{i\text{-}10}\ W_{i\text{-}9}) \oplus SV_0) + \gg 13\ (s_0(W_{i\text{-}8}, W_{i\text{-}7}, W_{i\text{-}6}, W_{i\text{-}5}) + s_1(W_{i\text{-}4}, W_{i\text{-}3}, W_{i\text{-}2}, W_{i\text{-}1}) \oplus SV_1) \text{ for } 16 \le t \le 63$$

where, s_0 (w, x, y, z) = w⊕x⊕y⊕z and s_1 (w, x, y, z) = w+x+y+ z.

It uses two salt values to support the message expansion of STITCH-256, where SV_0 = 67452301 and SV_1 = 41083726. The message expansion of STITCH-256 is illustrated as in Fig. 3. In STITCH-256, the 512 bit message input is expanded to 32 bit message words. This gives the output of message expansion as 1024 bits. All the message words $W_i \ldots W_{31}$ are then reordered to give different message ordering in each line. The compression function of STITCH-256 requires eight Σ_i (M) for the whole function as described earlier and the orderings are depicted in Fig. 3. It shows the input order of message words M_0,..., M_{15} applied to Bj($1 \le j \le 4$) branches. The number with an asterisk denotes the message words W_i for $16 \le i \le 31$. This means 1' refers to message words W_{16}, 2' refers to message word W_{17}, so on and so forth.

Br	Msg Ord	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
	2'	15'	0'	1'	2'	3'	4'	5'	6'	7'	8'	9'	10'	3'	12'	13'	14'
2	3	14	15	0	1	10	11	4	5	6	7	8	9	2	3	12	13
	4'	13'	14'	15'	0'	9'	10'	11'	4'	5'	6'	7'	8'	1'	2'	3'	12'
3	5	12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
	6'	3'	12'	13'	14'	7'	8'	9'	10'	11'	4'	5'	6'	15'	0'	1'	2'
4	7	2	3	12	13	6	7	8	9	10	11	4	5	14	15	0	1
	8'	1'	2'	3'	12'	5'	6'	7'	8'	9'	10'	11'	4'	13'	14'	15'	0'

Fig. 2: Message orderings for four branches in STITCH-256

Step operation of STITCH-256: Step operation of STITCH-256 is designed as shown in Fig. 4. The step operation of STITCH-256 can be seen as a Balanced Feistel Network where registers A, B, C, D is one half and registers E, F, G, H is the other half. Here, we simply name the division as the left and right wings, respectively. Each wing has different step operation, different Boolean functions and different constants. We divide the step operations in STITCH-256 into two phases; the first phase serves as a heavy step operation and the second phase involves only permutation. We will use the notation of left and right wings throughout this study.

In the step operations of STITCH-256, two distinct Boolean functions F3, F4 are used in the right wing and one Boolean function F2 is used in the left wing. In the right wing, the registers and message words are processed through seven steps whereas for the left wing the registers are processed through six steps, prior to the initial permutation. Registers A_i, B_i, C_i and Di are first processed in the left wing during the first phase, followed by initial permutation and finally processed in the first phase of right wing. This means the registers are

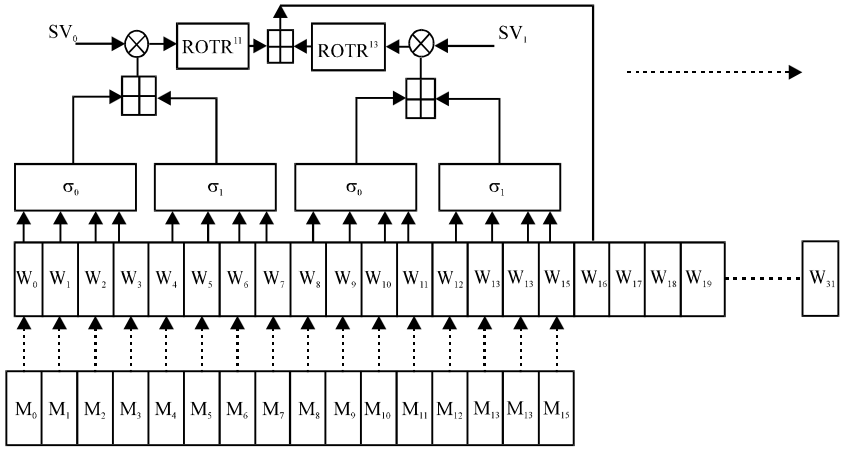

Fig. 3: Message expansion of STITCH-256

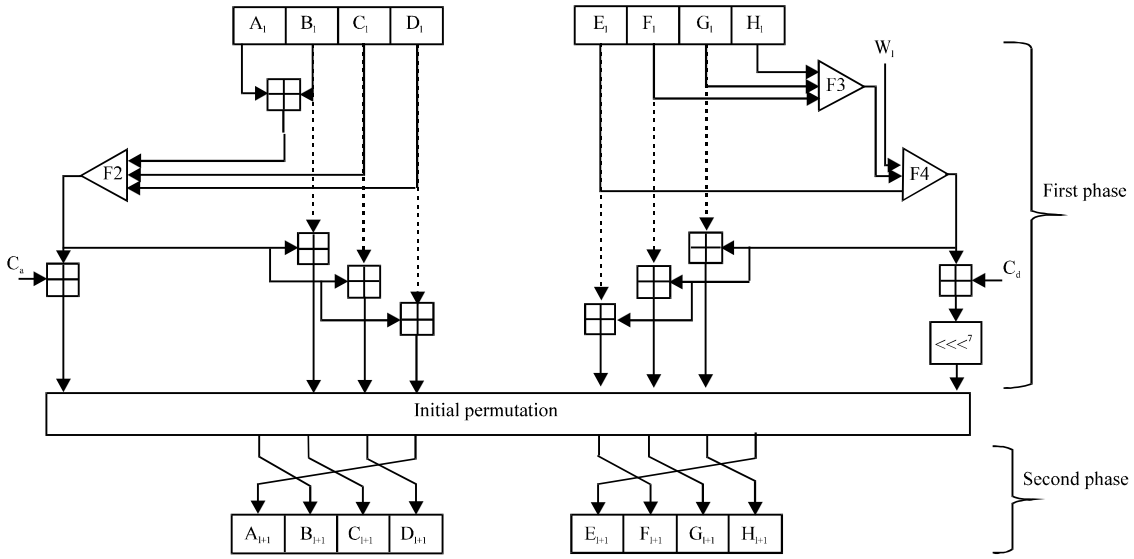

Fig. 4: Step operation of STITCH-256

processed in two sets of non-overlapping step operations. After being processed in the right wing, registers A_i, B_i, C_i and D_i are permuted in the second phase and used to update the values of E_{i+1}, F_{i+1}, G_{i+1} and H_{i+1}. This process occurs vice versa and in parallel, for registers E_i, F_i, G_i and H_i, where they finally update the values of A_{i+1}, B_{i+1}, C_{i+1} and D_{i+1}.

Let the outputs for A_i, B_i, C_i, D_i, E_i, F_i, G_i and H_i after the first stage are denoted as A', B', C', D', E', F', G' and H', respectively, defined as follows:

$$
\begin{aligned}
A'_i &= Di+F2((A_i+B_i), C_i, D_i)\\
B'_i &= \varphi_{i,j}+F\ 2((A_i+B_i), C_i, D_i)\\
C'_i &= B_i+F2((A_i+B_i),C_i, D_i)\\
D'_i &= C_i+F2((A_i+B_i), C_i, D_i)\\
F'_i &= E_i+F4\ (E_i, W_i, F\ 3(F_i, G_i, H_i))\\
G'_i &= F_i+F\ 4(E_i, W_i, F\ 3(F_i, G_i, H_i))\\
H'_i &= G_i+F\ 4(E_i, W_i, F\ 3(F_i,G_i,H_i))\\
E'_i &= «^7\ (\varphi_{i,j}+F\ 4(E_i, W_i, F\ 3(F_i, G_i, H_i))
\end{aligned}
$$

A single step operation in each wing updates its new registers A_{i+1}, B_{i+1}, C_{i+1}, D_{i+1}, E_{i+1}, F_{i+1}, G_{i+1} and H_{i+1} by producing the following outputs:

$$
\begin{aligned}
A_{i+1} &= F2((E_i+F_i), G_i, H_i)+H_i\\
B_{i+1} &= \varphi_{i,j}+F\ 2((E_i+F_i), G_i, H_i)\\
C_{i+1} &= F\ 2((E_i+F_i), G_i, H_i)+F_i\\
D_{i+1} &= F\ 2((E_i+F_i), G_i, H_i)+G_i\\
E_{i+1} &= «^7(\varphi_{i,j},+F\ \ 4(A_i, W_i, F\ 3(B_i, C_i, D_i)))\\
F_{i+1} &= A_i+F\ 4(A_i, W_i, F\ 3(B_i, C_i, D_i))\\
G_{i+1} &= B_i+F\ 4(A_i, W_i, F\ 3(B_i, C_i, D_i))\\
H_{i+1} &= C_i+F\ 4(A_i, W_i, F\ 3(B_i, C_i, D_i))
\end{aligned}
$$

where, $\varphi_{i,j}$ is different for each branch and the details can be found by Jamil *et al.* (2012). The step operation of STITCH-256 is illustrated in Fig. 4.

As described earlier, the outputs after initial permutation in the left wing, A', B', C', D', are processed in the first phase of the right wing. They are not only permuted, but also become the new value for E_{i+1}, F_{i+1}, G_{i+1}, H_{i+1} after the permutation in the second phase of the right wing. Then, the new values in registers E, F, G, H are iterated in the first stage of the right wing step operation, permuted and become the new value for A_{i+1}, B_{i+1}, C_{i+1}, D_{i+1}. This happens similarly for the output after initial permutation in the right wing.

Note that the initial permutation is the same as the permutation in the second phase, where it permutes the registers one block to their right. The entire process looks like a stitching permutation when one views it from top. Thus it gives the name as STITCH-256. The stitching permutation is designed to maximize the propagation of intermediate chaining variables, thus lower the probability to construct differential characteristics. The stitching permutation when it is viewed from top is illustrated in Fig. 5. Finally, each wing in a branch processes sixteen message words and this makes each branch in the compression function of STITCH-256 to have only sixteen rounds of step operation.

Boolean Functions in STITCH-256: STITCH-256 derives its Boolean functions from a set of one-dimensional Cellular Automata (CA) rules (Wolfram, 1984). Table 2 shows the CA rules represented as Boolean functions, used in STITCH-256.

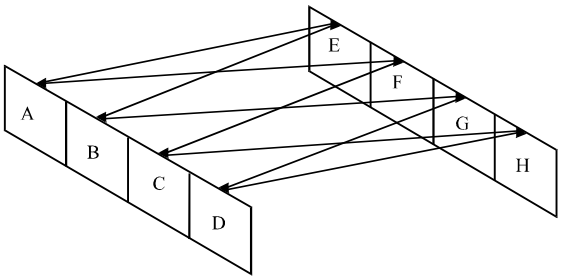

Fig. 5: Stitching permutation in STITCH-256

Table 2: CA rules used in STITCH-256

Notation	Description
$A \oplus B$	XOR operation of A and B
A+B	Addition of A and B modulo 2^{32}
M_i	The i-th block of the 32 bit input message M
W_i	The i-th block of the 32 bit expanded input message W
$\ll^n(A)$	Bit rotation of A by n position/s to the left
$\ll^n(A)$	Bit shift of A by n position/s to the left
$\gg^n(A)$	Bit rotation of A by n position/s to the right
$\gg^n(A)$	Bit shift of A by n position/s to the right
N	Number of rounds in the message expansion
h_i	i-th chaining variable

AVALANCHE EFFECT OF STITCH-256

Having a strong avalanche effect is seen as a continuous effect where small differences are mapped to big differences. However, as we only work on discrete spaces here, all functions are also continuous. Therefore, continuity itself cannot be of any help for quantifying the avalanche effect. In our study, we are interested in the average behavior of the step operations of STITCH-256. Here, we present our empirical results on the effects of inducing small differences. In particular, we study the behavior of the output bits in the full round of step operations in STITCH-256 for various samples.

Experiment: In principle, any cryptographic algorithm must have a property that the redundancies at the input should not leak any information in the output. Based on this understanding, we construct three types of data sets, each having different structures. Then, we observe the avalanche effect in the output bits and observe the structure in some of the test vector samples. The types of data sets are as follows:

- **Low density message:** The low density message is a message formed by binary strings of low weight. In our case, the number of '1's in the binary strings is less or equal to 3
- **High density message:** The high density message is a message formed by binary strings of high weight. In other words, the inputs of the low density messages are complemented bitwise
- **Random message:** The random message is a message formed by a random distribution between bits 0 and 1 in the sequence

For each of the data type, we prepare 1000 sequences of 512 input bits each which makes up a total of 512000×3 = 1536000 bits tested in the experiments.

Empirical results and analysis: Here, we present our empirical results on the effects of inducing a single bit difference for a full round of STITCH-256 algorithm in three different data categories. For our analysis, we chose the following parameter:

- The number of steps s that we consider. In our experiment, we measure the avalanche effect of STITCH-256 algorithm for s = 16 and 32
- Size δ of the induced input difference is 1. It means that we are mainly interested in a smallest value here to observe the property of Strict Avalanche Criterion (SAC) in STITCH-256

The changes or the effects from a single bit flip in the input are calculated and stored. Table 3 describes the breakdowns of the results obtained from the experiment conducted on three data categories.

It can be seen from Table 3 the 3000 sequences of 256 bit hash values of STITCH-256 produced the desired avalanche effect and also satisfy Strict Avalanche Criterion (SAC). Figure 6 shows the avalanche effect of a single flip-ping in input bit for low density data. It can be seen that the avalanche effect for 1000 sequences of 256 bit hash values for low density data is well scattered around 0.5. Figure 7 shows the total number of 512 bit output sequences with respect to the avalanche effect. From Fig. 7, it can be seen that the avalanche effects of all the output sequences lie between 0.495-0.504. This shows that on average, half of the output bits were changed with a single input bit flipping.

The similar effect applies to the high density data and random data, where their avalanche effect for the same length of sequences and output bits are well scattered around 0.5. This is depicted in Fig. 8-11. Figure 8 shows the total number of 512 bit output sequences of high density data with respect to its avalanche effect. From Fig. 9 it can be seen that the avalanche effects of all the

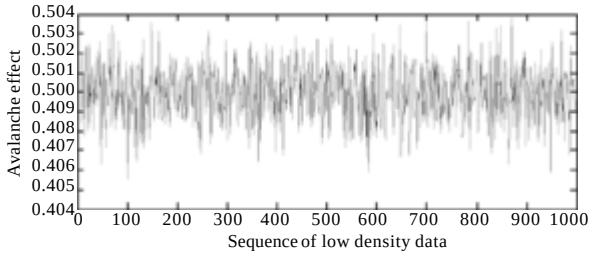

Fig. 6: Avalanche effect for 1000 sequences of 512 bit low density data

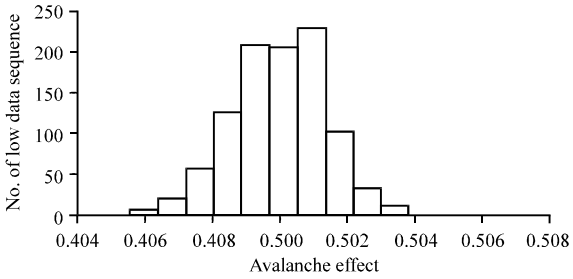

Fig. 7: Number of 256 bit output sequences vs. avalanche effect

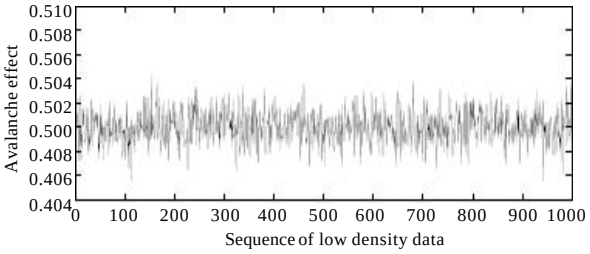

Fig. 8: Avalanche effect for 1000 sequences of 512 bit high density data

Table 3: Results from the avalanche experiments for three different data types

Data category	Sample size/ sequence length	Min avalanche	Max avalanche	Average avalanche
Low density	1000/256	0.4956	0.5042	0.5000
High density	1000/256	0.4956	0.5039	0.5000
Random	1000/256	0.4949	0.5046	0.5001

output sequences lie between 0.496-0.504. This shows that, on average, half of the output bits were changed with a single input bit flipping. From Fig. 11, it can be seen that the avalanche effects of all the output sequences lie between 0.495-0.505. This shows that on average, half of the output bits were changed with a single input bit flipping in random data.

From the results of our experiment, it was shown that the output of STITCH-256 algorithm

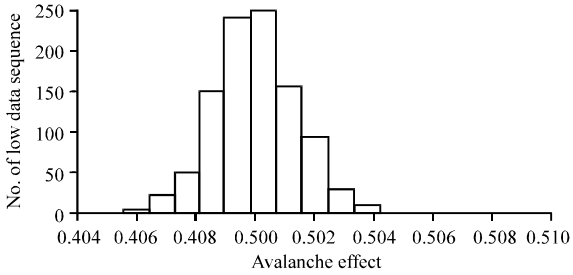

Fig. 9: Number of 256 bit output sequences vs avalanche effect

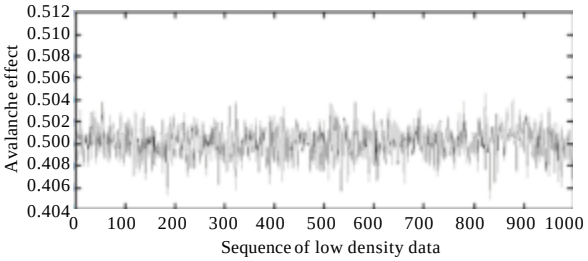

Fig. 10: Avalanche effect for 1000 sequences of 512 bit random data

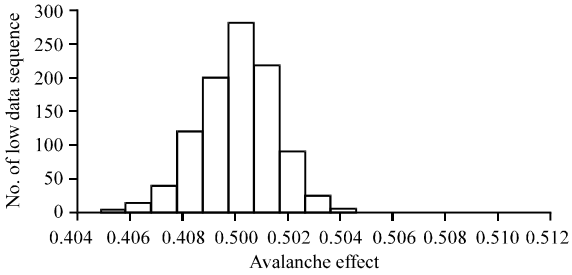

Fig. 11: Number of 256 bit output sequences vs avalanche effect

appears to provide Strict Avalanche Criterion for all three data categories. Next, we showed that stitching permutation in STITCH-256 increases the diffusion property of the algorithm. This can be seen from Fig. 12 that the avalanche factor for STITCH-256 if it runs without stitching permutation lies at 0.24-0.26. The main property in stitching permutation is it swaps the intermediate chaining variables from the left wing to the right wing and vice versa, in each iteration. It means that each register has its step operation changed every after a single iteration. This provides high bit propagation through a single iteration.

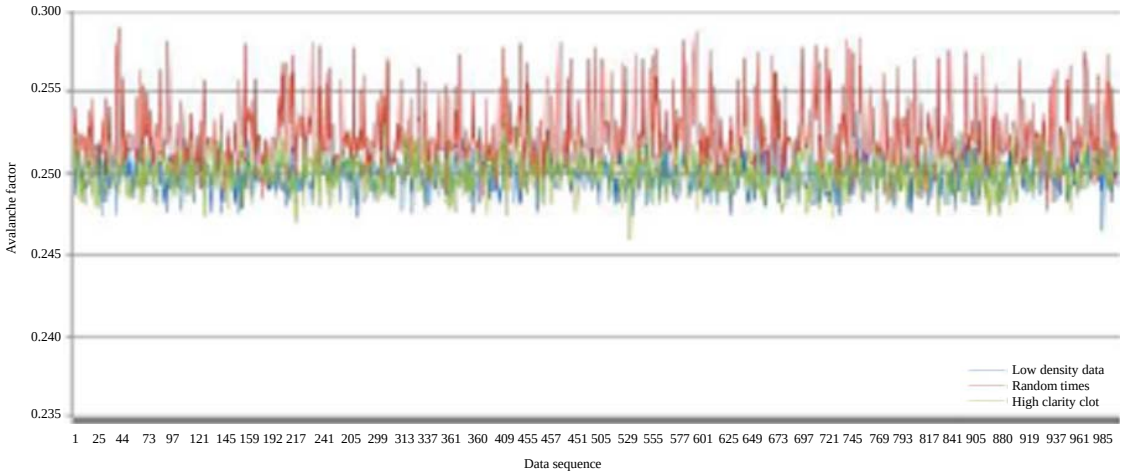

Fig. 12: The avalanche factor for STITCH-256 without stitching permutation

RANDOMNESS OF STITCH-256

For the outputs of the hash function to be indistinguishable from a random oracle, they have to be tested in terms of randomness tests. It means the outputs or the hash value of hash function has to be shown to be random. This evaluation of the randomness can be done by using statistical test. In our case, we consider using NIST statistical test suites that contains 15 tests that, under different data category, can be viewed as 48 statistical tests. The list of the tests are frequency test, block frequency test, runs test, long runs of ones test, rank test, discrete Fourier transform test, non-overlapping template matching test, overlapping template matching test, linear complexity test, universal statistical test, approximate entropy test, cumulative sums test, random excursion test, random excursion variant test and serial test.

Experiments: some of the tests, for example overlapping template matching test, universal statistical test, random excursion test, linear complexity test and random excursions variant test, require the sequence length to be more than 1000000 bits and Rank Test requires the sequence to be minimum 38,912 bits. Due to the hardware constraint, we only provide 768000 bits (3000 sequences of 256 bit) for the entire experiments, i.e., these six tests were not performed. Thus the appropriate NIST tests are only nine which are frequency test, block fre-quency test, runs test, long runs test, cumulative sum test, approximate entropy test, serial test, discrete Fourier transform test and non-overlapping template matching test. Table 4 shows the breakdown of the 9 statistical tests applied during the experiments.

Table 4: Test ID for 9 NIST statistical tests

Statistical test	Test ID
Approximate entropy	1
Block frequency	2
Cumulative sum	3
Frequency	4
Long runs	5
Runs	6
Serial	7
Discrete Fourier transform	8
Non-overlapping template matching	9

We use the same data categories as that used in avalanche test. A full round of STITCH-256 is run and repeated many times. The sequences of 256 bit hash values are concatenated to produce large data input to the statistical test. We follow the same procedure as described by Soto and Bassham (2000). The statistical tests are applied only to the compression functions of STITCH-256. It means that the padding procedure is omitted. The output from the experiments is 9 results for each data category. Statistical tests were performed using the strategies as follows.

Finally, we show the avalanche effect of 32 rounds of STITCH-256. This is depicted in Fig. 13. It is shown in the figure that the avalanche factors for 32 rounds of STITCH-256 have not much different from that of 16 rounds of STITCH-256. As 32 rounds incur more processing and time, STITCH-256 employs only 16 rounds with almost similar avalanche effect with 32 rounds. This provides STITCH-256 with comparable efficiency and security with other hash functions:

Step 1: For each data category, all input parameters such the sequence length, sample size and significance level were fixed. They are defined as 256, 256000 and 0.01, respectively. For each binary sequence and each statistical test, a p-value was reported

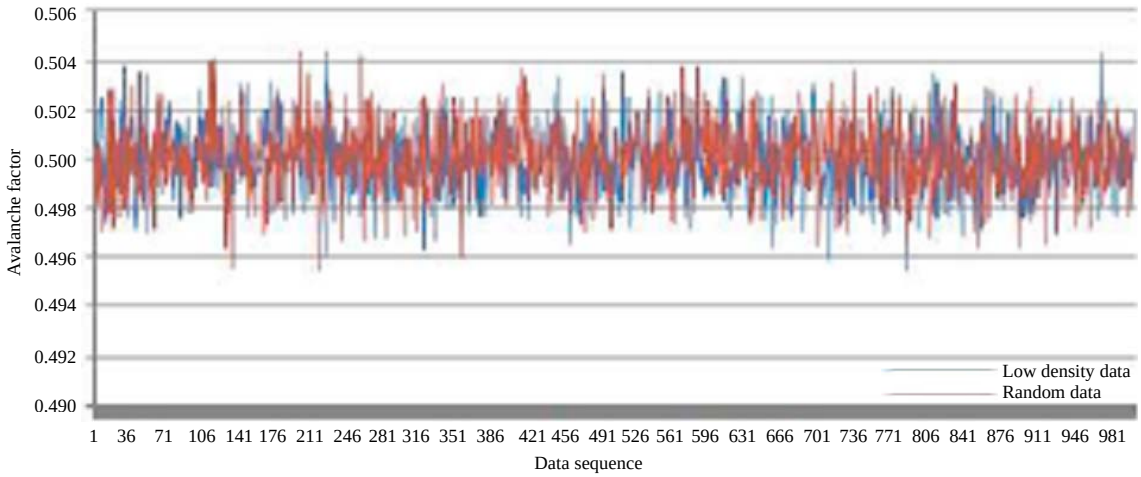

Fig. 13: The avalanche factor for 32 rounds of STITCH-256

Step 2: The status of success or failure was made based on whether or not for each p-value exceeded or fell below the pre-selected significance level

Step 3: Two evaluations were made for each statistical test and each sample. Firstly, we calculated the proportion of binary sequences in a sample that passed the statistical test. The p-value for this proportion is equal to the probability of observing a value that is equal to or greater than the calculated proportion. Secondly, we calculated an additional p-value based on a χ^2 test (with nine degrees of freedom) applied to the p-values in the entire sample to guarantee uniformity

Step 4: An assessment was made for both steps done in step 3. A sample was considered to pass a statistical test if it satisfied both the proportion and uniformity assessments. If one of the two p-values in step 2 fell below 0.0001, then the sample is labeled as suspect. In this case, then additional samples needed to be evaluated. Otherwise, the sample is said to satisfy the criterion for being random from the perspective of a specific statistical test (Rukhin *et al*., 2001)

Empirical results and analysis: Given the empirical results for a particular test, we computed the proportion of sequences that pass. In our case, we tested 1000 binary sequences for each data category, i.e., m = 1000. The significance level, σ, for our experiment is set at σ = 0.01. We use the confidence interval formula defined by Soto and Bassham (2000) as:

$$p \pm 3\sqrt{\frac{p(1-p)}{m}}$$

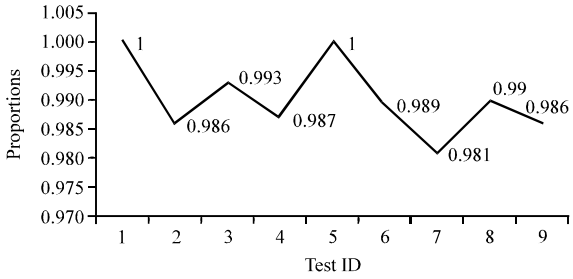

Fig. 14: Proportion of low density sequences undergoing 9 statistical tests

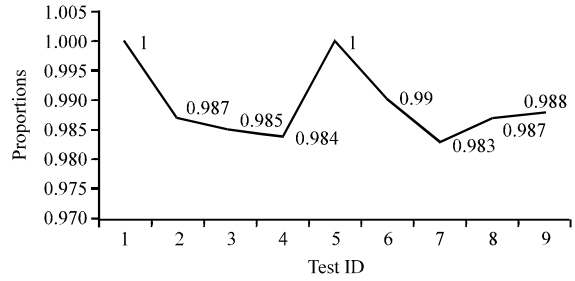

Fig. 15: Proportion of high density sequences undergoing 9 statistical tests

where p = 1-σ. The data is classified as non-random if the calculated proportion is not in this interval. In our case, the confidence interval is 0.99±0.0094392. This means that the proportion should be above 0.980560. The calculated confidence interval is an approximation to the binomial distribution which is reasonably for m = 1000. Figure 14-16 show the proportion of sequences that undergo nine statistical tests for each of data category.

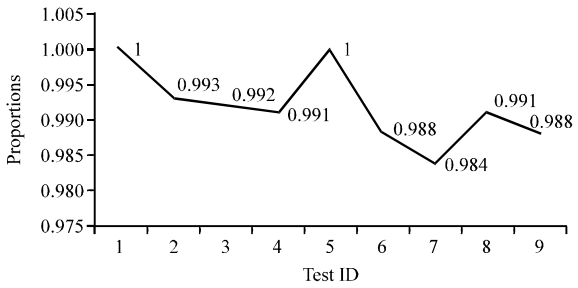

Fig. 16: Proportion of random sequences undergoing 9 statistical tests

From the results of the experiments, it showed that the outputs from the full round of STITCH-256 algorithm appeared to be random for all nine statistical tests.

CONCLUSION

In this study, the avalanche effect of STITCH-256 algorithm was measured by testing 3000 sequences of 256 bit hash values. The empirical results from the avalanche test showed that STITCH-256 exhibits Strict Avalanche Criterion (SAC) which is a desired property in any cryptographic algorithm. The contributing factors to this property are the implementation of different message orderings in each line in the compression function of STITCH-256 and stitching permutation which maximizes the diffusion property of STITCH-256. Finally, 3000 sequences of 256 bit hash values of different data categories were tested with NIST statistical test suites. The tests spanned many well-known cryptographic properties which have to be satisfied by any cryptographic algorithm. One of the properties included the absence of any detectable correlation between input/output pairs and also the absence of any detectable bias resulting from a single bit flip in input. Nine statistical tests were identified from the test suite that are relevant with 256 bit output from any cryptographic algorithm and run the test for three different data categories separately. The proportion of p-values of the data sequences were then computed and compared with the confidence interval. The empirical results showed that the outputs from three different data categories in full round of STITCH-256 passed all the nine tests, i.e., appeared to be random. From the avalanche and statistical tests conducted, we showed that the claim made with regards to high diffusion and random output of STITCH-256 is correct.

REFERENCES

Biham, E., R. Chen, A. Joux, P. Carribault, C. Lemuet and W. Jalby, 2005. Collisions of SHA-0 and Reduced SHA-1. In: Advances in Cryptology-EUROCRYPT 2005, Cramer, R. (Ed.). Vol. 3494, Springer-Verlag, Germany, ISBN: 978-3-540-25910-7, pp: 36-57.

Damgard, I.B., 1990. A Design Principle for Hash Functions. In: Advances in Cryptology, Brassard, G. (Ed.). Springer-Verlag, New York, pp: 416-427.

FIPS 180, 1993. Secure Hash Standard (SHS). National Institute of Standards and Technology, May 1993. US Department of Commerce.

Jamil, N., R. Mahmod, M.R. Z'aba, N.I. Udzir and Z.A. Zukarnain, 2012. STITCH-256: A new dedicated cryptographic hash function. J. Appl. Sci., 15: 1526-1536.

Merkle, R.C., 1989. One Way Hash Functions and DES. In: Advances in Cryptology, Brassard, G., (Ed.). Vol. 435. Springer-Verlag, USA., pp: 428-446.

NIST, 1995. FIPS 180-1. Secure Hash Standard (SHS). National Institute of Standards and Technology, USA.

Rivest, R.L., 1992a. The MD4 message-digest algorithm. Request for Comments (RFC) 1320, Internet Activities Board, Internet Privacy Task Force, April 1992.

Rivest, R.L., 1992b. The MD5 message-digest algorithm. Request for Comments (RFC) 1321, Internet Activities Board, Internet Privacy Task Force, April 1992.

Rukhin, A., J. Soto, J. Nechvatal, M. Smid and E. Barker *et al*., 2001. A statistical test suite for random and pseudorandom number generators for cryptographic applications. http://csrc.nist.gov/publications/nistpubs/800-22-rev1/SP800-22rev1.pdf

Soto, J. and L. Bassham, 2000. Randomness testing of the advanced encryption standard finalist candidates. National Institute of Standards and Technology, NIST IR 6483, pp: 1-14. http://csrc.nist.gov/publications/nistir/ir6483.pdf

Van Rompay, B., A. Biryukov, B. Preneel and J. Vandewalle, 2003. Cryptanalysis of 3-pass HAVAL. Proceedings of the 9th International Conference on the Theory and Application of Cryptology and Information Security, Advances in Cryptology, November 30-December 4, 2003, Taipei, Taiwan, pp: 228-245.

Wang, X., X. Lai, D. Feng and H. Yu, 2004. Collisions for hash functions MD4, MD5, HAVAL-128 and RIPEMD. http://eprint.iacr.org/2004/199.pdf

Wang, X. and H. Yu, 2005. How to break MD5 and other hash functions. Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, May 22-26, 2005, Aarhus, Denmark, pp: 19-35.

Wang, X., H. Yu and Y.L. Yin, 2005a. Efficient collision search attacks on SHA-0. Adv. Cryptol., 3621: 1-16.

Wang, X., X. Lai, D. Feng, H. Chen and X. Yu, 2005b. Cryptanalysis of the hash functions MD4 and RIPEMD. Proceedings of the 24th Annual International Conference on the Theory and Applications of Cryptographic Techniques, May 22-26, 2005, Aarhus, Denmark, pp: 1-18.

Wang, X., Y.L. Yin and H. Yu, 2005c. Finding Collisions in the Full SHA-1. In: Advances in Cryptology-CRYPTO 2005, Shoup, V. (Ed.). Springer Verlag, New York, USA., ISBN: 978-3-540-28114-6, pp: 17-36.

Wolfram, S., 1984. Universality and complexity in cellular automata. Physica, 10: 1-35.

Yu, H., X. Wang, A. Yun and S. Park, 2006. Cryptanalysis of the Full Haval with 4 and 5 Passes. In: Fast Software Encryption, Robshaw, M. (Ed.). Springer, New York, ISBN: 9783540365976, pp: 89-110.

Zheng, Y., J. Pieprzyk and J. Seberry, 1993. HAVAL: A One-Way Hashing Algorithm With Variable Length Of Output. In: Advances in Cryptology Auscrypt 92, Seberry, J. and Y. Zheng (Eds.). Vol. 718, Springer Verlag, Germany, ISBN-13: 978-3540572206, pp: 83-104.