The RIPEMD^L and RIPEMD^R Improved Variants of MD4 Are Not Collision Free

Christophe Debaert[1] and Henri Gilbert[2]

[1] DGA/DCE/CELAR
[2] France Télécom R&D

Abstract. In 1992, the cryptographic hash function RIPEMD, a European proposal, was introduced as an improved variant of the MD4 hash function. RIPEMD involves two parallel lines of modified versions of the MD4 compression function. Three years later, an attack against a reduced version of RIPEMD in which the first or the last round of the RIPEMD compression function is omitted was described by Hans Dobbertin, who also published in 1998 a cryptanalysis of MD4. In this paper, we present a method for finding collisions in each of the parallel lines of RIPEMD. The collision search procedure requires only a few seconds computing time. We show that although the modifications of the MD4 compression function Used in RIPEMD introduce additional constraints in the cryptanalysis as Compared with Dobbertin's attack of MD4, these modifications do not result in an increase of the collision search computation time. It is still an open question whether collisions can be found for the full RIPEMD function.

1 Introduction

A collision resistant hash function can be informally defined as an easy to compute but hard to invert function which maps a message of arbitrary length into a fixed length (m-bit) hash value, and satisfies the property that finding a collision, i.e. two messages with the same hash value, is computationally unfeasible. The best collision resistance one can hope to achieve with an m-bit function is bounded above by the $O(2^{m/2})$ complexity of a birthday attack. For most currently used hash functions, $m = 128$ or $m = 160$. Collision resistant hash functions represent a quite useful primitive in cryptography and are of frequent use for the purposes of message pre-processing in digital signature, commitment in public key authentication schemes, etc.

Most collision resistant hash function candidates proposed so far are based upon the iterated use of a so-called compression function, which maps a fixed length ($m + n$-bit) input value into a shorter fixed length m-bit output value. First padding data (which include filling bits and sometimes information such as the message length) is appended to the M message to be hashed as to obtain a padded message which length is a multiple of n, and then split into n-bit blocks M_1 to M_k. Denote the compression function by f and the hash function by h. The m-bit hash value $h(M)$ is computed using the recurrence: $H_0 = IV0$ (where

M. Matsui (Ed.): FSE 2001, LNCS 2355, pp. 52–65, 2002.

IV0 is an m-bit constant initial value); for i=1 to k $H_i = f(H_{i-1}||M_i)$; $h(M) = H_k$. We are using in the sequel a terminology introduced by H. Dobbertin to distinguish two kinds of situations where two distinct (IV,M) and (IV',M') inputs have to the same image by the compression function of an iterated hash function: we will restrict the use of the term collision to the case where in addition IV=IV' and use the term pseudo collision otherwise. It was independently shown by Merkle and Damgard that if a compression function f is collision and pseudo collision resistant, then under some simple additional conditions on the padding rule, the associated hash function is collision resistant.

The most commonly used and (for the most recent ones) most trusted existing collision resistant hash functions do all belong to the MD-family of iterated hash functions, which includes MD4 (a 128-bit hash function proposed in 1990 by R. Rivest [9]), MD5 (a more conservative 128-bit hash function proposed in 1991 by R. Rivest [10]), RIPEMD (a 128-bit function which mixes the outputs of two parallel improved variants of MD4, proposed in 1995 by the European RIPE consortium [8]), RIPEMD-128 and RIPEMD-160 (128-bit and 160-bit variants of RIPEMD [6]), SHA (a 160-bit hash function proposed in 1992 by NIST), and SHA-1 (which was proposed by NIST as an improvement and replacement to the initial SHA [7] in 1994).

The hash functions of the MD family and the associated compression functions of MD family have been submitted to an extensive cryptanalytic investigation during the past years. Some collisions attacks were discovered on two of the three rounds of the MD4 compression function: an attack on the two last rounds by den Boer and Bosselaers [2] and an attack on the two first rounds leading to almost collisions on the full MD4, by Vaudenay [11]... This provided initial arguments in favor of moving from MD4 to MD5. Later on, Dobbertin established three major results on the cryptanalysis of the MD family of hash functions, namely (1) collisions for the full MD4 compression and hash functions that led to recommending the abandonment of its use (2) collisions for both the first and the last two rounds of the compression function of RIPEMD and (3) pseudo collisions on the whole MD5 compression [4], which do not lead to a collision of the associated MD5 hash function, but nevertheless invalidate the applicability of the Merkle-Damgard construction. Finally, Joux and Chabaud [1] discovered an attack allowing to find collisions for SHA in approximately 2^{61} SHA computations (instead of the about 2^{80} computations an ideal 160-bit hash function would require). As a consequence of these analyses, MD4 and SHA are no longer recommended, RIPEMD-128 seems to be a more conservative 128-bit hash function than MD5 and RIPEMD, and RIPEMD-160 and SHA-1 seem to be far from reach of known attack methods. In this paper we present a new result on the cryptanalysis of RIPEMD, which is to a certain extent complementary of the attack on the two-round version of RIPEMD described in [5]: we show how to construct collisions for the RIPEMD^L and RIPEMD^R three-rounds improved variants of MD4 used in the two parallel lines of computations of the RIPEMD compression function (in the left line and the right line respectively). The overall structure of our attack on RIPEMD^L and RIPEMD^R is close to the one in Dob-

bertin's attack on MD4 [3]. However, some of the MD4 modifications introduced in RIPEMD^L and RIPEMD^R prevent against too direct transpositions of the MD4 attack, and do substantially complicate the construction of collisions, so that the key steps of both attacks are quite different. The rest of this paper is organized as follows. Section 2 briefly describes the RIPEMD, RIPEMD^L and RIPEMD^R compression functions (a full description of which is provided in appendix), and introduces some basic techniques applicable to the cryptanalysis of hash functions of the MD family. Section 3 provides an overview of our attack strategy and of the main steps of the attack, which are later on detailed in Sections 5, 6 and 7. Section 8 concludes the paper.

2 Preliminaries

2.1 Notation

Throughout this paper, all the operations or functions considered relate to 32-bit words. The + symbol represents a modulo 2^{32} addition, the $\oplus$, $\wedge$, and $\vee$ symbols represent the bitwise exclusive OR, bitwise AND and bitwise OR respectively. If X is a 32-bit word and $s \in [0..31]$ then $X^{<<s}$ denotes the left cyclic shift of X by s bit positions to the left.

2.2 Description of the RIPEMD, RIPEMD^L, and RIPEMD^R Compression Functions

The RIPEMD compression function transforms a 4-word (128-bit) initial state value $IV = (IV_A, IV_B, IV_C, IV_D)$ and a 16-word message block $X = (X_0, .., X_{15})$ into a 128-bit output value (AA, BB, CC, DD). As said before, it consists of two parallel lines of computation achieving two variants of the MD4 compression function, namely the RIPEMD^L and RIPEMD^R three round compression functions.

The RIPEMD^L (resp RIPEMD^R) register values (A, B, C, D) (resp $(A'B'C'D')$) are initially set to the IV value. Each of the three rounds of RIPEMD^L (resp RIPEMD^R) consists of 16 steps. Each step consists of a transformation of the form

$$A = (A + f_r(B, C, D) + X_{\varphi(i \bmod 16)} + K_r)^{<<s_i}$$

$$A' = (A' + f_r(B', C', D') + X_{\varphi(i \bmod 16)} + K'_r)^{<<s_i}$$

where $i \in [0..47]$ denotes the step number, $r \in [0..2]$ denotes the round number, f_r is a round dependent 32-bit bitwise boolean function of three inputs (namely the majority function in the first round, the multiplexing function in the second round and the exclusive or in the third round; φ is a round-dependent permutation of i mod 16, (namely the identity permutation of in the first round and two other permutations denoted by σ and ρ in the two other rounds), so that each message word is involved in exactly one step of each round; s_i is a rotation amount which depends upon the step number and K_r and K'_r are round

dependent constants: they represent the single element which causes RIPEMD^L and RIPEMD^R to differ. Finally the RIPEMD (resp RIPEMD^L and RIPEMD^R) outputs are deduced from the (A, B, C, D) and (A', B', C', D') states obtained at the completion of step 47.

A detailed description of the above compression functions is provided in appendix.

The problem of finding an (X, X^*) pair of colliding messages for a compression function of the MD family such as say RIPEMD^L can be restated in terms of controlling the $(\Delta A, \Delta B, \Delta C, \Delta D) = (A, B, C, D) - (A^*, B^*, C^*, D^*)$ differences between the register values induced by X and X^* at the various steps of the computation.

3 Outline of Our Attack

In this Section we provide an outline of the overall structure of our attack on the RIPEMD^L and RIPEMD^R compression functions. Our aim is to construct a collision of the form $f(IV0, X) = f(IVO, X^*)$ where $IV0$ is defined as the proposed initial value for RIPEMD [8] and $X = (X_i), i = 0..15$ and $X^* = X_i^*, i = 0..15$ are two 16-word messages. We further require that only one X word X_{i_0} be distinct from the corresponding X^* word $X_{i_0}^*$, and that the $\Delta X_{i_0} = X_{i_0} - X_{i_0}^*$ difference be of weight 1 (in other words there exists an integer $k \in [0..31]$, such that $\Delta X_{i_0} = 1^{<<k}$).

3.1 Attack Structure

There are three occurrences of the X_{i_0} (resp $X_{i_0}^*$) word in the f computation, namely at steps i_0 , $16 + \rho(i_0 \bmod 16)$, $32 + \sigma(i_0 \bmod 16)$, which respectively belong to the first, the second and the third round[1] of f [table 1].

Due to the fact that each step represents a one to one transformation of the (A, B, C, D) register, the collision must necessarily happen at occurrence 3 of X_{i_0}.

Our overall attack strategy is similar (at least at the level of detail considered in this Section) to the one used in Dobbertin's MD4 attack [3]. It consists in trying to enforce a suitably selected "almost collision", i.e. a suitably chosen low weight difference value, just after occurrence 2 of the X_{i_0} (resp $X_{i_0}^*$) message words, in such a way that with sufficiently high probability this almost collision results, just before occurrence 3, in an intermediate difference value which is compensated during that step by the ΔX_{i_0} message difference. The collision is then preserved between both computations after occurrence 3.

A little bit more in detail, our attack consists (once a suitable choice of i_0 has been found) of three main parts, which purpose is to efficiently generate allocations of the 16 X_i words in such a way that the probability of obtaining a full collision for one of these allocations is high.

[1] ρ and σ are defined at the end of the article.

Table 1. The three main parts of the attack.

IV0
Part 3: (backward adjustment)
Occurrence 1 of $X_{i_0}, X^*_{i_0}$ (step $i_0 = 10$)
Part 2: (inner almost-collisions search)
Occurrence 2 of $X_{i_0}, X^*_{i_0}$ (step $16 + \rho(i_0 \bmod 16) = 20$)
Part 1: (differential part)
Occurrence 3 of $X_{i_0}, X^*_{i_0}$ (step $32 + \sigma(i_0 \bmod 16) = 33$)

(1) Differential part [occurrences 2 to 3]

This preliminary part of the attack consists in finding a suitable Δ low weight intermediate difference value such that if $(\Delta A, \Delta B, \Delta C, \Delta D) = \Delta$ after occurrence 2 of the ΔX_{i_0} message difference, then with a sufficiently high probability $(\Delta A, \Delta B, \Delta C, \Delta D) = 0$ after occurrence 3. It is based upon quite simple differential cryptanalysis techniques.

(2) Almost collision search [occurrences 1 to 2]

This is the most complex part of the attack. It consists of finding an allocation of the $A_{i_0}, B_{i_0}, C_{i_0}, D_{i_0}$ registers value at the end of step i_0 and allocations of the X_i words corresponding to those i values involved in steps i_0 (occurrence 1) to $16 + \rho(i_0 \bmod 16)$ (occurrence 2) which together ensure that an almost collision occurs at occurrence 2, i.e. that the difference value on the registers A to D is equal to the Δ low weight difference found in part (1).

(3) Backward adjustment [beginning to occurrence 1]

This part of the attack consists of finding allocations of the X_i values not fixed in part (2) such that initial value IV0 results in the $A_{i_0}, B_{i_0}, C_{i_0}, D_{i_0}$ register values at the completion of occurrence 1 (step i_0).

The actual collision search procedure consists of first achieving the precomputations of parts (1) and parts (2) above, which determine a first subset of the X_i allocations, and then of using part (3) to efficiently generate many allocations of the other X_i values leading to an almost collision after occurrence 2, until eventually one of these almost collisions provides a full collision after occurrence 3.

3.2 Selecting an Appropriate i_0 Value: Reasons for the $i_0 = 10$ Choice

The selected i_0 value must achieve the best possible trade-off between the requirements inherent to the different parts of the attack, which can be summarized as follows:

- Part 1 (differential part): the number of steps between occurrences 2 and 3 of X_{i_0} must not be too large in order for the probability of the differential characteristic used in part 1 to be large enough. More importantly, the $\sigma(i_0 \bmod)$ number of steps where the H function (exclusive OR) is used must be extremely low, because otherwise the very fast diffusion achieved by H would result in unacceptably small collision probabilities.

- Part 2 (almost collisions search): the $16+\rho(i_0 \bmod 16)-i_0$ number of steps between occurrences 1 and 2 of X_{i_0} must be large enough (say at least 6), but not too large in order to leave some free variables for part 3 of the attack, and if possible not involve a not too large fraction of F functions (multiplexing), which are harder to exploit than the G function (majority).

- Part 3 (backward adjustment): as said before, the number of X_i variables left free after part 2 must be sufficient. Due to the modifications introduced in the RIPEMD ρ and σ permutations as compared with those involved MD4, it is impossible to find any i_0 value which is as suitable for an attack as the $i_0 = 12$ value used in the attack of MD4.

The part 1 requirement on a low $\sigma(i_0 \bmod 16)$ number of steps involving the H functions between occurrences 2 and 3 is very demanding, and leads to discarding candidate index value such as $i_0 = 15$. The $i_0 = 13$ index value had also to be discarded because the number of steps between occurrences 1 and 2 of X_{i_0} is only 5, a too low value. The index value $i_0 = 10$ appeared to realize the best trade-off between the various requirements: the number of steps between the end of occurrence 2 (step 21) and the end of occurrence 3 (step 34) is 13, a rather low value (note that the corresponding number of steps in the MD4 attack is 15), and more importantly only 3 of these 13 steps involve the H function; finally, the number of steps between occurrence 1 (step 10) and occurrence 2 (step 20) is acceptable (11 steps, which involve the 9 X_i input words 1, 4, 7, and 10 to 15).

The difference between X_{10} and X^*_{10} is chosen equal to the low Hamming weight value $1^{<<6}$. The attack aims at finding a collision for two collections of words X and X^* defined by setting:

$$X^*_i = X_i \text{ for } i \neq 10,\ X^*_{10} = X_{10} + 1^{<<6}$$

Each of the 3 parts ends with an occurrence of X_{10}. The first occurrence of X_{10} introduces differences in the computation of the collision. This differences in the registers are compensated by the difference of the third occurrence of X_{10} at step 33 and are equal to zero. So between the first and the second occurrences of the word 10, the differences have to be controlled to allow the compensation of the variations between steps 21 and 33.

To achieve this, a well-chosen value for the differences at step 20, $\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6})$, is required.

4 Differential Attack Modulo 2^{32} (Steps 21-33).

The Δ_{20} differences value just before step 21 must result after step 32 in an intermediate difference value compensated at step 33 by the $\Delta X_{10} = 1^{<<6}$ difference. This part of the attack is a routine differential attack.

The variations on the register occurring at any computation step i come from the difference on the inputs X_i and X_i^*, the direct diffusion of the changing register and the indirect diffusion from the boolean vector function. The difference on the inputs occurs only when X_{10} is used in the step (steps 10, 20 and 33). The direct diffusion cannot be suppressed whereas the differences due to indirect diffusions have a non zero probability to appear for the multiplexing (F) and Majority (G) functions. Thus, with a certain probability, the indirect diffusions are equal to zero in those steps belonging to the second round of the compression function (21-31) and compensate the variations during the third round steps (31-33).

The probabilities can be computed backward from step 33 to step 21. We obtain at the beginning of step 21, a $\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6})$ difference leading to no difference at the end of step 33, $\Delta_{33} = (0, 0, 0, 0)$. Table 2 gives for step i, the probability for Δ_i to be equal to the values in the table under the assumption that Δ_{i-1} does the same.

Table 2. Differential attack.

step	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33
A				$1^{<<24}$				$1^{<<31}$				$1^{<<6}$				0	
B			0				0				0				0		
C		0				0				0				0			
D	$-1^{<<6}$				$-1^{<<15}$				$-1^{<<27}$				$-1^{<<6}$				0
P	1	1	1	1	1	1/9	1/9	1/3	1/3	1/9	1/9	1/3	1/3	1/9	1/9	1/3	1

The probabilities in table 2 are an approximation calculated without taking the correlations between the lines into account. Each step is strongly dependent on the 3 previous steps, but an exact computation of the probability would require lengthy considerations.

The differential attack cannot be achieved unless an initial constraint is satisfied. Actually, in an inner almost-collision, the registers are given fixed values. So registers from 18 to 20 are known and their values fix the probability of a computation with no indirect diffusion at step 21, to 0 or 1. A constraint is added. We reflect this additional condition in saying that in order for an inner almost-collision to be admissible, the following equations must be satisfied:

$$G(A_{20}, B_{19}, C_{18}) = G(A_{20}^*, B_{19}^*, C_{18}^*) \quad (1)$$

$$\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6}) \quad (2)$$

Suppose we have an admissible inner-collision. Then the probability[2] to achieve the differential attack is smaller than but close to $2^{-26.95}$.

5 Backward Adjustment (Steps 0 to 9)

Suppose we have found an admissible inner almost-collision. The method used to find an inner almost-collision corresponds to the resolution of a system where the parameters are the 32-bit registers used at steps 8 to 20 and the variation of the input word X_{10}. Solving the system fixes the contents of the registers R_8 to R_{20}. So using the equations of steps 12 to 20, we obtain the values of words 1, 4, 7, 10, 12, 13, 14 and 15.

The input word X_{11} appears in step 11. But a solution of the almost collision does not fix B_7 which X_{11} depends upon. So it does not fix X_{11}. Nevertheless, since the word X_{10} is fixed, B_7 and C_6 are largely linked and X_{11} cannot be considered as a free variable:

$$C_6 = C_{10}^{<<18} - X_{10} - F(D_9, A_8, B_7) - K_0 \tag{3}$$

There are still 7 free variables left for the rest of the attack. This is 2 variables less than in the cryptanalysis of MD4. But since the differential part of the attack requires numerous trials, it must be possible to take arbitrary values for some of the input words. For this, we randomly select X_0, X_2 and X_3. Since X_1 and X_4 are fixed by the admissible inner almost-collision found before, we can compute steps 0 to 4 to obtain the register values 0 to 4. Thus the backward adjustment problem consists in finding the remaining free variables, namely D_5, X_5, X_6, X_8 and X_9.

Let's fix $C_6 = -1$, set B_7 and compute X_{11}. The equations of step 7 and 10 become:

$$D_5 = B_7^{<<23} - X_7 - B_3 - K_0 \tag{4}$$

$$F(D_9, A_8, B_7) = C_{10}^{<<18} - X_{10} - C_6 - K_0 \tag{5}$$

Due to the involvement of the F boolean function (multiplexing) in the above equation , B_7 is not easy to compute after the inner almost-collision search. So, the equation (5) is handled as a new constraint for the resolution of the admissible inner almost-collision search system. It must be satisfied before the beginning of the computation for the right initial value.

Finally we know D_5 from equation (4) above, i.e. we know all registers from 0 to 20. At steps 5, 6, 8 and 9, the values of X_5, X_6, X_8 and X_9 compensate the contents of the registers.

This means we can reach the connection to a given inner almost-collision from a given initial value and keep 3 words free for the differential attack if the given inner almost-collision satisfies (5).

[2] This probability has to be compared with $2^{-30.11}$, the probability found in the cryptanalysis of MD4.

Explicitly, the prescribed initial value is matched by the settings:

$$A_4 = (A_0 + X_4 - F(B_3, C_2, D_1) + K_0)^{<<5} \tag{6}$$
$$C_6 = 0xffffffff \tag{7}$$
$$D_5 = B_7^{<<23} - B_3 - X_7 - K_0 \tag{8}$$
$$X_5 = D_5^{<<24} - D_1 - F(A_4, B_3, C_2) - K_0 \tag{9}$$
$$X_6 = C_6^{<<25} - C_2 - F(D_5, A_4, B_3) - K_0 \tag{10}$$
$$X_8 = A_8^{<<21} - A_4 - F(B_7, C_6, D_5) - K_0 \tag{11}$$
$$X_9 = D_9^{<<19} - D_5 - F(A_8, B_7, C_6) - K_0 \tag{12}$$

6 Inner Almost-Collisions

6.1 Steps 10 to 20: A System with Constraints

In this Section, we consider the compression function between the first two occurrences of X_{10} and X_{10}^* (10-20) and search an inner almost-collision between steps 10 and 20. The inner almost-collision has to be admissible and to satisfy the constraints resulting from the backward adjustment and differential parts of the attack:

$$C_6 = -1 \tag{13}$$
$$F(D_9, A_8, B_7) = C_{10}^{<<18} - X_{10} + 1 - K_0 \tag{14}$$
$$G(A_{20}, B_{19}, C_{18}) = G(A_{20}^*, B_{19}^*, C_{18}^*) \tag{15}$$
$$\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6}) \tag{16}$$

Thus, finding an inner almost-collision can be expressed as solving a system of equations under these four Constraints[3]. For each step from 10 to 20, an equation involving only the ΔX_j difference values, not the X_j values themselves, is provided by eliminating the value of the words.

$$X_j = a_{j+1}^{<<32-s_j} - (a_j + f_i(b_j, c_j, d_j) + K_j) \tag{17}$$
$$X_j^* = a_{j+1}^{*<<32-s_j} - (a_j^* + f_i(b_j^*, c_j^*, d_j^*) + K_j) \tag{18}$$
$$\Longrightarrow \Delta X_j = \Delta a_{j+1}^{<<32-s_j} - \Delta a_j - \Delta f_i(b_j, c_j, d_j) \tag{19}$$

Adding the equation (15) and using equation (16), we obtain a system of 12 (19)-like equations.

The resolution of this system fixes the contents of the registers from 8 to 20. Once the register values are known, the values of the words 10 to 15, 1, 4 and 7 can be deduced, using the (17)-like equations.

As the words X_{10} and X_{13} appear twice between steps 10 and 20, the X_{10} (resp X_{13}) values provided by the equations of steps 10 and 20 (resp 13 and 18)

[3] there were only two constraints in MD4 cryptanalysis.

must be equal. Thus two constraints must be added to the system in order for the register values to provide consistent X_{10} and X_{13} values. be added to

Therefore, an admissible inner almost-collision can be found by solving the system under the additional two constraints:

$$A_{20}^{<<21} - (A_{16} + G(B_{19}, C_{18}, D_{17}) + K_{20}) = C_{10}^{<<18} - (G(D_9, A_8, B_7) + K_{10} - 1)$$
$$D_{13}^{<<25} - (D_9 + F(A_{12}, B_{11}, C_{10}) + K_{13}) = C_{18}^{<<24} - (C_{14} + G(D_{17}, A_{16}, B_{15}) + K_{18})$$

6.2 Resolution

The resolution of the system and constraints defined above aims at finding the suitable values for the contents of the registers from B_7 to A_{20} and differences on registers R_{11} to R_{20}. The registers after 21 are computed by choosing randomly X_0, X_2 and X_3, whereas the registers before 6 are fixed by the backward adjustment as described previously.

We have to control the differences in the registers between the two first occurrences of the word 10, from $\Delta_9 = (0, 0, 0, 0)$ to $\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6})$. There are two ways to make the choices on the registers. On the one hand, the content of a register can be fixed to simplify the system, this specialization of the system reduces strongly the choices on the content of the register. On the other hand, at first, only the differences in the registers are chosen as to ensure consistency in the diffusions and afterwards the registers are given values. This leads to more possible values for a register.

In the resolution of the system, after suitable "specializations", solutions can be found by iterative solving of equations of the form:

$$G(Z + dZ, a_1, b_1) - G(Z, a_2, b_2) = C$$

where the Z unknown is a 32-bit word, G is a derived from a ternary boolean function and dZ, $a1$, $b1$, $a2$, $b2$ and C are known words. Equations are solved recursively and bitwise from the lowest bit to the highest.

Let's continue the backward computation of the differences begun in the differential part of the attack. We obtain the differences for registers from R_{13} to R_{20} [table 3] and simplify the system.

Table 3. Diffusion.

step		13	14	15	16	17	18	19	20
A	specialization				$1^{<<13}$				$1^{<<24}$
B	and			0				0	0
C	resolution		0				0		0
D		-1				$-1^{<<6}$			$-1^{<<6}$

Since the first equations of the system involve the multiplexing function F, the indirect diffusions are far more difficult to control. We specialize this part of the system by choosing $D_{13} = -1$ and $D_{13}^* = 0$.

Without the two additional constraints, the system can be rewritten:

$$D_{13} = -1 \ , \ D_{13}^{*} = 0 \tag{20}$$
$$\Delta C_{10}^{<<18} = 1^{<<6} \tag{21}$$
$$\Delta F(C_{10}, D_9, A_8) = \Delta B_{11}^{<<17} \tag{22}$$
$$\Delta F(B_{11}, C_{10}, D_9) = \Delta A_{12}^{<<26} \tag{23}$$
$$\Delta F(A_{12}, B_{11}, C_{10}) = -1 \tag{24}$$
$$A_{12}^{*} = B_{11} + \Delta C_{10} \tag{25}$$
$$\Delta F(C_{14}, D_{13}, A_{12}) = -\Delta B_{11} \tag{26}$$
$$B_{15} = C_{14} \oplus (1^{<<31} - \Delta A_{12}) \tag{27}$$
$$\Delta G(A_{16}, B_{15}, C_{14}) = 0 \tag{28}$$
$$\Delta G(D_{17}, A_{16}, B_{15}) = 0 \tag{29}$$
$$\Delta G(C_{18}, D_{17}, A_{16}) = 0 \tag{30}$$
$$\Delta G(B_{19}, C_{18}, D_{17}) = -1^{<<6} \tag{31}$$
$$\Delta G(A_{20}, B_{19}, C_{18}) = 0 \tag{32}$$

The end of the system (from equation (28)) can be solved as it is done in MD4 cryptanalysis: except in equation 31 (step 20), the diffusions are direct as it is in the differential attack and at step 20, the diffusion from the input are compensated by the indirect diffusion[4], whereas the direct diffusion introduces the variations on registers R_{20} required for the rest of the complete attack.

Only the variations for the registers R_{10}, R_{11} and R_{12} are still free. Nevertheless, we know $\Delta C_{10}^{<<18} = 1^{<<6}$. By choosing C_{10} and C_{10}^{*} to maximize the Hamming weight of the ΔC_{10} difference, we obtain $\Delta C_{10} = 1^{<<20} - 1^{<<14} + 1$ and maximize the possibilities of solving the rest of the system.

7 Collision Search Algorithm

We can now describe the collision search algorithm for RIPEMD^L.

There is a practical algorithm which requires less than 1 second and allows us to compute an inner almost-collision that satisfies the four equations:

$$C_6 = -1$$
$$F(D_9, A_8, B_7) = C_{10}^{<<18} - X_{10} + 1 - K_{10}$$
$$\Delta_{20} = (1^{<<24}, 0, 0, -1^{<<6})$$
$$\Delta G(A_{20}, B_{19}, C_{18}) = 0$$

[4] We can notice that since registers R_{17} has the same variation as words 10, the majority boolean function leads to a high probability to have a solution for register R_{19}.

1. Choose C_{10} and C^*_{10} that satisfy $\Delta C_{10} = 1^{<<20} - 1^{<<14} + 1$.
2. Choose A_{12}, ΔA_{12} and ΔB_{11}. Verify $\Delta F(A_{12}, B_{11}, C_{10}) = -1$.
3. Compute registers from 9 to 18 and verify the first constraint to obtain the value of X_{13}.
4. Compute registers R_7, R_8, R_{19} and R_{20}. Then verify the second constraint to obtain the value of X_{10}.
5. With the contents of the registers, set the value for X_{11}, X_{12}, X_{14}, X_{15}, X_1, X_4 and X_7. We obtain an admissible inner almost-collision.
6. Choose randomly X_0, X_2 and X_3.
7. Compute the registers for 0 to 5 as described in section 5. We reach the right initial value.
8. Achieve the differential part of the attack by making new trial for X_0, X_2 and X_3 until we reach a collision $\Delta_{33} = (0, 0, 0, 0)$.

With RIPEMD^L, we obtained a collision for the (X, X^*) pair determined by the following values:

$X_0 = 0x10978d07$	$X^*_0 = 0x10978d07$	$X_8 = 0xd0f6edc3$	$X^*_8 = 0xd0f6edc3$
$X_1 = 0x937e0e67$	$X^*_1 = 0x937e0e67$	$X_9 = 0xd22c0042$	$X^*_9 = 0xd22c0042$
$X_2 = 0x697a5fe9$	$X^*_2 = 0x697a5fe9$	$X_{10} = 0x847dc027$	$X^*_{10} = 0x847dc067$
$X_3 = 0x313d55b0$	$X^*_3 = 0x313d55b0$	$X_{11} = 0x84fbc027$	$X^*_{11} = 0x84fbc027$
$X_4 = 0xa816066b$	$X^*_4 = 0xa816066b$	$X_{12} = 0xfc7f4140$	$X^*_{12} = 0xfc7f4140$
$X_5 = 0xed3f60d6$	$X^*_5 = 0xed3f60d6$	$X_{13} = 0xfff00001$	$X^*_{13} = 0xfff00001$
$X_6 = 0xc6c1e1a$	$X^*_6 = 0xc6c1e1a$	$X_{14} = 0x7fff82$	$X^*_{14} = 0x7fff82$
$X_7 = 0xa58dc669$	$X^*_7 = 0xa58dc669$	$X_{15} = 0xffbf43$	$X^*_{15} = 0xffbf43$

With RIPEMD^R, we obtained a collision for the (X, X^*) pair determined by the following values:

$X_0 = 0x3e00cc54$	$X^*_0 = 0x3e00cc54$	$X_8 = 0xa32b5a9c$	$X^*_8 = 0xa32b5a9c$
$X_1 = 0xc6008000$	$X^*_1 = 0xc6008000$	$X_9 = 0xd6e17c84$	$X^*_9 = 0xd6e17c84$
$X_2 = 0x4f269ad0$	$X^*_2 = 0x4f269ad0$	$X_{10} = 0xc0003440$	$X^*_{10} = 0xc0003480$
$X_3 = 0x398358a6$	$X^*_3 = 0x398358a6$	$X_{11} = 0xc07e3440$	$X^*_{11} = 0xc07e3440$
$X_4 = 0x1448002$	$X^*_4 = 0x1448002$	$X_{12} = 0xabdcb55a$	$X^*_{12} = 0xabdcb55a$
$X_5 = 0xcc41f9d9$	$X^*_5 = 0xcc41f9d9$	$X_{13} = 0xaf4d741b$	$X^*_{13} = 0xaf4d741b$
$X_6 = 0xb0304fed$	$X^*_6 = 0xb0304fed$	$X_{14} = 0xafdd739c$	$X^*_{14} = 0xafdd739c$
$X_7 = 0x104002$	$X^*_7 = 0x104002$	$X_{15} = 0xb05d335d$	$X^*_{15} = 0xb05d335d$

8 Conclusion

It turns out that the modifications of the MD4 compression function introduced in each of the RIPEMD^L and RIPEMD^R compression functions of RIPEMD leads to more constraints in the cryptanalysis as compared with the Dobbertin's cryptanalysis attack of MD4, but collisions can still be found easily (smaller than $2^{-26.95}$ for the differential part of the attack). Our attack provides some arguments in favor of the conjecture that the existence of an attack on MD4 is

not due to a suboptimal selection of parameters such as the σ and ρ, the rotation amounts, etc. The selection made in RIPEMD does not lead to a stronger version of MD4.

Collisions could be found for each line of the RIPEMD compression function and for the two-round versions of RIPEMD described in [5]. The two methods used have not lead yet to a successful attack on the full compression function and RIPEMD is still holding up.

9 Appendix

Description of the RIPEMD RIPEMD^L and RIPEMD^R Compression and Hash Functions

Define the ρ and σ permutations of [0..15] give by Table 4 hereafter

Table 4. Permutations for rounds 2 and 3

Word	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
ρ	9	3	13	7	1	11	5	0	15	10	4	14	8	2	12	6
σ	10	7	2	0	3	14	11	9	6	4	1	12	12	13	8	5

Define register R_k ($0 \leq k \leq 47$) as the 32-bit register value computed at step k. R_k is also denoted A_k when $k = 0 \bmod 4$, D_k when $k = 1 \bmod 4$, C_k when $k = 2 \bmod 4$ and B_k when $k = 3 \bmod 4$ as far as the left line is concerned. The same definitions apply for the right line and for R'_k, A'_k, D'_k, C'_k and B'_k.

Start with the initial value $IV = (IV_A, IV_B, IV_C, IV_D)$. For the first iteration, start with $IV = IV0 = (IV_A, IV_B, IV_C, IV_D)$, where $IV_A = 0x67452301$; $IV_B = 0xefcdab89$; $IV_C = 0x98badcfe$; $IV_D = 0x10325476$.

Copy IV_A, IV_B, IV_C, IV_D into the registers A, B, C, D for the left line and $A'B'C'D'$ for the right line. These registers correspond to steps -4, -3, -2 and -1.

Define the constants $K_0 = 0x0$, $K_1 = 0x5a827999$, $K_2 = 0x6ed9eba1$ for the left line and $K'_0 = 0x50a28be6$, $K'_1 = 0x0$, $K'_2 = 0x5c4dd124$ for the right line.

Define $F(x, y, z) = (x \wedge y) \vee ((\neg x) \wedge z)$, $G(x, y, z) = (x \wedge y) \vee (x \wedge z) \vee (z \wedge y)$ and $H(x, y, z) = (x \oplus y \oplus z)$ functions used in rounds 1, 2 and 3 respectively.

Define the word X and the shift for each step given by Table 5 hereafter.

Compute for ($i = 0$; $i < 48$;$i = i + 1$):

$$R_i = (R_{i-4} + f_r(R_{i-1}, R_{i-2}, R_{i-3}) + X_{\varphi(i \bmod 16)} + K_0)^{<<s_i}$$
$$R'_i = (R'_{i-4} + f_r(R'_{i-1}, R'_{i-2}, R'_{i-3}) + X_{\varphi(i \bmod 16)} + K'_0)^{<<s_i}$$

Where $r = 0$ when $i \in [0..15]$, $r = 1$ when $i \in [16..31]$ and $r = 2$ when $i \in [32..47]$ and $f_0 = F$, $f_1 = G$ and $f_2 = H$.

Table 5. Rotations for each step

step	X	s	X	s	X	s	X	s	step	X	s	X	s	X	s	X	s	step	X	s	X	s	X	s	X	s
0 to 3	0	11	1	14	2	15	3	12	16 to 19	7	7	4	6	13	8	1	13	32 to 35	3	11	10	13	2	14	4	7
4 to 7	4	5	5	8	6	7	7	9	20 to 23	10	11	6	9	15	7	3	15	36 to 39	9	14	15	9	8	13	1	15
8 to 11	8	11	9	13	10	14	11	15	24 to 27	12	7	0	12	9	15	5	9	40 to 43	14	6	7	8	0	13	6	6
12 to 15	12	6	13	7	14	9	15	8	28 to 31	14	7	2	11	11	13	8	12	44 to 47	11	12	13	5	5	7	12	5

Finally compute the compression function output AA, BB, CC, DD as follows:

For the RIPEMD compression function: $AA = IVB + C + D'$,
$BB = IVC + D + A'$, $CC = IVD + A + B'$, $DD = IVA + B + C'$
For the RIPEMD^L compression function: $AA = IVA + A$,
$BB = IVB + B$, $CC = IVC + C$, $DD = IVD + D$.
For the RIPEMD^R compression function: $AA = IVA + A'$,
$BB = IVB + B'$, $CC = IVC + C'$, $DD = IVD + D'$.

References

1. F.Chabaud and A.Joux. Differential Collisions in SHA-0. extended abstract. In CRYPTO'98, LNCS 1462, pp 56–71, 1998.
2. B.den Boer and A.Bosselaers. An attack on the last two rounds of MD4. In Advances in Cryptology - Crypto'91 pages 194-203 LCNS 576 Springer-Verlag 1992.
3. H.Dobbertin. Cryptanalysis of MD4. In Journal of Cryptology vol.11 n.4 Autumn 1998.
4. H.Dobbertin. Cryptanalysis of MD5 Compress. Presented at the rump session of Eurocrypt '96, May 14, 1996.
5. H.Dobbertin. Ripemd with two round compress function is not collision-free. In Journal of Cryptology vol.10 n.1, winter 1997.
6. H.Dobbertin, A.Bosselaers and B.Preneel. RIPEMD-160: a strenghened version of RIPEMD. April 1996.
ftp.esat.kuleuven.ac.be/pub/COSIC/bossselae/ripemd.
7. National Institute of Standards and Technology (NIST) FIPS Publication 180-1: secure Hash Standard. April 1994.
8. RIPE. Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040). In LNCS 1007 Springer-Verlag 1995.
9. R.L.Rivest. The MD4 message digest algorithm. In Advances in Cryptology - Crypto'90 pages 303-311 Springer-Verlag 1991.
10. R.L.Rivest. RFC1321: The MD5 message digest algorithm. M.I.T. Laboratory for Computer Science and RSA Data Security, Inc., April 1992.
11. S.Vaudenay. On the need for multipermutations: Cryptanalysis of MD4 and SAFER. In FSE, LCNS 1008, pages 286-297 Springer-Verlag 1995.