Journal of Applied Sciences

ISSN 1812-5654

science alert

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

Journal of Applied Sciences 12 (15): 1526-1536, 2012
ISSN 1812-5654 / DOI: 10.3923/jas.2012.1526.1536

STITCH-256: A Dedicated Cryptographic Hash Function

[1,2]Norziana Jamil, [2]Ramlan Mahmod, [3]Muhammad Reza Z'aba, [2]Nur Izura Udzir and [2]Zuriati Ahmad Zukarnain
[1]College of Information Technology, Universiti Tenaga Nasional, Selangor, Malaysia
[2]Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Selangor, Malaysia
[3]MIMOS Berhad, Technology Park Malaysia, Kuala Lumpur, Malaysia

Abstract: Hash functions are an important cryptographic primitive in a wide range of applications, such as message authentication and digital signatures. Among others, Message Digest (MD)-4/5, Secure Hashing Algorithm (SHA)-0/1/2 and RACE Integrity Primitives Evaluation Message Digest (RIPEMD) family are examples of widely used hash functions in cryptographic applications. However with the advances of cryptanalysis, MD-4/5, SHA-0/1 and RIPEMD were broken in 2004-2005 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. This study describes a 256-bit hash function named as STITCH-256, to overcome problems found in MD- and SHA-family hash functions. We analyze all the components of STITCH-256 and the security analysis of STITCH-256 shows that it is not feasible to construct a collision path for STITCH-256 with a high probability of success.

Key words: Hash function, cryptographic boolean functions, cellular automata rules, collision resistance, differential attack

INTRODUCTION

Cryptographic hash functions are an important primitive deployed in many applications and protocols for secure communication. Their main objective is to provide data (or message) integrity. The computation of a hash function over a particular message yields a message digest, sometimes referred to as the hash value or hash code. A message is said to be of no integrity if the hash value of the message after transmission does not match the hash value of the message before transmission. Thus, the integrity of the original message can be verified from the hash value of the message itself. In principle, a hash function is a function H that accepts a message of arbitrary length * and produces a message of fixed length n via a compression function f, H: $[0, 1]^{*} \rightarrow [0, 1]^{n}$. Cryptographic hash functions are not the same as the common hash function used in a look-up table for passwords. In principle, cryptographic hash functions must fulfill three basic cryptographic properties: preimage resistance, second preimage resistance and collision resistance. These properties are explained as follows.

- **Preimage resistance:** Given any hash value y, it is infeasible to find its corresponding message m. This property is also known as one-wayness. The hash function is considered to be preimage resistant if the complexity of finding the corresponding message is equal to or greater than 2^{n}
- **Second preimage resistance:** Given a message m and its corresponding hash value H(m), it is infeasible to find another message m', where $m \neq m'$, such that $H(m) = H(m')$. Suppose that second preimage resistance also implies preimage resistance; therefore, the complexity of finding a message m' through a brute force attack is also 2^{n}
- **Collision resistance:** Given a hash value H(m), it is infeasible to find any two random messages m and m', where $m \neq m'$, such that $H(m) = H(m')$. For a collision-resistant hash function, the fastest way to find m and m' is with a birthday attack which has a complexity of $2^{n/2}$

In the past, most cryptographic applications used MD-family hash functions, such as MD4 (Rivest, 1992a) and MD5 (Rivest, 1992b), SHA-family hash functions, such as SHA-0 (NIST, 1993) and SHA-1 (NIST, 1995), RIPEMD (Bosselaers and Preneel, 1995) and HAVAL (Zheng *et al*., 1993). These hash functions survived for more than a decade until, in 2004. A powerful techniques based on differential attack to find a pair of colliding messages was presented by Wang and Yu (2005) and

Corresponding Author: Norziana Jamil, College of Information Technology, Universiti Tenaga Nasional, Selangor, Malaysia
Tel: +60389212020 Fax: +0389212161

Wang *et al.* (2005a-c). However, RIPEMD-128/160 and SHA-2 (NIST, 2002) family hash functions are still secure and resistant against Wang *et al.*'s attacks.

RIPEMD-128/160 has different structures than that of MD and SHA family hash functions. The compression function consists of two parallel branches having the same step operations of MD5 and SHA algorithm. However, this structure results in its efficiency was degenerated almost half of them. This is described by and they introduced FORK-256 Hong *et al.* (2006) to overcome the problem in RIPEMD. FORK-256 has four parallel branches as compared to two in RIPEMD-128/160, but with less number of step operations in each branch. However, the memory requirement in FORK-256 is bigger than that of RIPEMD-320. Furthermore, Matusiewiez *et al.* (2006) and Matusiewiez *et al.* (2007) showed that a near-collision can be constructed for a complete version of FORK-256 with a complexity of 2^{125} hash computation. Using Matusiewiez's technique, Contini *et al.* (2007) showed that it can be extended to find full collisions for a complete version of FORK-256.

From the weaknesses found in MD-4/5, SHA-0/1, RIPEMD and FORK-256 hash functions, we proposed a dedicated cryptographic hash function, STITCH-256. It was designed with more careful observations in all of its components. We also carefully selected cryptographically strong Boolean functions from a set of one dimensional Cellular Automata (CA) rules and used them in STITCH-256 step operations. Recent research on cryptography based on CA can be found (Ayanzadeh *et al.*, 2009; Ayanzadeh *et al.*, 2012; Jaberi *et al.*, 2012). We analyzed the security of all of the components of STITCH-256 and compared with the other hash functions discussed earlier.

DESCRIPTIONS OF MD-4/5, SHA-0/1/2, RIPEMD AND FORK-256

Here, we briefly describe the widely used hash functions such as MD-4/5, SHA-0/1/2 and RIPEMD and recently introduced hash function, FORK-256. The notation used throughout this study is given in Table 1.

MD-4/5: MD-4 was born in 1990 and performed excellently on 32-bit software architecture. The design was novel and was expected to provide high security. However, it was less than a year after it's born, found that the security level of MD4 was not up to the expectation. collisions for reduced versions of the algorithm (Den Boer and Bosseleras, 1991; Dobbertin, 1996, 1998). MD-5 was born a year after MD4, introduced to be the improved version of MD4. Both MD-4 and MD-5 process 512-bit input message and produce 128-bit output message. MD-4 and MD-5 have 3 and 4 rounds for a single iteration, respectively. The step operation in MD4 is depicted in Fig. 1. The followings are the Boolean functions used in MD-4:

Table 1: Notation

Notation	Description
$A \oplus B$	XOR operation of A and B
$A+B$	Addition of A and B modulo 2^{32}
M_t	32-bit input message word of tth block
W_t	32-bit expanded input message word of tth block
$ROTR/L^n(A)$	Bit rotation of A by n position/s to the right or left, respectively
$SHFR/L^n(A)$	Bit shift of A by n position/s to the right or left, respectively
N	Number of rounds of message expansion

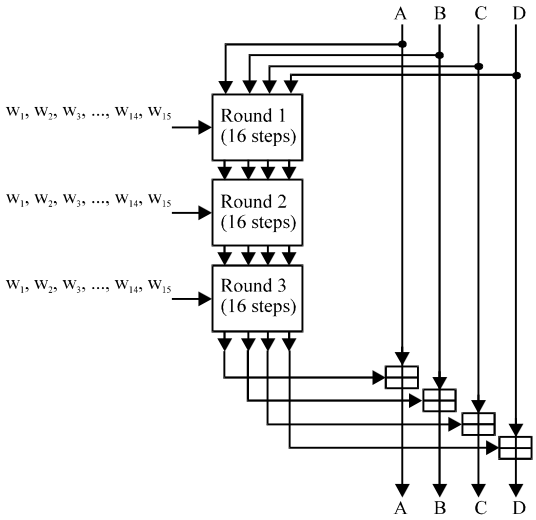

Fig. 1: The structure of MD4 hash function

$$F(X, Y, Z) = X \wedge Y \vee \sim X \wedge Z$$
$$G(X, Y, Z) = X \wedge Y \vee X \wedge Z \vee Y \wedge Z$$
$$H(X, Y, Z) = X \oplus Y \oplus Z$$

where, $\wedge$, $\vee$ and $\oplus$ are bitwise AND, OR and XOR operations, respectively. Functions F, G and H are used in different round. MD-5 is slightly different in that it has four Boolean functions used in different round. The step operation in MD-5 adds in a unique additive constant and the rotation constant values are different for the 64 rounds in the full algorithm. The details of MD-4/5 can be found in (Rivest, 1992a, b).

SHA-0/1/2: SHA family hash functions were designed by NSA and published by NIST as federal standard FIPS.

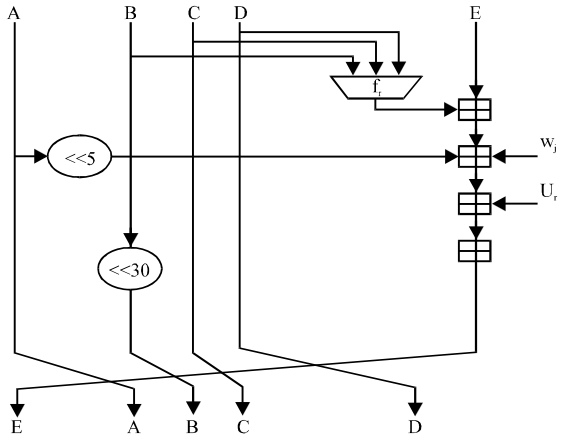

Fig. 2: The step operation of functions SHA-0 and SHA-1

The design principle of SHA-0 was not disclosed but it was similar to that of MD-4. SHA-1 is the improvement to SHA-0. SHA-2 family hash functions were introduced in 2002 offering a longer hash value to increase the security level of the algorithms against birthday attacks. The step operation for SHA-1 is depicted in Fig. 2. SHA family hash functions are interesting as they used recursive functions for their message expansion. The details of SHA-0/1/2 can be found in NIST (1993, 1995, 2002).

RIPEMD: The designs of RIPEMD family hash functions are based on MD-4, in which the only difference is RIPEMD has two parallel branches in its compression function. Both parallel branches have similar step operation of MD-4. The step operation for RIPEMD is depicted in Fig. 3 and the details of RIPEMD hash functions (Preneel *et al*., 1997).

FORK-256: FORK-256 maps 256 bits of state and 512 bits of message to 256 bits of hash value. The compression function of FORK-256 consists of four parallel branches as illustrated in Fig. 4. FORK-256 employs fewer operations as compared to SHA-256 to maintain the efficiency of the algorithm. Note that a major difference in FORK-256, as compared to the other hash functions, is several words impact more than one word in a step operation. The details of FORK-256 can be found by Hong *et al.* (2006) and the step operation of FORK-256 is illustrated in Fig. 5.

The cryptanalysis shows that these hash functions have several weaknesses. We outline the weaknesses as follows:

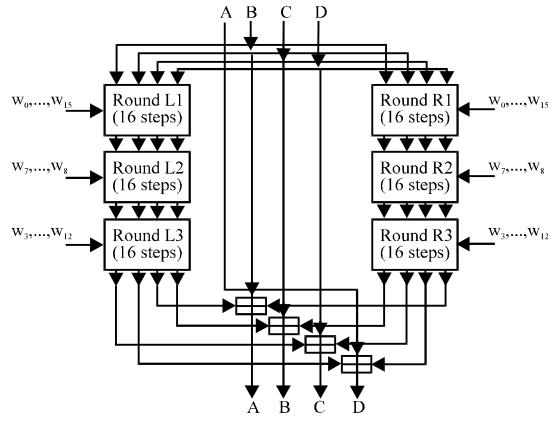

Fig. 3: The compression function of RIPEMD

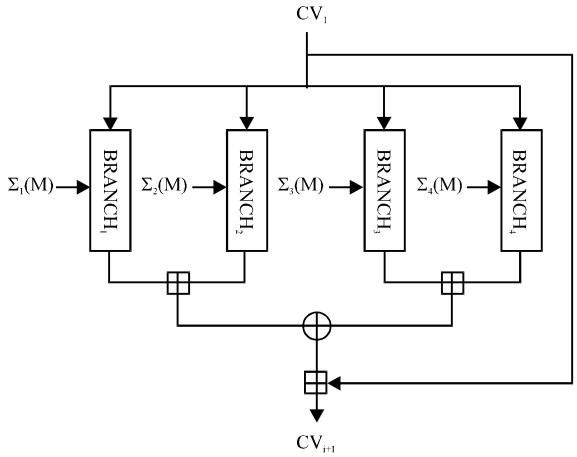

Fig. 4: The compression function of FORK-256 (Hong *et al*., 2006)

- Some of the message expansion procedures in these hash functions are linear functions which lead to poor diffusion
- Certain mathematical operations have unexpected security problems. This could probably due the weakness exposed by the Boolean functions used in the compression function
- Similar message expansion procedure (i.e., close distance for two message words) for parallel branches (for RIPEMD hash function) leads to similar patterns for differential characteristics
- Light step operations and similar constants are used in different round in step operations. This leads to poor bit propagation and differential characteristics can be constructed easily

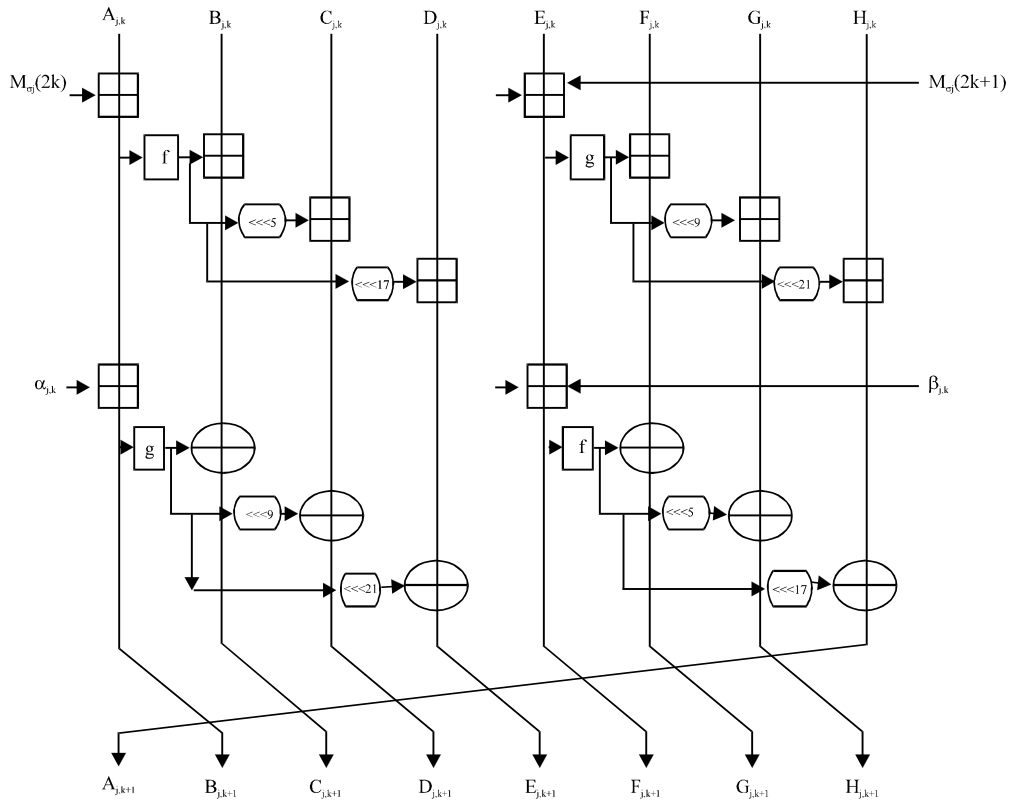

Fig. 5: The step operation of FORK-256 (Hong *et al*., 2006)

- As in the MD, SHA and RIPEMD family hash functions, the output from Boolean functions that takes three words only affects one word in every single round. This leads several input words to be controllable by the attacker to produce a particular output word. In other word, the output words of Boolean function are used to update only one chaining variable

Observing the weaknesses above, we propose a dedicated cryptographic hash function, STITCH-256, to overcome the problems raised in these hash functions.

DESCRIPTION OF STITCH-256

STITCH-256 is a cryptographic hash function that accepts a message of arbitrary length and produces one of fixed size. It has four main components, namely padding, message expansion, compression function and step operation. In the first component, the arbitrary input message is first padded by a single bit '1' next to the least significant bit of the message, followed by zero as many as possible until the length of the message is 448 modulo 512. At least one bit and at most 512 bits are appended to the message. This is then followed by a 64-bit unsigned big-endian representation of message length modulo 2^{64} to the message. This procedure ensures that the length of the padded message is a multiple of 512. Padding for STITCH-256 can be represented as:

$$M \leftarrow m \parallel 10000\ldots.00001\langle \text{message length} \rangle_{64}$$

This is also known as MD-strengthening (Merkle, 1989; Damgard, 1989) which allows the function to be resistant against fixed point and message expansion attacks.

Every message block M is then expanded into several 32-bit message words in a procedure called message expansion.

Message expansion of STITCH-256: To proceed to the iterated hash, each successive 512-bit message block M is expanded into thirty two 32-bit message words W_i according to the following formulas:

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

$$W_t = ROT^{11}(s_0(W_{i-16}, W_{i-15}, W_{i-14}, W_{i-13}) + s_1(W_{i-12}, W_{i-11}, W_{i-10}, W_{i-9}) \oplus SV_0) + ROT^{13}(s_0(W_{i-8}, W_{i-7}, W_{i-6}, W_{i-5}) + s_1(W_{i-4}, W_{i-3}, W_{i-2}, W_{i-1}) \oplus SV_1) \text{ for } 16 \le t \le 32$$

where, $\sigma_0(w, x, y, z)$ is $w \oplus x \oplus y \oplus z$ and $\sigma_1(w, x, y, z)$ is $w+x+y+z$.

We use two salt values to support message expansion in STITCH-256, namely SV_0 = 67452301 and SV_1 = 41083726.

The process of message expansion in STITCH-256 is illustrated in Fig. 6.

Note that the output from message expansion in STITCH-256 is thirty-two 32-bit message words W_i. We then re-arrange the ordering of the message words W_i according to Table 2 and we extract out eight different message orderings where each ordering consists of sixteen 32-bit message words. All the eight different message orderings are used in the compression function of STITCH-256.

Table 2: Boolean functions used in the STITCH-256 compression function

CA rule	Boolean representation
29	$p \otimes ((p \otimes (\sim r)) \vee q)$
39	$((p \otimes (\sim q)) \vee r) \otimes q$
27	$p \otimes ((p \otimes (\sim q)) \vee r)$
46	$(p \wedge q) \otimes (q \vee r)$
53	$(p \vee (q \otimes (\sim r))) \otimes q$
58	$(p \vee (q \otimes r)) \otimes q$
71	$((p \otimes (\sim r)) \vee q) \otimes r$
83	$(p \vee (q \otimes (\sim r))) \otimes r$

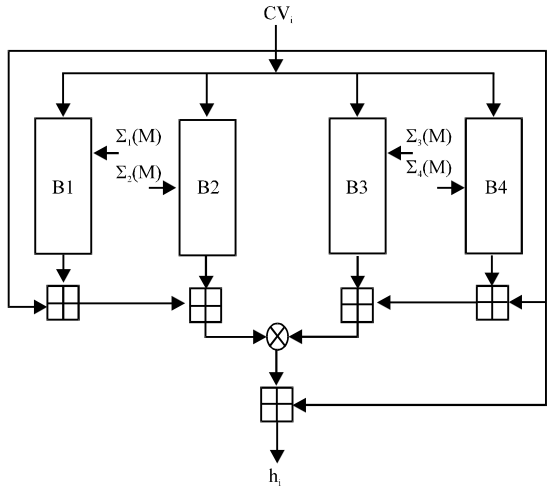

Fig. 6: Compression function of STITCH-256

Compression function of STITCH-256: The compression function of STITCH-256 is inspired by the design of parallel processing as this makes any cryptographic algorithm ready for future's architecture that would solve current speed problem and open cryptography to a whole new range of uses. RIPEMD family hash function is one of the examples that runs in parallel of two branches in its compression function. However, as the step operation in both of the branches are similar, this causes the efficiency of RIPEMD is degenerated almost half of them. STITCH-256 has four parallel branches, B1, B2, B3 and B4 in its compression function, as illustrated in Fig. 7.

Cv_i is a 256-bit chaining variable at ith iteration, is made up of eight 32-bit word registers A, B, C, D, E, F, G and H. The initial values or the chaining variable at i=0 of STITCH-256 are the same as that used in SHA-224:

A = C1059ED8, B = 367CD507
C = 3070DD17, D = F70E5939
E = FFC00B31, F = 68581511
G = 64F98FA7, H = BEFA4FA4

Each branch processes different message orderings through the step operations. This gives eight different message orderings for the whole STITCH-256 function, two for each branch. The intermediate chaining variable CV_i is updated to CV_{i+1} through the following process:

$$CV_{i+1} = [\, CV_i + B1(CV_i, \Sigma_1(M)) + B2(CV_i, \Sigma_2(M))\,] \oplus [CV_i + B4(CV_i, \Sigma_4(M)) + B3(CV_i, \Sigma_3(M))\,] + CV_i$$

where, $\Sigma_j(M)$ is the sixteen message words of different ordering in each branch j.

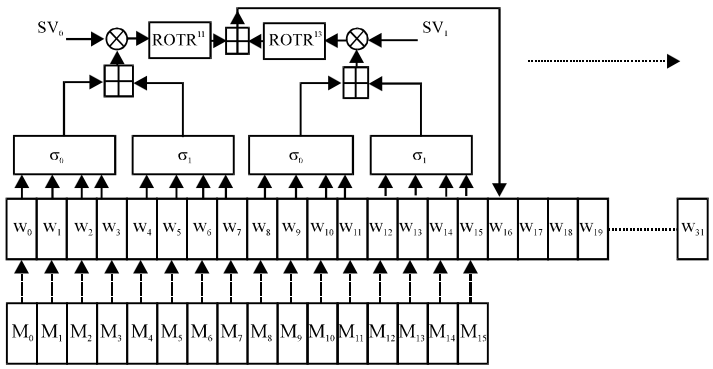

Fig. 7: Message expansion in STITCH-256

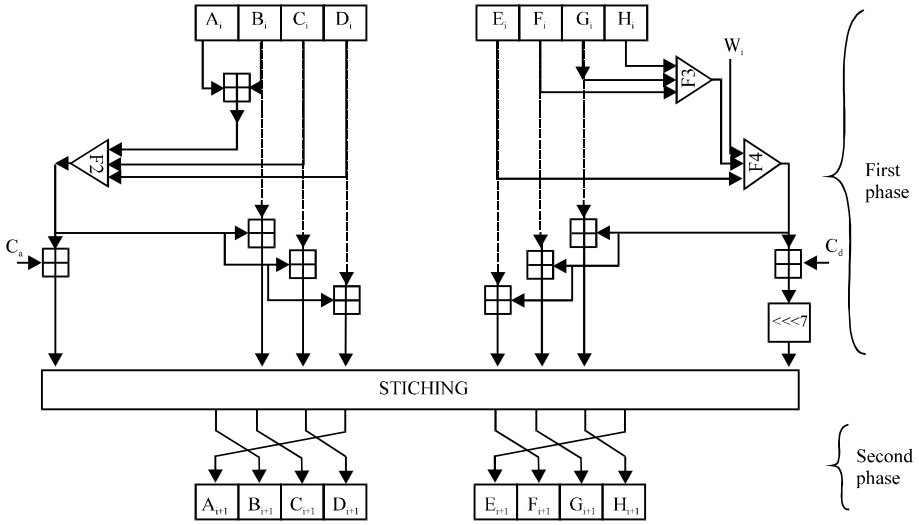

Fig. 8: A single step transformation in STITCH-256

Each of the branches has similar step operation.

Step operation in the compression function of STITCH-256: The compression function of STITCH-256 has two wings which we simply name the left and right wings. Each wing has different step operations, 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 chapter.

In the step operations of the right and left wings, two distinct Boolean functions are used in the right wing and one Boolean function 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 stitching permutation. For example, registers A_i, B_i, C_i and D_i are processed in the left wing during the first phase. The processed A_i, B_i, C_i and D_i are then rearranged using the stitching permutation and are processed in the right wing's first phase. This means the registers are processed in two 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.

Let the outputs for the first stage (before stitching) be as follows:

$$A'_i = D+f_i((A+B), C, D)$$
$$B'_i = C_{i,j}+f_i((A+B), C, D)$$
$$C'_i = B+f_i((A+B), C, D)$$
$$D'_i = C+f_i((A+B), C, D)$$
$$E'_i = ROT^7(C_{i,j}+f_2(E_i, W_i, f_3(F_i, G_i, H_i)))$$
$$F'_i = E_i+f_2(E_i, W_i, f_3(F_i, G_i, H_i))$$
$$G'_i = F_i+f_2(E_i, W_i, f_3(F_i, G_i, H_i))$$
$$H'_i = G_i+f_2(E_i, W_i, f_3(F_i, G_i, H_i))$$

A single step operation in each wing updates the registers by producing the following outputs:

$$A_{i+1} = f_1((E_i'+F_i'), G_i', H_i')+H_i'$$
$$B_{i+1} = c_a+f_1((E_i'+F_i'), G_i', H_i')$$
$$C_{i+1} = f_1((E_i'+F_i'), G_i', H_i')+F_i'$$
$$D_{i+1} = f_1((E_i'+F_i'), G_i', H_i')+G_i'$$
$$E_{i+1} = ROT^7(C_{i,j},+f_2(A_i', W_i, f_3(B_i', C_i', D_i')))$$
$$F_{i+1} = A_i'+f_2(A_i', W_i, f_3(B_i', C_i', D_i'))$$
$$G_{i+1} = B_i'+f_2(A_i', W_i, f_3(B_i', C_i', D_i'))$$
$$H_{i+1} = C_i'+f_2(A_i', W_i, f_3(B_i', C_i', D_i'))$$

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. The step operation of STITCH-256 is illustrated in Fig. 8.

The stitching permutation is designed to maximize the propagation of intermediate chaining variables, thus lower the probability to construct differential characteristics. The stitching permutation is illustrated in Fig. 9.

As described earlier, in the stitching permutation, the intermediate values Iv_i, $0 \le i \le 7$, from the right wing are permuted to the left wing and vice versa.

The Boolean functions used in the STITCHx-256 step transformation are shown in Table 2. These functions are selected from balanced Cellular Automata (CA) rules which exhibit strong cryptographic properties (Jamil *et al*., 2011). Each branch uses two Boolean functions (CA rules) and one diffusion function f_d. The orderings of Boolean functions used in the compression function of STITCH-256 are shown in Table 3.

The step operation in each branch uses two different constants, one for each wing, derived from the first digit of π. Table 3 shows the eight different constants used in the compression function of STITCH-256.

DESIGN STRATEGIES

Each of the components in STITCH-256 is designed to maximize the diffusion property of the function and to resist known generic attacks with efficient implementation in mind. To achieve these objectives, the design strategy for each of the component is described in the following chapter.

Message expansion in STITCH-256: The principle behind the design of the message expansion process in STITCH-256 is this: we want to involve the previous 16 message words to get a new value for the expanded message word. We choose the rotation that maximizes the diffusion in the STITCH-256 message expansion. The motivation for high diffusion in message expansion comes from the fact that the larger the minimum weight introduced in the message expansion, the more difficult it is to construct a collision path (Wang *et al*., 2005c).

Though the step operation in the right wing is not the reverse process of step operation in the left wing, we believe that the message differences that appear in the right wing can still be cancelled by a disturbance-correction strategy (Chabaud and Joux, 1998) with non-negligible probability. To avoid this from happening, each wing in a branch accepts a different message ordering to prevent an attacker from constructing a collision path with a high probability of success. This gives a total of eight message orderings in the compression function of STITCH-256 and each ordering consists of sixteen 32-bit message words. The message orderings are designed based on the following design principles:

- Each ordering has a different arrangement of message words, such that the sum of every element in a column is 60 and the sum of every element in a row is 120
- For a single step transformation, two different message orderings are used simultaneously in each branch

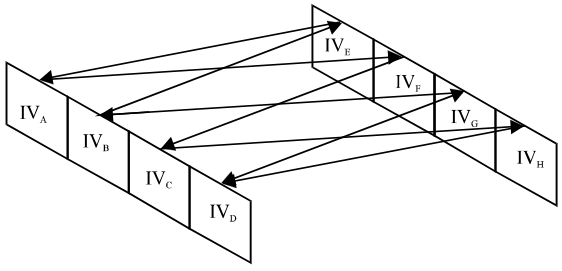

Fig. 9: Stitching permutation

Table 3: Boolean functions used in the compression function of STITCH-256

	Left wing	Right wing	
Branch	f_1	f_2	f_3
B1	f_d	CA rule 29	CA rule 39
B2	CA rule 27	CA rule 46	f_d
B3	CA rule 53	f_d	CA rule 58
B4	f_d	CA rule 71	CA rule 83

Table 4 shows the message orderings in STITCH-256. The number with 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 forth and so on.

With this arrangement, we confuse the attacker who aims at constructing a differential characteristic with high probability (Table 5).

Compression function of STITCH-256: The compression function of STITCH-256 is designed to have parallel branches with fewer operations and rounds to cater both security and speed. The choice of four parallel branches comes from the result of our experiment that shows less than four parallel branches having similar

Table 4: Message orderings (MO) for four branches

Branch	MO	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'

Table 5: Constants used in each branch

Branch	Wing	Constants
B1	Left	243F6A8885A308D3
	Right	082EFA98EC4E6C89
B2	Left	452821E638D01377
	Right	3F84D5B5B5470917
B3	Left	9216D5D98979FB1B
	Right	B8E1AFED6A267E96
B4	Left	BA7C9045F12C7F99
	Right	636920D871574E69

step operations such as that in four branches do not maximize the function's diffusion.

Step operation: The strategies for step operation are as follows:

- The output from a single Boolean function is distributed to all of the registers. This is to maximize the bit propagation of intermediate chaining variables
- Double permutations are introduced to maximize the bit propagation in a single round
- Only cryptographically strong Boolean functions are used in the step operation to confuse the attacker who aims at constructing the relationship between in the input and the output bits. One diffusion function f_d is introduced in each branch, $f_d = x \oplus y \oplus z$
- Diffusion is also maximized by a proper selection of bit rotations and the arrangement of addition modulo 2^{32}
- Different constants are used in every branch to randomize the pattern of message differences

SECURITY ANALYSIS OF STITCH-256

Here, we analyze all the components in STITCH-256 to show that STITCH-256 is collision resistant against known generic attacks.

Message expansion: In the message expansion of STITCH-256, every 16 message words are taken into account to form the (i-16)th message word. This maximizes bit propagation in the message expansion of STITCH-256. The bit rotations in this message expansion method are carefully selected to increase the diffusion of the whole message expansion. The strength of the message expansion can be measured by the effect of a single bit difference in a factor of $2^{-2.5}$ (Jutla and Patthak, 2009). It means that each weight in the expanded message words lowers the probability of successful collision characteristics by, on average, a factor of $2^{-2.5}$. In the message expansion of STITCH-256, the effect from a single bit difference has caused an average of 215 bits in the expanded message words to change. Furthermore, the lowest minimum weight found by a single bit difference in a full 32 rounds of message expansion in STITCH-256 is 171. This gives a probability of $2^{-2.5\times171}$ for an attacker to successfully construct a collision characteristic in the message expansion of STITCHx-256. This tells that it is infeasible to construct a collision characteristic in the message expansion of STITCH-256 with a high probability.

The main objective of having message orderings in the compression function of STITCH-256 is to make the effort of an attacker to construct collision characteristics in the compression function of STITCH-256 becomes useless. This is because, in order for an attacker to construct one, he needs to construct similar characteristics for all the branches having different message orderings. This seems to be impossible because the probability to find a collision in one branch is already 2.2^{-128}, as one branch has two wings taking different message orderings.

Compression function: Assume that the attacker inserts a message difference $\Delta m = m'-m$, to the i-th branch and suppose the output difference Δ_i is produced. The attacker expects that a collision might occur if the following event is satisfied:

$$\underbrace{\left[(CV+\Delta 1)+\Delta 2\right]}_{\alpha}\oplus\underbrace{\left[(CV+\Delta 4)+\Delta 3\right]}_{\beta}+CV=0$$

Say the attacker simplify the operation by denoting (CV+Δ1) as α and (CV+Δ4+Δ3) as β. There are several strategies that the attacker can take to fulfill this event. For every strategy, we provide the arguments on how to defeat the strategies"

- The attacker constructs α+Δ2 = -(β+Δ3+CV) to have a collision to occur. He can also reduce the complexity by constructing Δ2 = -Δ3 and α = -(β+CV). However differential pattern of the message words Δ2 = -Δ3 is difficult to achieve because the output of each branch is random. Therefore the probability to construct the differential pattern to occur with high probability is close to 2^{-128}.
- The attacker constructs two distinct differential characteristics and expects that α = - Δ2 and β = -Δ3. However, this is also difficult to construct since both α and β contain random output differential Δ in branch 1 and 4. Message reordering and bit propagation in a stitching way increase the probability for an attacker to construct differential characteristics with low probability.

Step operation: The step operation of STITCH-256 is analyzed from the perspective of differential attack since this is the most successful attacks to break the hash functions so far. In other words, we analyze only the property of collision resistance of STITCH-256.

To break a hash function using a differential attack, an attacker first aims to construct a collision path in each nonlinear component satisfying certain conditions. The whole collision path can be constructed when certain conditions hold for the message difference, intermediate values difference and output difference. We want to show that the probability of constructing a collision path in the step transformation of STITCH-256 is very small and the conditions to be fulfilled are large which in turn makes the attacker's efforts useless.

To show this, we first linearized the step transformation of STITCH-256. For this purpose, we replaced all the Boolean functions with addition modulo 2^{32} and rotation as the identity function. We also omitted the *stitching* permutation. We found a set of conditions that need to be fulfilled by applying the correction-disturbance strategy and then constructed a collision path for a linearized version of STITCH-256 by determining the number of steps where the difference between intermediate values is zero. This is also referred to as inner collision.

To do this, we introduced a disturbance in bit i = 1 of message words W_i, where $0 \leq i \leq 31$. The pattern of differences that we obtained for the left wing is listed in Table 6.

From Table 6, we can see that the difference becomes bigger as s increases. The condition for the left wing is complex and the disturbance cannot be corrected because the difference grows exponentially with the increasing number of steps. We introduced the same disturbance to the right wing and present the pattern of differences in Table 7.

Table 6: Correcting a single disturbance Δ_i introduced in step s in a linearized variant of STITCH-256 (left wing step transformation)

Steps	ΔA_s	ΔB_s	ΔC_s	ΔD_s	Δw_s
i	0	0	0	0	Δ
i+1	Δ	Δ	Δ	Δ	-26Δ
i+2	-6Δ	-Δ	-2Δ	0	61Δ
i+3	16Δ	7Δ	7Δ	0	-203Δ
i+4	-53Δ	17Δ	17Δ	0	-354Δ

Table 7: Correcting a single disturbance Δ_i introduced in step s in a linearized variant of STITCH-256 (right wing step transformation)

Steps	ΔA_s	ΔB_s	ΔC_s	ΔD_s	Δw_s
i	0	0	0	0	Δ
i+1	5Δ	4Δ	5Δ	5Δ	-24Δ
i+2	-6Δ	-6Δ	-6Δ	-7Δ	31Δ
i+3	6Δ	6Δ	6Δ	6Δ	-30Δ
i+4	-6Δ	-6Δ	-6Δ	-6Δ	30Δ
i+5	6Δ	6Δ	6Δ	6Δ	-30Δ

From Table 7, it is easy to see that the disturbance cannot be corrected because the pattern is simply repeating up to step i+5. In other words, there is no way to correct the disturbance unless the attacker can find a more intelligent way to cancel it beyond step i+5. For both the left and right wing step transformations, no appropriate conditions can be built.

DISCUSSION

We first discuss the advantage of each the strengthened components as follows:

Message expansion: As the formula to expand the message in STITCH-256 is inspired by a nonlinear recursive function in SHA-256, Table 8 shows the comparison between the message expansion of STITCH-256 and SHA-256 in terms of number of operations used in the algorithms. It shows that the total number of operations used in STITCH-256 is less than that of SHA-256. Having a better security with lower number of operations seemed to be a good advantage in the message expansion of STITCH-256.

Compression function: As the security analysis shows that it is infeasible to find colliding messages in the compression function of STITCH-256. It is observed that the mode of operation used is Davies-Meyer which provides one-wayness (preimage resistance) to the function.

Step operation: Preliminary security analysis done on the step transformations of STITCH-256 showed that the compression function of STITCH-256 is collision resistant from the perspective of differential attacks, particularly using the technique used to attack MD and SHA family hash functions. Table 9 shows the number of operations used in the step transformation of STITCH-256. We compare the number of operations with that of SHA-256 and found out that STITCH-256 consumes fewer operations than SHA-256, though STITCH-256 has four parallel branches. This is partly due to the fewer number of rounds in the compression function of STITCH-256.

Table 8: The number of operations used in the message expansion of STITCH-256 and SHA-256

Operations	STITCH-256	SHA-256
ADD MOD	6×16 = 96	3×48 = 144
XOR	5×16 = 80	4×48 = 192
AND	-	-
OR	-	-
NOT	-	-
ROT	2×16 = 32	4×48 = 192
SHIFT	-	2×48 = 96
TOTAL	208	624

Table 9: The number of operations used in the step transformation of STITCH-256 and SHA-256

	STITCH-256				
Operations	B1	B2	B3	B4	SHA-256
ADD MOD	9×16 = 144	9×16 = 144	9×16 = 144	9×16 = 144	7×64 = 448
XOR	7×16 = 112	6×16 = 96	7×16 = 112	7×16 = 112	4×64 = 256
AND	-	1×16 = 16	-	-	5×64 = 320
OR	2×16 = 32	2×16 = 32	2×16 = 32	2×16 = 32	3×64 = 192
NOT	2×16 = 32	1×16 = 16	1×16 = 16	2×16 = 32	1×64 = 64
ROT	1×16 = 16	1×16 = 16	1×16 = 16	1×16 = 16	6×64 = 384
SHIFT	-	-	-	-	-
TOTAL	336	336	320	336	
	1312				1664

CONCLUSION

In this study, we proposed a dedicated cryptographic hash function known as STITCH-256. The main components include message expansion, message ordering, compression function and step operation. As a result of these components, STITCH-256 became stronger from the perspective of security and was shown to be more collision resistant from the perspective of a differential attack, where no suitable condition could be constructed from the STITCH-256 step operation. Thus, it is very difficult for an attacker to construct collision characteristics during the step operation of STITCH-256. From the security analysis carried out in this study, we believe that STITCH-256 is a good candidate for cryptographic hash functions, especially in applications which collision resistance is the main sought property. However, since it is relatively new hash function, we invite the cryptographic community to further cryptanalyze STITCH-256.

ACKNOWLEDGMENT

This research work is supported by Ministry of Higher Education Malaysia under a Fundamental Research Grant Scheme Phase II 2010/11.

REFERENCES

Ayanzadeh, R., A.S.Z. Mousavi and E. Shahamatnia, 2012. Fuzzy cellular automata based random numbers generation. Trends Applied Sci. Res., 7: 96-102.

Ayanzadeh, R., K. Hassani, Y. Moghaddas, H. Gheiby and S. Setayeshi, 2009. Innovative approach to generate uniform random numbers based on a novel cellular automata. J. Applied Sci., 9: 4071-4075.

Bosselaers, A. and B. Preneel, 1995. Integrity Primitives for Secure Information Systems: Final Report of Race Integrity Primitives Evaluation Ripe-Race 1040. 1st Edn., Vol. 1007, Springer-Verlag, Germany, ISBN-13: 978-3540606406, Pages: 239.

Chabaud, F. and A. Joux, 1998. Differential Collisions in SHA-0. In: Advances in Cryptology-CRYPTO 98, Krawczyk, H. (Ed.). Vol. 1462, Springer-Verlag, USA., ISBN: 9783540648925, pp: 56-71.

Contini, S., K. Matusiewicz and J. Pieprzyk, 2007. Extending FORK-256 attack to the full hash function. Proceedings of the 9th International Conference on Information and Communications Security, December 12-15, 2007, Zhengzhou, China, pp: 296-305.

Damgard, I.B., 1989. A Design Principle for Hash Functions. In: Advances in Cryptology-CRYPTO 89, Brassard, G. (Ed.). Vol. 435, Springer-Verlag, USA., pp: 416-427.

Den Boer, B. and A. Bosselaers, 1992. An attack on the last two rounds of MD4. Adv. Cryptol., 576: 194-203.

Dobbertin, H., 1996. Cryptanalysis of MD4. In: Fast Software Encryption, Gollmann, D. (Ed.). Vol. 1039, Springer-Verlag, UK., ISBN: ISBN: 9783540608653, pp: 53-69.

Dobbertin, H., 1998. Cryptanalysis of MD4. J. Cryptol., 11: 253-271.

Hong, D., D. Chang, J. Sung, S. Lee and S. Hong *et al*., 2006. A New Dedicated 256-Bit Hash Function: FORK-256. In: Fast Software Encryption, Robshaw, M.J.B. and IACR (Eds.). Springer-Verlag, Austria, ISBN: 9783540365976, pp: 195.

Jaberi, A., R. Ayanzadeh and A.S.Z. Mousavi, 2012. Two-layer cellular automata based cryptography. Trends Applied Sci. Res., 7: 68-77.

Jamil, N., R. Mahmood, M.R. Z'aba, Z.A. Zukarnaen and N.I. Udzir, 2011. On Observation of Cryptographic Properties of 256 One-Dimensional Cellular Automata Rules. In: Informatics Engineering and Information Science: Proceedings of the International Conference on Informatics Engineering and Information Science, Abd-Manaf, A., A. Zeki, M. Zamani, S. Chuprat and E. El-Qawasmeh (Eds.). Volume 251, Springer-Verlag, Malaysia, pp: 409-420.

Jutla, C.S. and A.C. Patthak, 2009. Provably good codes for hash function design. IEEE Trans. Info. Theory, 55: 33-45.

Matusiewiez, K., S. Contini and J. Pieprzyk, 2006. Weaknesses of the FORK-256 compression function. IACR Cryptology e-print Archive 2006, Report 2006/317.

Matusiewiez, K., T. Peyrin, O. Billet, S. Contini and J. Pieprzyk, 2007. Cryptanalysis of FORK-256. Proceedings of the Fast Software Encryption, March 26-28, 2007, Springer, Luxumbourg.

Merkle, R.C., 1989. One Way Hash Functions and DES. In: Advances in Cryptology-CRYPTO 89, Brassard, G., (Ed.). Vol. 435, Springer-Verlag, USA., pp: 428-446.

NIST, 1993. FIPS 180. Secure Hash Standard (SHS). National Institute of Standards and Technology, USA.

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

NIST, 2002. FIPS 180-2. Secure Hash Standard (SHS). National Institute of Standards and Technology, USA.

Preneel, B., A. Bosselaers and H. Dobbertin, 1997. RIPEMD-160: A strenghtened version of RIPEMD. LNCS, 1039: 71-82.

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.

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: 551.

Wang, X., Y.L. Yin and H. Yu, 2005c. Finding collisions in the full SHA-1. Proceedings of the 25th Annual International Cryptology Conference, August 14-18, 2005, Santa Barbara, CA., USA., pp: 17-36.

Zheng, Y., J. Pieprzyk and J. Seberry, 1993. HAVAL-a one-way hashing algorithm with variable length of output. Proceedings of the Workshop on the Theory and Application of Cryptographic Techniques Gold Coast, December 13-16, 1992, Queensland, Australia, pp: 81-104.