World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:10, 2008

Evaluation on Recent committed Crypt Analysis Hash Function

A. Arul Lawrence Selvakumar, and C. Suresh Ganandhas

***Abstract*—**This paper describes the study of cryptographic hash functions, one of the most important classes of primitives used in recent techniques in cryptography. The main aim is the development of recent crypt analysis hash function. We present different approaches to defining security properties more formally and present basic attack on hash function. We recall Merkle-Damgard security properties of iterated hash function. The Main aim of this paper is the development of recent techniques applicable to crypt Analysis hash function, mainly from SHA family. Recent proposed attacks an MD5 & SHA motivate a new hash function design. It is designed not only to have higher security but also to be faster than SHA-256. The performance of the new hash function is at least 30% better than that of SHA-256 in software. And it is secure against any known cryptographic attacks on hash functions.

***Keywords*—**Crypt Analysis, cryptographic.

I. Introduction

FOR cryptographic hash function, the following properties are required:

- **Preimage resistance:** it is computationally infeasible to find any input which hashes to any pre-specified output.
- **Second preimage resistance:** it is computationally infeasible to find any second input which has the same output as any specified input.
- **Collision resistance:** it is computationally infeasible to find a collision, i.e. two distinct inputs that hash to the same result.

For an ideal hash function with an m-bit output, finding a preimage or a second preimage requires about 2^m operations and the fastest way to find a collision is a birthday attack which needs approximately $2^{m/2}$ operations. Most dedicated hash functions which have iterative process use the Merkle-Damgard construction [6, 10] in order to hash inputs of arbitrary length. They work as follows. Let **HASH** be a hash function. The message X is padded to a multiple of the block length and subsequently divided into t blocks $X_1,\cdots, X_t$. Then **HASH** can be described as follows:

CV_0= IV; CV_i = **COMP** $(CV_{i-1,} X_i)$, $1 \le i \le t$; **HASH** *(X)* = CV_t,

where **COMP** is the compression function of **HASH**, CV_i is the chaining variable between stage i and stage i + 1, and IV denotes the initial value. The most popular method of designing compression functions of dedicated hash functions is a serial successive iteration of a small step function, as like round functions of block ciphers.

Many hash functions such as MD4 [12], MD5 [13], HAVAL [19], SHA-family [11], etc., follow that idea. Attacks on hash functions have been focused on vanishing the difference of intermediate values caused by the difference of messages. On the other hand, a hash function has been considered secure if it is computationally hard to vanish such difference in its compression function. Usually, the lower the probability of the differential characteristic is, the harder the attack is.

Therefore a step function is regarded as a good candidate if it causes a good avalanche effect in the serial structure. A function which has a good diffusion property can not be so light in general. However, most step functions have been developed to be light for efficiency. This may be why MD4-type hash functions including SHA-1 are vulnerable to Wang et al.'s collision-finding attack [15–18].

RIPEMD-family [9] has somewhat different approach for designing a secure hash function. The attacker who tries to break members of RIPEMD-family should aim simultaneously at two ways where the message difference passes. This design strategy is still successful because so far there is not any effective attack on RIPEMD-family except the first proposal of RIPEMD. However, RIPEMD-family have heavier compression functions than hash functions with serial structure. For example, the first proposal of RIPEMD consists of two lines of MD4. Total number of steps is twice as many as that of MD4. Also, the number of steps of RIPEMD-160 is almost twice as many as that of SHA-0.

In this paper, we propose a new dedicated hash function FORK-256. According to the above observation, we determined the design goals as follows.

- It should have a 256-bit output because the security of 2^{128}operations is recommended for symmetric key cryptography as the computing power increases.
- Its structure should be resistant against known attacks including Wang et al.'s attack [1–5, 7, 8, 14–18].
- The performance should be as competitive as that of SHA-256.

A. Arul Lawrence Selvakumar is with Department of Computer Science and Engineering, Oxford college of Engineering, Bangalore, India (e-mail: aarul72@hotmail.com).

C. Suresh Ganandhas is with Department of Computer Science and Engineering, VelMultitech SRS Engineering College, Chennai, India (e-mail: sureshc_me@yahooo.com).

II. Description of FORK-256

In this section, we will describe FORK-256. These are basic notations used in FORK-256.

$\boxplus$: Addition mod 2^{32}
$\oplus$: XOR (exclusive OR)
$A^{<<<s}$: s-bit left rotation for a 32-bit string A

A. Input Block Length and Padding

An input message is processed by 512-bit block. FORK-256 pads a message by appending a single bit 1 next to the least significant bit of the message, followed by zero or more bits 0's until the length of the message is 448 modulo 512, and then appends to the message the 64-bit original message length modulo 2^{64}.

B. Structure of Fork-256

Fig. 1 depicts the outline of the compression function of FORK-256. The name 'FORK' was originated from the figure. The compression function of FORK-256 hashes a 512-bit string to a 256-bit string. It consists of four parallel branch functions, $BRANCH_1$, $BRANCH_2$, $BRANCH_3$, and $BRANCH_4$. Let CV_i = (A, B, C, D, E, F, G, H) be the chaining variable of the compression function. It is initialized to IV_0which is:

$A=6a09e667_x$ $B=bb67ae85_x$ $C=3c6ef372_x$ $D=a54ff53a_x$
$E=510e527f_x$ $F=9b05688c_x$ $G=1f83d9ab_x$ $H=5be0cd19_x$

Each successive 512-bit message block M is divided into sixteen 32-bit words M_0, $M_{1,\ldots,}M_{15}$ and the following computation is performed to update CV_i to CV_{i+1}:

$$CV_{i+1} = CV_i \boxplus \{[BRANCH_1 (CV_{i,} \Sigma_1 (M)) \boxplus BRANCH_2(CV_i,\Sigma_2(M))] \oplus [BRANCH_3 (CV_i, \Sigma_3(M)) \boxplus BRANCH_4(CV_i,\Sigma_4(M))]\},$$

Where $\Sigma_j(M) = (M_{\sigma j} (0)\ldots, M\sigma j \ (15))$ is the re-ordering of message words for j = 1, 2, 3, 4, given by Table I.

C. Branch Functions: $BRANCH_j$

Each $BRANCH_j$ is computed as follows:

1) The chaining variable CV_i is copied to initial variables $V_{j,0}$ for j-th branch.
2) At k-th step of each $BRANCH_j(0 \le k \le 7)$, the step function $STEP_{j,k}$ is computed as follows:

$$V_{j,k+1} = STEP_{j,k}(V_{j,k}, M\sigma_j(2k) , M\sigma_j(2k+1), \alpha_{j,k}, \beta_{j,k})$$
Where $\alpha_{j,k}$ and $\beta_{j,k}$ are constants.

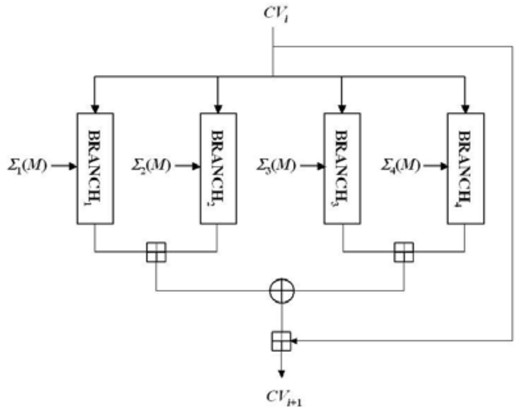

Fig. 1 Outline of the FORK-256 compression function

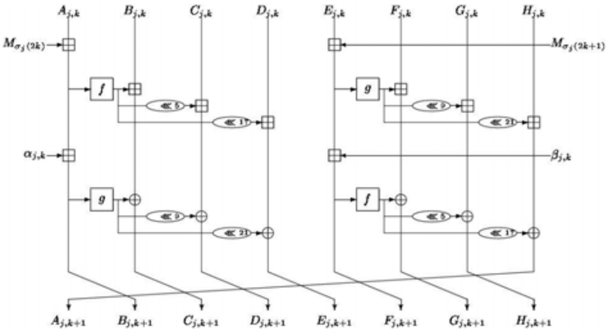

Fig. 2 Step function of FORK-256, $STEP_{j,\ k}$

Input Order of Message Words: This table shows the input order of message words M_0~ M_{15}applied to $BRANCH_j$ (1≤j≤4) functions.

TABLE I
Ordering Rule of Message Words

T	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ 1(t)	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
σ2(t)	14	15	11	9	8	10	3	4	2	13	0	5	6	7	12	1
σ3(t)	7	6	10	14	13	2	9	2	11	4	15	8	5	0	1	3
σ4(t)	5	12	1	8	15	0	13	11	3	10	9	2	7	14	4	6

Constants: The compression function of FORK-256 uses sixteen constants given by the following table:

δ0 = 428a2f98x	δ1 = 71374491x
δ2 = b5c0fbcfx	δ3 = e9b5dba5x
δ4 = 3956c25bx	δ5 = 59f111f1x
δ6 = 923f82a4x	δ7 = ab1c5ed5x
δ8 = d807aa98x	δ9 = 12835b01x
δ10 = 243185bex	δ11 = 550c7dc3x
δ12 = 72be5d74x	δ13 = 80deb1fex
δ14 = 9dbc06a7x	δ15 = c19bf174x

These constants are applied to each $BRANCH_j$ according to the ordering rule of them as follows:

Step k	α1,k	β1,k	α2,k	β2,k	α3,k	β3,k	α4,k	β4,k
0	δ 0	Δ1	δ15	Δ14	δ1	δ0	δ14	δ15
1	δ2	δ3	δ13	δ12	δ3	δ2	δ12	δ13
2	δ4	δ5	δ11	δ10	δ5	δ4	δ10	δ11
3	δ6	δ7	δ9	δ8	δ7	δ6	δ8	δ9
4	δ8	δ9	δ7	δ6	δ9	δ8	δ6	δ7
5	δ10	δ11	δ5	δ4	δ11	δ10	δ4	δ5
6	δ12	δ13	δ3	δ2	δ13	δ12	δ2	δ3
7	δ14	δ15	δ1	δ0	δ15	δ14	δ0	δ1

Step Functions: STEP $_{j,k}$ The input register $V_{j,k}$ of $STEP_{j,,k}$ is divided into eight 32-bit words:

$V_{j,k}= (A_{j,k}, B_{j,k}, C_{j,k}, D_{j,k}, E_{j,k}, F_{j,k}, G_{j,k}, H_{j,k})$

$STEP_{j,k}$ takes $V_{j,k}$, $M\sigma j(2k)$, $M\sigma j(2k+1)$, $\alpha_{j,k}$ and $\beta_{j,k}$ as inputs, and then provides the output as follows (See Fig 2):

$A_{j,k+1} =H_{j,k}\boxplus g(E_{j,k}\boxplus M\sigma j(2k+1))^{<<<21} \oplus f(E_{j,k}\boxplus M\sigma j(2k+1)\boxplus\beta_{j,k})^{<<<17}$

$B_{j,\ k+1} = A_{j,k}\boxplus M\sigma_j(2k) \boxplus \alpha_{\ j,k},$

$C_{j,k+1} = B_{j,k} \boxplus f (A_{j,k} \boxplus M\sigma j(2k)) \oplus g(A_{j,k}M\sigma j\ (2k) \boxplus\alpha_{j,k}),$

$D_{j,k+1} =C_{j,k}\boxplus f(A_{j,k}M\sigma_j(2k))^{<<<5} \oplus g(A_{j,k}\boxplus M\sigma_j(2k)\boxplus\alpha_{j,k})^{<<<9,}$

$E_{j,k}+1=D_{j,k}\boxplus f(A_{j,k}M\sigma j(2k))^{<<<17} \oplus g(A_{j,k}M\sigma j(2k)\boxplus\alpha_{j,k})^{<<<21,}$

$F_{j,k}+1=E_{j,k}\boxplus M\sigma_j\,(2k+1) \boxplus \beta_{j,k},$

$G_{j,k}+1=F_{j,k}\boxplus g(E_{j,k}\boxplus M\sigma j(2k+1)) \oplus f(E_{j,k}\boxplus M\sigma j(2k+1)\boxplus\beta_{j,k}),$

$H_{j,k}+1=G_{j,k}g\boxplus g(E_{j,k}\boxplus M\sigma_j(2k+1))^{<<<9} \oplus f(E_{j,k}\boxplus M\sigma_j(2k+1) \boxplus\beta_{j,k})^{<<<5,}$

Where f and g are nonlinear functions as follows:

$f(x) = x \boxplus (x^{<<<7} \oplus x^{<<<22}),$

$g(x) = x \oplus (x^{<<<13}\boxplus x^{<<<27}).$

III. Design Strategy

A. Motivation for our Proposal

In Wang et al.'s attacks on MD4, MD5, HAVAL, and RIPEMD [15, 16] and SHA-0/1 [17, 18] brought the big impact on the field of symmetric key cryptography including hash function. However, RIPEMD-128/160 is the algorithms which are still secure against their attacks. No attacks on them are found so far.

They were designed to have two parallel lines, which is different from MD4, MD5 and SHA-family. This makes an attacker take into account two lines simultaneously. However, since each line needs almost same operation of MD5 and SHA algorithms, its efficiency was degenerated almost half of them. This motivates our design. We use four lines instead of two. In order to overcome disadvantage of RIPEMD algorithms, we manage to reduce operations for step functions of each line. The message reordering of each branch is deliberately designed to be resistant against Wang et al.'s attack and differential attacks. The function f and g in each step are chosen to have good avalanche effects.

B. Design Principle

Structure FORK-256 consists of 4 Branches. In the security aspect, we can give the security against known attacks with the different message-ordering in branches. For example, RIPEMD, which consists of 2 branches, was fully attacked by Wang et al. because RIPEMD has same message-ordering in 2 branches. On the other hand, in case of RIPEMD-128/160, there is no attack result because RIPEMD-128/160 have different message-ordering in branches. In the implementation aspect, FORK-256 can be implemented efficiently be cause the message-ordering is simpler than the message expansion such as that of SHA-256.

Constants: Each $BRANCH_i$ uses 16 different constants $\alpha_{i,j}$ and $\beta_{i,j}$ for $j = 0,\cdots,7$. By using constants we pursue the goal to disturb the attacker who tries to find a good differential characteristic with a relatively high probability. So, we prefer the constants which represent the first thirty-two bits of the fractional parts of the cube roots of the first sixteen four prime numbers.

Nonlinear Functions: Nonlinear functions f and g output one word with one input word. Almost dedicated hash functions use boolean functions which output one word with three words at least. The boolean functions make it easy to control the output one word by adjusting the input several words. The attacks on MD4, MD5, HAVAL, RIPEMD and SHA-0/1 are based on this weakness of Boolean functions. In addition, the output words of f and g functions are used to update other chaining variables. In almost dedicated hash functions output words of boolean functions are used to update only one chaining variable. This weakness is also used to analyze above hash functions.

Shift Rotations in Nonlinear Functions: If the addition is changed into the bitwise x or operation in f and g, nonlinear functions are generalized as

$$x \oplus (x^{<<<s1} \oplus x^{<<<s2})$$

We consider all 465 (=$_{31}C_2$) cases for $s_1$and s_2 and want to define shift rotations satisfying the following 7 conditions. HW(x) denotes the Hamming Weight of x.

- The branch number of f and g is four.
- If HW (input word) = 2, then HW (output word) ≥ 4.
- If HW (input word) = 3, then HW (output word) ≥ 3.
- If HW (input word) = 4, then HW (output word) ≥ 4.
- If HW (output word) = 1, then HW (input word) ≥ 17.
- If HW (output word) = 2, then HW (input word) ≥ 14.
- The interval of shift rotations are greater than or equal to 4.

By above all conditions, we have defined f and g functions.

Ordering of Message Words We adopt the message word ordering instead of the message word extension. If an attacker constructs an intended differential characteristics for one branch function, the ordering of message words will cause unintended differential patterns in the other branch functions. This is the core part of the security in the compression function. When we define the ordering of message words, following four conditions are considered.

- Balance of upper (step 0~3) and lower (step 4~7) parts: Each value is applied twice to upper and lower parts, respectively.
- Balance of left and right parts: Each value is applied twice to left and right parts, respectively.
- Balance of sums of input orders Each word is applied four times and is indexed by 0~15.
- Total sum of indexes is 480. Therefore, the average of sum of indexes applied to each word is 30.
- We search the ordering so that the sum of indexes Corresponding to each word is 25~35.
- Conditions which do not have same differential Patterns in all branches
- Specific differential pattern used at a branch may be applied to other branches.
- Therefore, except the case of giving a same difference to all words, we try to find an ordering such that there is no same differential patterns in all branches.

Shift Rotations and Rank: In the step function, 5 and 17, the values of shift rotation, are fixed. Then we search all the case and find candidate values (corresponding to 9 and 21) so that the rank of the linearly-changed step function is maximized. The maximum of the rank is 252.

Finally we select 9 and 21 among candidate values so that differences generated from the outputs of f and g functions do not overlap when a message word inputted at a step function has a one-bit difference.

IV. Security Analysis of FORK-256

A. Collision-Finding Attack

Assume that an attacker inserts the message difference. Let Δ_I be the output difference of i-th branch $BRANCH_i$. Then the attacker expects the following event for finding collisions:

$$(\Delta_1 \boxplus \Delta_2) \oplus (\Delta_3 \boxplus \Delta_4) = 0.$$

For this, he can take several strategies:

1. The attacker constructs a differential characteristic with a high probability for a branch function, say $BRANCH_1$, and then expects that the operation of the output differences in the other branches, $\Delta_3 \boxplus \Delta_4 \boxplus \Delta_2$ is equal to Δ_1.
2. The attacker constructs two distinct differential characteristics, and expects that $\Delta_1 = -\Delta_2$ and $\Delta_3 = -\Delta_4$.
3. The attacker inserts the message difference which yields same message difference pattern in four branches, and expects that same differential character istic occurs simultaneously in four branches. Then the output difference of the compression function vanishes if the hamming weight of the output difference of each branch is small. This is because the final output is generated with using $\oplus$ and $\boxplus$ by turns.

Let us see the first strategy. If we assume that the outputs of each branch function are random, the probability of the event is almost close to 2_{-256}. It is also difficult for the attacker to mount any attack following the second strategy because he should find such differential pattern of the message words.

Third strategy is relatively easy for the attacker to perform. For example, if he inserts the same difference to all the message words, then the same message difference pattern occurs in every branches. However, the message word reordering was designed so that the third strategy is satisfied only if the attacker inserts the same difference to all the message words. Under the assumption that every step is independent, we can compute the upper bound of the probability that such kind of differential characteristic occurs, which frustrates the attacker.

B. Attacks Using Inner Collision Patterns

When the attacker inserts the differences to the message words, the event that the difference of the intermediate value becomes zero often occurs. It is called inner collision. We call a differential characteristic which causes an inner collision with a probability, inner collision pattern. Note that an inner collision is not a real collision, but the notion of inner collision pattern is important in cryptanalysis of hash function because it can be repeatedly used to yield a real collision with a high probability. The main idea of attacks on SHA-0 and SHA-1 is also the repetition of an inner collision pattern.

So, in hash functions with a serial structure it is related to the resistance against collision-finding attack how many times an inner collision can be repeated. Let us focus on only one branch function, say $BRANCH_1$. We can construct 5-step inner collision pattern easily. Let $\Delta A, \Delta B, \ldots, \Delta H$ denote the differences of $A_{1,k}, B_{1,k}, \ldots, H_{1,k}$, respectively. ΔM_L and ΔM_R denote the differences of $M\sigma_1$ (2k) and $M\sigma_1$ (2k+1), respectively. We found 5-step inner collision patterns of FORK-256 with the probability 2^{-40} as listed in Table II and III. If we apply these patterns to $BRANCH_1$, the output difference $\Delta_1$will be zero with the probability 2^{-40}.

As mentioned in the previous subsection, however, it is hard to use the pattern for the attack on FORK-256 because the following events seldom occurs: either that the computation of the output differences of the other branches is zero or that the other branches have the same differential pattern in the message words as $BRANCH_1$.

TABLE II
Case 1: 5-Step Inner Collision Pattern of FORK-256:
The Numbers in the Entries of the Table Denotes the Bits in which the Difference is 1

Step	ΔA	ΔB	ΔC	ΔD	Δ E	ΔF	ΔG	ΔH	ΔM_L	ΔM_R	Prob.
0									31		
1		31	6,12 1,26	3,4 8,11 21,26	1,6 15,16					1,6 15,16 20,23	2^{-16}
2			31	6,12 21,26						3,4 8,11 21,26	2^{-10}
3				31						6,12 21,26	2^{-4}
4										31	1

TABLE III
CASE 2: 5-STEP INNER COLLISION PATTERN OF FORK-256: THE NUMBERS IN THE ENTRIES OF THE TABLE DENOTES THE BITS IN WHICH THE DIFFERENCE IS 1

Step	ΔA	ΔB	ΔC	ΔD	ΔE	ΔF	ΔG	ΔH	ΔM_L	ΔM_R	Prob
0										31	2_{-10}
1	1,6 15,16 20,23					31	6,12 21,26	3,4 8,11 21,26	1,6 15,16 20,23		2_{-16}
2	3,4 8,11 21,26						31	6,12 21,26	3,4 8,11 21.26		2_{-10}
3	6,12 21,26							31	6,12 21,26		2_{-4}
4	31								31		1

V. EFFICIENCY AND PERFORMANCE

In this section we compare the total number of operations and the performance of FORK-256 and SHA-256. The total number of operations is compared in the Table IV, Implementations were written in C language. We denote the simulation environment as CPU/OS/Compiler. The performance is compared in the following environments:

TABLE IV
NUMBER OF OPERATIONS USED IN FORK-256 AND SHA-256

Operation	*Fork – 256*	*SHA-256*
Addition (+)	472	600
Bitwise operation (⊕ , ∨ ∧)	328	1024
Shift (<<,>>)		96
Shift rotation (<<<, >>>)	512	576

– P3/WinXP/VC
– P4/WinXP/VC

Where the notations are as follows:

P3: Pentium III, 801 MHz, 192MB RAM
P4: Pentium IV, 2.0 GHz, 768MB RAM

WinXP : Microsoft Windows XP Professional ver 2002
VC : Microsoft Visual C++ Ver 6.0

TABLE V
PERFORMANCE OF FORK-256 AND SHA-256 ON SEVERAL ENVIRONMENTS

Environment	FORK - 256		SHA - 256	
	Mbps	Cycle/Byte	Mbps	Cycle/Byte
P3/WinXP/VC	192.1	31.413	132.46	44.581
P4/WinXp/VC	521.1	28.755	318.72	46.372

These implementations of FORK-256 are not optimized, so we expect performance can be improved for the optimized version.

VI. CONCLUSION

In this paper we have proposed a recent committed crypt analysis 256-bit hash function FORK 256, which is designed to be not only secure but also fast than SHA-256. The main features are the followings;

– Four branches are used in parallel, where as SHA-256 uses four serial rounds. This means that FORK-256 can be implemented in hardware and it is difficult to analyze all branches simultaneously.
– Unlike other dedicated hash functions, FORK-256 doesn't use Boolean functions but uses other nonlinear functions which output one word with one input word.
– Especially, FORK-256 updates several words with using one word.

These properties make it difficult to analyze FORK-256 with known attack methods including Wang et al.'s attack.

It is believed that FORK-256 is secure against any known attacks on hash functions. However, the extensive analysis of our new hash function is required and also we believe that Fork-512 is highly secured attack are developed latter. We believe that new FORK 512 hash function is launched in future with high security measures.

REFERENCES

[1] E. Biham and R. Chen, "Near-Collisions of SHA-0," Advances in Cryptology CRYPTO 2004, LNCS 3152, Springer-Verlag, pp. 290–305, 2004.
[2] E. Biham, R. Chen, A. Joux, P. Carribault, C. Lemuet and W. Jalby, "Collisions of SHA-0 and Reduced SHA-1,"Advances in Cryptology – EUROCRYPT 2005, LNCS 3494, Springer-Verlag, pp. 36–57, 2005.
[3] B. den Boer and A. Bosselaers, "An Attack on the Last Two Rounds of MD4," Advances in Cryptology – CRYPTO'91, LNCS 576, Springer-Verlag, pp. 194–203, 1992.
[4] B. den Boer and A. Bosselaers, "Collisions for the Compression Function of MD5," Advances in Cryptology – CRYPTO'93, LNCS 765, Springer-Verlag, pp. 293–304, 1994.
[5] F. Chabaud and A. Joux, "Differential Collisions in SHA-0," Advances in Cryptol ogy – CRYPTO'98, LNCS 1462, Springer-Verlag, pp. 56–71, 1998.
[6] I. Damg˚ard, "A Design Priciple for Hash Functions," Advances in Cryptology CRYPTO'89, LNCS 435, Springer-Verlag, pp. 416–427, 1989.
[7] H. Dobbertin, "RIPEMD with Two-Round Compress Function is Not Collision- Free," Journal of Cryptology 10:1, pp. 51–70, 1997.
[8] H. Dobbertin, "Cryptanalysis of MD4," Journal of Cryptology 11:4, pp. 253–271, 1998.
[9] H. Dobbertin, A. Bosselaers and B. Preneel, "RIPEMD-160, a strengthened version of RIPEMD," FSE'96, LNCS 1039, Springer-Verlag, pp. 71–82, 1996.
[10] R. C. Merkle, "One way hash functions and DES," Advances in Cryptology CRYPTO'89, LNCS 435, Springer-Verlag, pages 428–446, 1989.
[11] NIST/NSA, "FIPS 180-2: Secure Hash Standard (SHS)", August 2002 (change notice: February 2004).
[12] R. L. Rivest, "The MD4 Message Digest Algorithm," Advances in Cryptology CRYPTO'90, LNCS 537, Springer-Verlag, pp. 303–311, 1991.
[13] R. L. Rivest, "The MD5 Message-Digest Algorithm," IETF Request for Comments, RFC 1321, April 1992.
[14] B. Van Rompay, A. Biryukov, B. Preneel and J. Vandewalle, "Cryptanalysis of 3- pass HAVAL," Advances in Cryptology – ASIACRYPT 2003, LNCS 2894, Springer- Verlag, pp. 228–245, 2003.
[15] X. Wang, X. Lai, D. Feng, H. Chen and X. Yu, "Cryptanalysis of the Hash Func tions MD4 and RIPEMD," Advances in Cryptology – EUROCRYPT 2005, LNCS 3494, Springer-Verlag, pp. 1–18, 2005.
[16] X. Wang and H. Yu, "How to Break MD5 and Other Hash Functions," Advances in Cryptology – EUROCRYPT 2005, LNCS 3494, Springer-Verlag, pp. 19–35, 2005.

[17] X. Wang, H. Yu and Y. L. Yin, “Efficient Collision Search Attacks on SHA-0,” Advances in Cryptology – CRYPTO 2005, LNCS 3621, Springer-Verlag, pp. 1–16, 2005.

[18] X. Wang, Y. L. Yin and H. Yu, “Finding Collisions in the Full SHA-1,” Advances in Cryptology – CRYPTO 2005, LNCS 3621, Springer-Verlag, pp. 17-36, 2005.

[19] Y. Zheng, J. Pieprzyk and J. Seberry, “HAVAL – A One-Way Hashing Algorithm with Variable Length of Output,” Advances in Cryptology – AUSCRYPT’92, LNCS 718, Springer-Verlag, pp. 83–104, 1993.

Open Science Index, Computer and Information Engineering Vol:2, No:10, 2008 waset.org/Publication/3917