STITCH-256: A New Dedicated Cryptographic Hash Function

Norziana Jamil, Ramlan Mahmood, Muhammad Reza Z'aba, Nur Izura Udzir, Zuriati Ahmad Zukarnaen.

Abstract

Recent progress in cryptanalysis on cryptographic hash functions has shown that the most of the hash functions based on the design principles of MD4 are susceptible to differential attack. This paper describes a new 256-bit hash function which is based on parallel branches having a stronger compression function. It is designed to have higher security than that of MD family and its variant. The performance of the new hash functions are evaluated and compared with SHA-256 and FORK-256. It is shown that STITCH-256 exhibit the desired cryptographic properties and comparable with SHA-256 and FORK-256 in its compression function.

1. Introduction

MD4 [1] is the first cryptographic hash function which made use of the structure of 32-bit processors. It employs serial operations and treatment of little-endian architecture in its compression function. Just a year after its publication in 1990, an attack on the last 32 out of 48 steps has been presented [2]. MD5 [3] turns out as a strengthen version of MD4 with additional round using the same structure but have a longer hash value. This design principle has long survived against any generic attacks when new techniques in cryptanalysis have treated the structure of MD in 1993 by Den Boer and Bosselaers [4]. They showed a way to find two different values of the IV and a common message *M* such that *MD5* (IV, *M*) = *MD5* (IV', *M*). However the attempt did not pose any threat to the usual applications of MD5 because one cannot control the inputs of chaining variables. The effort to find weaknesses in the structure of hash functions of MD-like design continued in 1996 when H. Dobbertin [4] found collisions for MD4 [5] and the last two out of three rounds of RIPEMD [6]. The attack was quite surprising since RIPEMD has a different structure than that of MD. The techniques used by H. Dobbertin on RIPEMD has been also used to produce collisions for MD4 and show that the two first rounds of MD4 are not one way [7]. He also chowed collisions for the compression function of MD5 with a chosen IV [8, 9]. These techniques have been improved by Wang et. al [10, 11, 12] to successfully find collisions for MD5, SHA-0 [13] and RIPEMD first proposal [14]. However RIPEMD-128/160 are the algorithms which are still immune against their attacks since there are no attacks on them are found so far.

RIPEMD-128/160 has different structure than that of MD and SHA family. The compression functions are within two parallel branches which makes the attack harder since two branches have to be taken into account simultaneously. Both branches of RIPEMD-128/160 [16] need

almost same operation of MD5 and SHA algorithm resulting in its efficiency was degenerated almost half of them. This is described by Hong et. al [15] in which they overcome this disadvantage of RIPEMD-128/160 by introducing FORK-256bit hash function having four branches. They also manage to reduce the number of operations for step functions of each line. However the speed performance of FORK-256 is slower than that of SHA-256 and memory requirement are bigger than that of RIPEMD-320. Furthemore, Matusiewiez et.al [17, 18] also showed that their technique can be used to yield a near-collision for a complete version of FORK-256 with a complexity of 2^{125} hash computation. FORK-256 shows a pretty good design since there is no successful attack found so far on its complete version. However, Matusiewiez's technique is claimed to be extended to find a collision against a complete version of FORK-256 [19]. This has motivated us to identify the weakness in FORK-256 and therefore STITCH-256 is designed with more careful step operations with good speed performance and security requirement in mind.

In this paper, we briefly describe FORK-256 hash function in Section 2, followed by a new proposed cryptographic hash function called STITCH-256 in Section 3, its security analysis is presented in Section 4 and we conclude the paper in Section 5.

2. A brief description of cryptographic hash function FORK-256

FORK-256 is a dedicated hash function that maps 256 bits of state and 512 bits of message to 256 bits of hash value. It is proposed by Hong et. al [9] and is based on the classical Merkle-Damgard iterative structure. The compression function of FORK-256 consists of four parallel branches as illustrated in figure 1 in which each branch processes set of message words in different order.

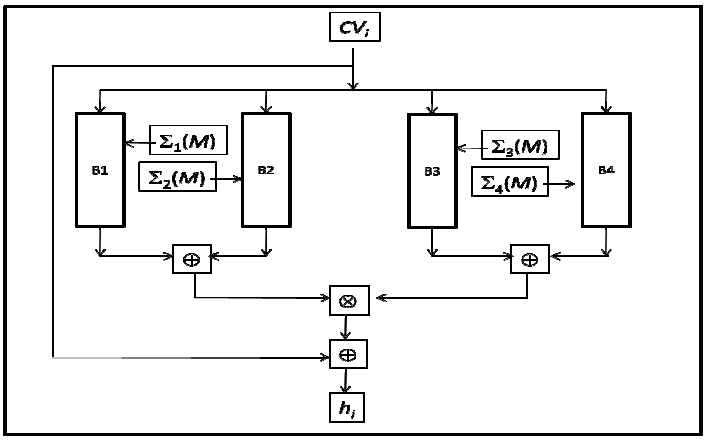

Fig. 1 The high level of compression function of FORK-256

Each branch function B1,…, B4 consists of eight steps where in each step $k = 1, \ldots, 8$ the branch function updates the eight chaining variables using step transformation as illustrated in figure 2 [9].

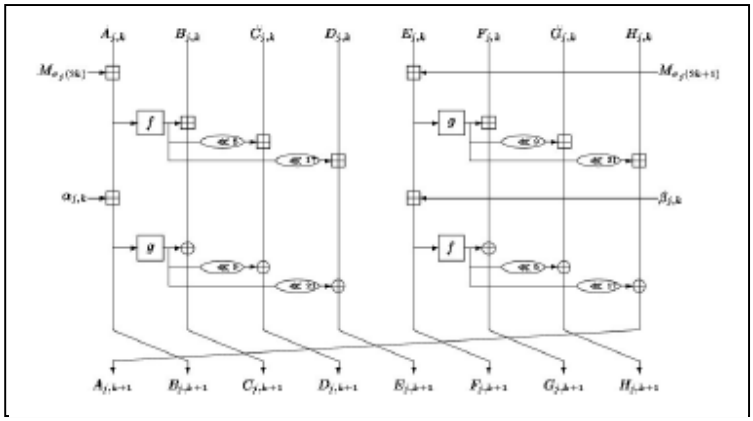

Fig. 2 Step transformation of FORK-256

Functions *f* and *g* map 32-bit words to 32-bit words are defined as

$f(x) = x + (ROL^{7}(x) \oplus ROL^{22}(x)), \quad g(x) = x + (ROL^{13}(x) \oplus ROL^{27}(x)).$

A more detail description can be found in [9].

It is reported in both [10] and [11] that a number of weaknesses of FORK-256 has been discovered. This includes the unexpected property of parallel branches that allows pairs of messages that differ on only a small number of bits to be found. This possibly is caused by almost similar operations in both sides in a branch where functions *f* and *g* are used in a different order. In the following section, we describe our proposed cryptographic hash function STITCH-256 in more detail and highlight its difference with FORK-256 with strong justification.

3. Description of STITCH-256

STITCH-256 hashes 512-bit block messages to 256-bit message digests. Therefore an input message of arbitrary length 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 are appended. Then a bit '1' is appended, 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 STITCT-256 can be represented as:

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

3.1 STITCH-256 Compression Function

The compression function of STITCH-256 is originally motivated by the design of RIPEMD-family which runs in parallel of two branches. STITCH-256 has four parallel branches, B1, B2, B3 and B4 as illustrated in Fig. 1.

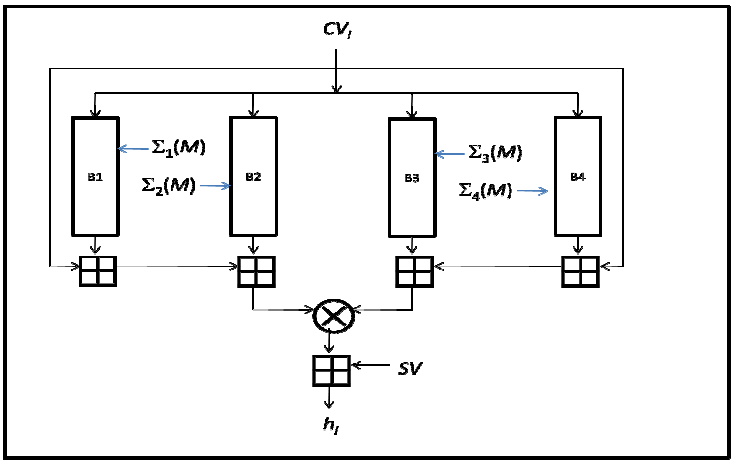

Fig. 3 Compression function of STITCH-256

CV_i is a 256-bit chaining variable at *i*-th iteration, is made up of eight 32-bit word registers *A, B, C, D, E, F, G* and *H. SV* is a fixed 32-bit salt value, 0x67452301. The initial values 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

To proceed to the iterated hash, each successive 512-bit message block *M* is split into 16 32-bit words $M_0, M_1, M_2, \ldots, M_{15}$. Fig. 3 shows that CV_i is updated to CV_{i+1} by computing:

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

where $\Sigma_j(M)$ is thirty two expanded and rearranged 32-bit message words M_i. The message expansion and rearrangement is described in the next section.

3.2 STITCH-256 Step Transformation, B_i

Step transformation of STITCH-256 is designed to maximize the propagation of the intermediate values difference, i.e to lower the probability of the differential characteristics. We allocate two phases for step transformation. The first step transformation is illustrated in Fig. 4.

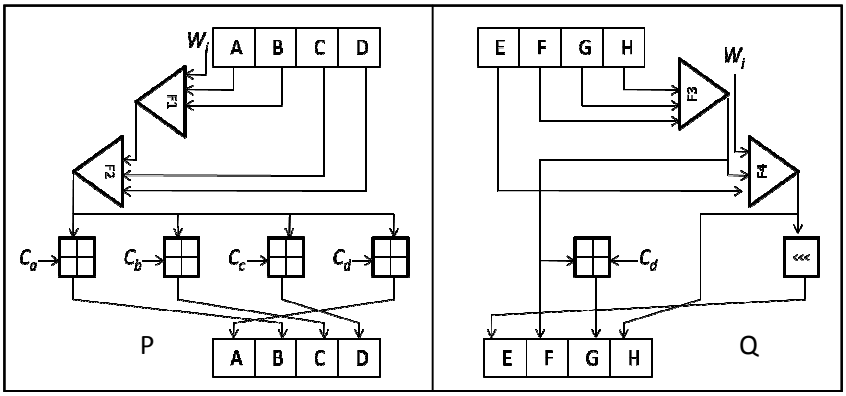

Fig. 4. First phase of single step transformation B_i of STITCH-256

First phase of step transformation is a phase where the chaining variables CV_i is equally divided into two halves, *P* and *Q*, and mixes with the message words M_i and constants to update the new registers.

The message arrangement $\Sigma_i(M)$ for all branches are different and are described in the next section. However, both *P* and *Q* in one branch accept the same message arrangement. The compression function of STITCH-256 uses four different constants, the first digit of π, in a single step transformation and the arrangement of the constants are different for both wings. Table 1 shows the 16 different constants used in the compression function of STITCH-256.

Table 1. Constants used in the compression function of STICTH-256

φ_1	243F6A8885A308D3	φ_2	13198A2E03707344	φ_3	A4093822299F31D0	φ_4	082EFA98EC4E6C89
φ_5	452821E638D01377	φ_6	BE5466CF34E90C6C	φ_7	C0AC29B7C97C50DD	φ_8	3F84D5B5B5470917
φ_9	9216D5D98979FB1B	φ_{10}	D1310BA698DFB5AC	φ_{11}	2FFD72DBD01ADFB7	φ_{12}	B8E1AFED6A267E96
φ_{13}	BA7C9045F12C7F99	φ_{14}	24A19947B3916CF7	φ_{15}	0801F2E2858EFC16	φ_{16}	636920D871574E69

These constants are used to each branch ordered as in Table 2.

Table 2. The ordering of constants in each wings of each branch

Branch		C_1	C_2	C_3	C_4
B1	P	φ_1	φ_2	φ_3	φ_4
	Q	-	-	-	φ_2
B2	P	φ_5	φ_6	φ_7	φ_8
	Q	-	-	-	φ_6
B3	P	φ_9	φ_{10}	φ_{11}	φ_{12}
	Q	-	-	-	φ_{10}
B4	P	φ_{13}	φ_{14}	φ_{15}	φ_{16}
	Q	-	-	-	φ_{14}

The constants are re-arranged as such to avoid the attack using cancellation technique. A single step transformation in each wing updates its new registers by producing the following outputs:

$A_{i+1} = c_d \boxplus f_2(C, D, f_1(A, B, W_i))$

$B_{i+1} = c_a \boxplus f_2(C, D, f_1(A, B, W_i))$

$C_{i+1} = c_b \boxplus f_2(C, D, f_1(A, B, W_i))$

$D_{i+1} = c_c \boxplus f_2(C, D, f_1(A, B, W_i))$

$E_{i+1} = ROT^{13}(f_4(E, W_i, f_3(F, G, H)))$
$F_{i+1} = f_3(F, G, H)$

$G_{i+1} = c_a \boxplus f_3(F, G, H)$

$H_{i+1} = f_4(E, W_i, f_3(F, G, H))$

Functions f_1, f_2, f_3 and f_4 are different in each branch. This is to disturb the attempt of an attacker to find good differential characteristics for parallel branches. The functions used in the compression function of STITCH-256 are mentioned in Table 3. The selection of functions used in STITCH-256 is carefully chosen in a way that all of them exhibit cryptographically strong properties.

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

Branch	P		Q	
	f_1	f_2	f_3	f_4
B1	p⊗q⊗r	(p∨ (q⊗ (~r))) ⊗r	p ⊗ ((p ⊗ q)∧r)	((p⊗ (~q)) ∨ r) ⊗q
B2	q⊗(p∨~r)	(p∧(q ⊗ r)) ⊗q	(p∧q)⊗(q∧r)⊗r	(p∧(q⊗r))⊗r
B3	(p ∨(q⊗ (~r))) ⊗q	p ⊗ (p∧q) ⊗ (q∧r)	((p ⊗ q) ∧ r) ⊗ q	q⊗ (p∨~r)
B4	((p⊗ (~r))∨ q) ⊗r	p⊗q⊗r	p⊗((p⊗(~r)) ∨q)	(p ∨(q⊗ (~r))) ⊗q / q⊗(p∨~r)

After each of a single step transformation is performed, the new register will be the input to the other wing of branch as illustrated in figure 5. This is the second phase of a single

step transformation. The propagation of intermediate values is maximized in a stitching way.

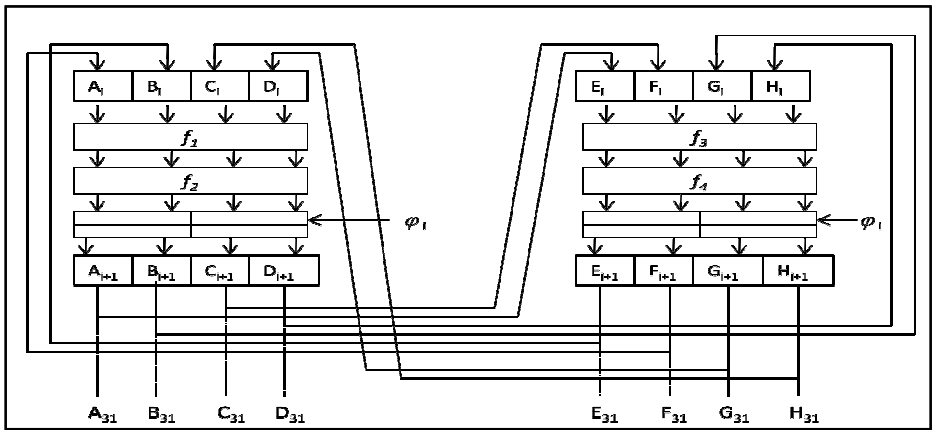

Fig. 5 Propagation of intermediate values in STITCH-256

Each branch iterates the step transformation for 32 rounds for both wings *P* and *Q*. The intermediate values for each iteration is swapped to the next wing in a branch as an input and the final registers are obtained after the final message word in a branch is iterated by concatenating all the registers from both wings:

$A_{31}\| B_{31}\| C_{31}\| D_{31}\| E_{31}\| F_{31}\| G_{31}\| H_{31}$

For iteration i+1, $0 \le i \le 30$, the step transformation updates its new registers by producing the following outputs:

$A_{i+1} = E_i$
$B_{i+1} = G_i$
$C_{i+1} = F_i$
$D_{i+1} = H_i$
$E_{i+1} = B_i$
$F_{i+1} = A_i$
$G_{i+1} = D_i$
$H_{i+1} = C_i$

3.3 Message expansion and rearrangement

To expand the message, we modify formulas σ_0 and σ_1 used in SHA-256 by testing a few parameters until the desired non-linear diffusion is achieved. Formulas σ_0 and σ_1 used in STITCH-256 are as follows:

$$Wi \begin{cases} Mi & 0 \le i \le 15 \\ \sigma_1(W_{i-7,}\, W_{i-16}) + W_{i-14} + \sigma_0(W_{i-9},\, W_{i-2}) + W_{i-3} \otimes SV & 16 \le i \le 32 \end{cases}$$

where $\sigma_0(x, y) = [\mathrm{ROT}^9(x) \otimes y] \otimes [\mathrm{ROT}^{18}(x) \otimes y] \otimes \mathrm{SHR}^3(x)$ and

$\sigma_1 = [\mathrm{ROT}^{15}(x) \otimes \mathrm{y}] \otimes [\mathrm{ROT}^{30}(x) \otimes y] \otimes \mathrm{SHR}^{10}(x)$

The total number of message words now becomes 32, where sixteen of them are the original message words and another sixteen are produced from the expansion process. To proceed to the iterated hash, we arrange the message words W_i as follows: Message words W_i where $0 \leq i \leq 15$, are arranged alternatively (or positioned in the odd number of array) such that the first message word M_0 is arranged in the first array R_0, second message word M_1 in the third array R_2, third message word M_2 in the fifth array R_5, so forth and so on until all the message words $W_0 \ldots W_{15}$ are arranged accordingly, filling up 16 arrays. Message words W_i where $16 \leq i \leq 32$ are then placed in the empty even arrays, such that W_{16} is placed in the second array R_1, W_{17} is placed in the forth array R_3, W_{18} is placed in the second array R_6, so forth and so on. The arrangement of the original and the expanded message words is illustrated in table 4.

Table 4 The arrangement of original and expanded message words in STITCH-256

W_0	W_{16}	W_1	W_{17}	W_2	W_{18}	W_3	W_{19}
W_4	W_{20}	W_5	W_{21}	W_6	W_{22}	W_7	W_{23}
W_8	W_{24}	W_9	W_{25}	W_{10}	W_{26}	W_{11}	W_{27}
W_{12}	W_{28}	W_{13}	W_{29}	W_{14}	W_{30}	W_{15}	W_{31}

This arrangement now becomes the initial arrangement of message words W_i and we now read it as $W_0, \ldots. W_{31}$. Next all the message words W_i are re-arranged for each branch following the arrangement described in Table 5. The number in each cell shows the message word's index, $W_1, \ldots . , W_{31}$.

Table 5. The re-arrangement of original message words and their expansion words

Branch/W_i	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
B1	0	1	26	27	20	21	14	15	8	9	2	3	28	29	22	23	16	17	10	11	4	5	30	31	24	25	18	19	12	13	6	7
B2	14	15	20	21	26	27	0	1	22	23	28	29	2	3	8	9	30	31	4	5	10	11	16	17	6	7	12	13	18	19	24	25
B3	24	25	18	19	16	17	10	10	16	17	10	11	4	5	30	31	8	9	2	3	28	29	22	23	0	1	26	27	20	21	14	15
B4	24	25	18	19	12	13	6	7	16	17	10	11	4	5	30	31	8	9	2	3	28	29	22	23	0	1	26	27	20	21	14	15

4. Design Strategy

The branch strategy in STITCH-256 is motivated by the design of RIPEMD-160 [15] which has two parallel lines running almost the same operations of MD5 and SHA-family hash functions. So far RIPEMD-160 hash function is still secure against the differential attack by Wang et. al [16] that was successfully attacking MD-family and SHA-family hash functions. The parallel design means that it has to take all the parallel branches simultaneously to build the differential characteristics with high probability. This is not a trivial job and another feature in STITCH which introduces message expansion and re-arrangement will increase the level of difficulty to build such characteristics. We manage to reduce the step operation of each branch to contribute to fast implementation of STITCH-256 while maintaining the security aspect by having a carefully studied of selected cryptographically strong Boolean functions f_1, f_2, f_3, f_4.

4.1 Design Principle

Compression Function of STITCH-256 It consists of four parallel branches, which is implemented to accept 512-bit input messages and give away 256-bit output message digests. STITCH-family can be considered to accept 1024-bit input messages and produce 512-bit output message digests by having a 512-bit register words in every branch where each wing *P* and *Q* processes 256-bit register words accordingly following the design principle of STITCH-256. In a case of STITCH-512, eight different Boolean functions will be involved in each branch and the number of constant values and the size of step transformations are doubled one time. However, we don't focus in STITCH-family in this paper; rather we just focus in STITCH-256. The security of compression function of STITCH-256 also comes highly from its message expansion and message re-arrangement. We believe that the level of security can be increased with the high message difference propagation throughout the step transformations. This is realized with the design of STITCH-256 message expansion in which the calculated Hamming codes are propagating the influence of message words vertically and horizontally.

Constants STITCH-256 uses 4 different constants in each branch, and they are used in different order in each wing in a branch. In total, STITCH-256 has 16 different constants for the four branches. The constants are the first digit of π, and differently selected to randomize the pattern of message differences.

Boolean functions There is only one diffusion function used in STITCH-256:

$$f_i = p \otimes q \otimes r.$$

Diffusion function is used in step transformation in all the branches. For non-linear functions, there are 4 different bitwise Boolean functions used in each branch. We

identified 12 distinct Boolean functions from Cellular Automata rules [21], and five of them are used twice to give sixteen Boolean functions in STITCH-256. All Boolean functions used in STITCH-256 are balanced functions, have been carefully studied for their cryptographic properties such as non-linearity, propagation criteria and high algebraic degree. The summary of their cryptographic properties are summarized in Table 6.

Table 6: Cryptographic properties of Boolean functions used in STITCH-256

Boolean functions	Non-linearity	Propagation criterion of order *m*	Algebraic degree of order *n*
p⊗((p⊗(~r)) ∨ q)	2	1	2
((p⊗ (~q)) ∨ r) ⊗q	2	1	2
(p ∨(q⊗ (~r))) ⊗q	2	1	2
((p⊗ (~r))∨ q) ⊗r	2	1	2
(p∨ (q⊗ (~r))) ⊗r	2	1	2
(p∧(q ⊗ r)) ⊗q	2	1	2
p ⊗ (p∧q) ⊗ (q∧r)	2	1	2
(p∧(q⊗r))⊗r	2	1	2
p ⊗ ((p ⊗ q)∧r)	2	1	2
(p∧q)⊗(q∧r)⊗r	2	1	2
((p ⊗ q) ∧ r) ⊗ q	2	1	2

For 3-variables Boolean functions, the desired cryptographic properties are the maximum non-linearity which is 2, propagation criterion of order 1 and high algebraic degree, 2. Further explanation of all the cryptographic properties can be referred to, for eg [22], [23], [24].

Message expansion and message re-arrangements STITCH-256 expands the original message words from 16 to 32 32-bit message words. This is done by calculating dividing the message block of 512 bits into 16 32-bit message words, and the expansion is following the formulas specified earlier with an objective that the bits are mixed to the maximum.

5. Preliminary Security Analysis of STITCH-256

Assume that the attacker inserts a message difference Δm = 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:

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

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.

1. The attacker constructs $\alpha + \Delta 2 = -(\beta + \Delta 3 + SV)$ to have a collision to occur. He can also reduce the complexity by constructing $\Delta 2 = -\Delta 3$ and $\alpha = -(\beta + SV)$. However differential pattern of the message words $\Delta 2 = -\Delta 3$ 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}.
2. The attacker constructs two distinct differential characteristics and expects that $\alpha = -\Delta 2$ and $\beta = -\Delta 3$. 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.

The strength of STITCH-256 lies in its message expansion which gives better bit propagation, message rearrangement which gives more difficulties for an attacker to construct differential characteristics with high probability and bit propagation through stitching step operation. The combination of these three characteristics gives better security for STITCH-256 on a whole.

We compare the primitives, number of operations and cycles used in STITCH-256, FORK-256 and SHA-256. The comparison is illustrated in Table 7.

Table 7. Number of operations used in step transformation of STITCH-256, FORK-256 and SHA-256

Operation	STITCH-256	FORK-256	SHA-256
Addition (+)	640	472	600
Bitwise operation ($\otimes$,$\wedge$,$\vee$)	513	328	1024
Shift (<< , >>)	-	-	96
Shift rotation (<<< , >>>)	128	512	576
TOTAL OPERATIONS	1281	1312	2296
Block size (bits)	512	512	512
Maximum message size (bits)	$2^{64} - 1$	$2^{64} - 1$	$2^{64} - 1$
Output size (bits)	256	256	256
Rounds	32	32	64

From table 6, we can see that STITCH-256 has lesser total operations than both FORK-256 and SHA-256. This will constitute to a faster performance as a whole function. The whole function also has exhibited a good avalanche effect when half of the output bits are changed on average of 1000 sample experiments.

6. Conclusion

In this paper, a new cryptographic hash function, STITCH-256 is proposed. It takes arbitrary length of message and outputs 256 bits message digest. STITCH-256 introduces a few components in its structure, namely message expansion and rearrangement, and stitching step operations. These components are designed to give a better security in STITCH-256. STITCH-256 also processes the bits in four parallel branches inspired by RIPEMD and FORK-256 but compress the outputs from the four branches slightly different from that of

FORK-256. STITCH-256 is a simple but elegant cryptographic hash functions. It has shown good bit propagation through the whole step operations and we hope to receive further analysis on the security of STITCH-256.

References

[1] R. Rivest. The MD4 Message-Digest Algorithm, Request for Comments (RFC) 1320", Internet Activities Board, Internet Privacy Task Force, 1992.
[2] B. d. Boer, and A. Bosselaers. An Attack on the Last Two Rounds of MD4, In J. Feigenbaum, editor, *Advances in Cryptology – CRYPTO '91*, volume 576 of LNCS, pages 194-203. Springer-Verlag, 1991.
[3] R. Rivest. The MD5 Message-Digest Algorithm, Request for Comments (RFC) 1321", Internet Activities Board, Internet Privacy Task Force, 1992.
[4] B. d. Boer, and A. Bosselaers. Collisions for the Compression Function of MD5, In *Advances in Cryptology – EuroCRYPT '93*, volume 765 of LNCS, pages 193-304. Springer-Verlag, 1994.
[5] H. Dobbertin. Cryptanalysis of MD4, Journal of Cryptology 11:4, pp. 253-271, 1998.
[6] H. Dobbertin. RIPEMD with Two Round Compression Function is Not Collision-Free, Journal of Cryptology, 10(1):51–70, 1997.
[7] H. Dobbertin. The first two rounds of MD4 are not one-way, In *Fast Software Encryption – FSE '98*, volume 1372 of LNCS, pages 284–292. Springer, 1998.
[8] H. Dobbertin. Cryptanalysis of MD5, Presented at the rump session of EUROCRYPT '96, May 12-16, 1996.
[9] H. Dobbertin. The Status of MD5 After a Recent Attack. CryptoBytes, 2(2):1,3–6, 1996.
[10] X. Wang, Y. L. Yin, and H. Yu. Efficient Collision Search Attacks on SHA-0, In *Advance in Cryptology – CRYPTO 05*, vol. 3621 of LNCS, pp. 1-16, Springer-Verlag, August 2005.
[11] X. Wang, Y. L. Yin, and H. Yu. Finding Collisions in the Full SHA-1, In *Advances in Cryptology – CRYPTO 05*, vol. 3621 of LNCS, pp. 17-36, Springer-Verlag, August 2005.
[12] X. Wang and H. Yu. How to Break MD5 and Other Hash Functions, In *Advances in Cryptology – EUROCRYPT 2005*, LNCS 3494, Springer-Verlag, pp. 19-35, 2005.
[13] FIPS 180. Secure Hash Standard (SHS). National Institute of Standards and Technology, May 1993. Replaced by [20].
[14] H. Dobbertin, A. Bosselaers and B. Preneel. RIPEMD-160, A Strengthened Version of RIPEMD, FSE '96, LNCS 1039, Springer-Verlag, pp. 71-82, 1996.
[15] D. Hong, J. Sung, S. Hong, S. Lee, and D. Moon. A New Dedicated 256-bit Hash Function: FORK-256, First NIST Workshop on Hash Functions, 2005.
[16] B. Preneel, A. Bosselaers, and H. Dobbertin. RIPEMD-160: A Strenghtened Version of RIPEMD. In D. Gollman, editor, Fast Software Encryption – FSE '96, volume 1039 of LNCS, pages 71-82. Springer-Verlag, 1997.
[17] K. Matusiewiez, T. Peyrin, O. Billet, S. Contini, and J. Pieprzyk. Cryptanalysis of FORK-256, In *Proc. Fast Software Encryption 2007*, March 26-28, 2007, Luxumbourg; LNCS 4593, Springer, 2007.
[18] K. Matusiewiez, S. Contini, and J. Pieprzyk. Weaknesses of the FORK-256 compression function, IACR Cryptology e-print Archive 2006, report 2006/317.

[19] S. Kontini, K. Matusiewicz, and J. Pieprzyk. Extending FORK-256 Attack to the Full Hash Function, Proc. ICICS 2007, December 12-15, Zhengzhou, China; LNCS 4861, Springer, 2007.
[20] National Institute of Standards and Technology. Secure hash standard(SHS). FIPS 180-2, August 2002.
[21] S. Wolfram. Universality and complexity in cellular automata. *Physica D*, 10:1-35, 1984.
[22] T. Siegenthaler. Correlation-immunity of non-linear combining functions for cryptographic applications, IEEE Trans. On Information Theory, vol. IT-30, no. 5, pp. 776-780, 1984.
[23] P. Camion, C. Carlet, P. Charpin, and N. Sendrier, On correlation immune functions, in Advances of Cryptology, CRYPTO 1991, Santa Barbara, USA, 1991, 00. 86-100.
[24] B. Preneel, W. V. Leekwijck, L. V. Linden, R. Govaerts, and J. Vandewalle. Propagation Characteristics of Boolean functions. In Advances in Cryptology – EUROCRYPT ’90, vol. 437 of Lecture Notes in Computer Science, pp. 155-165. Springer-Verlag, Berlin, Heidelberg, New York, 1991.