J. Cryptology (1997) 10: 51–69

Journal of
CRYPTOLOGY

RIPEMD with Two-Round Compress Function Is Not Collision-Free

Hans Dobbertin
German Information Security Agency, P.O. Box 20 03 63,
D-53133 Bonn, Germany
dobbertin@skom.rhein.de

Communicated by Ivan B. Damgård

Received 27 March 1995 and revised 14 October 1995

Abstract. In 1990 Rivest introduced the cryptographic hash function MD4. The compress function of MD4 has three rounds. After partial attacks against MD4 were found, the stronger mode RIPEMD was designed as a European proposal in 1992 (RACE project). Its compress function consists of two parallel lines of modified versions of MD4-compress. RIPEMD is currently being considered to become an international standard (ISO/IEC Draft 10118-3). However, in this paper an attack against RIPEMD is described, which leads to comparable results with the previously known attacks against MD4: The reduced versions of RIPEMD, where the first or the last round of the compress function is omitted, are not collision-free. Moreover, it turns out that the methods developed in this note can be applied to find collisions for the full MD4.

Key words. Dedicated hash functions, RIPEMD, MD4, RACE project, ISO/IEC 10118-3.

1. Introduction and Summary

In 1990 Rivest [4] introduced the hash function MD4. The MD4 algorithm is defined as an iterative application of a three-round compress function.

In view of an attack against the last two rounds of the compress function of MD4, which was found by den Boer and Bosselaers [2], the stronger mode RIPEMD [1] was designed as a European proposal in 1992 (RACE project). The compress function of RIPEMD consists of two parallel lines of a modified version of the MD4 compress function.

In what follows we show that if the first or the last round of the RIPEMD compress function is omitted, then collisions can be found by an attack starting with a basically very simple idea.

Thus we describe an attack against RIPEMD leading to results which are comparable

with those previously known for MD4 (see [2] and [6]), although our attack requires more computational effort.

Any attack against RIPEMD has to overcome the problem that the two parallel lines of the compress algorithm have to be handled simultaneously. However, as we shall see this can be managed if we cancel the first or the last round. This weakness is caused mainly by the fact that in both lines the message blocks are applied in exactly the same ordering; while our attack in principle does not depend on the particular choices of Boolean functions, shifts, and constants (see the Appendix for the definition of RIPEMD).

Therefore, although RIPEMD is certainly stronger than MD4, in our opinion the intention of the design has not been reached. The effect of combining two parallel chains of a modified MD4-compress, as is realized in the present version of RIPEMD, is not as strong as should be expected when the computational effort is doubled.

It has turned out that the methods developed in this note can be applied to find collisions for the full MD4 (see Section 6). This attack is explained in [3].

It remains a challenging task trying to attack MD5, a strengthened version of MD4 due to Rivest [5], with the techniques presented here (see "Note Added in Proof", p. 68).

Terminology and Basic Notation. Using the term "collision of a compress function" we assume that the corresponding initial values coincide for both inputs. For "pseudo-collisions" this is not required. However, the latter are of much less practical importance and are not considered here.

Throughout, all occurring variables and constants are 32-bit quantities, and accordingly the value of an expression is its remainder modulo 2^{32}. The symbols $\wedge$, $\vee$, and $\oplus$ are used for bitwise AND, OR, and XOR, respectively. For a 32-bit word X, let $X^{\lll s}$ denote the 32-bit value obtained by circularly shifting (rotation) X left by s bit positions for $0 \leq s < 32$. To complete this definition set $X^{\lll(-s)} = X^{\lll(32-s)}$. If X is an expression then, of course, evaluate it before shifting.

2. Main Result and Plan of the Attack

By RIPEMD$^{[12]}$ (resp. RIPEMD$^{[23]}$) we denote the hash functions, which are obtained as reductions of RIPEMD by canceling the last (resp. first) round of the compress function. (See the Appendix for details.) We can state our main result as follows:

RIPEMD$^{[12]}$ *and* RIPEMD$^{[23]}$ *are not collision-free.*

Empirical observations have shown that the attack described below, which leads to collisions, requires an average of about the same computational effort as 2^{31} computations of a (two-round) compress function. Concretely this means that it takes on average about 1 day on a 486-PC (66 MHz) to compute collisions.

Our attack is separated into three parts:

Part I: Inner Collisions. The basic idea is to find collisions for compress$^{[12]}$ or compress$^{[23]}$ by taking two collections $X_i, \tilde{X}_i (i < 16)$ of words such that, for some i_0, we have $X_{i_0} \neq \tilde{X}_{i_0}$, but $X_i = \tilde{X}_i$ for $i \neq i_0$. When the compress values of the inputs X

and $\tilde{X}$ are computed, then in the first round everything coincides until we come to step i_0. At this point the computations for X and $\tilde{X}$ become different. However, in the second round, X_{i_0} (resp. $\tilde{X}_{i_0}$) has to be applied again at a certain step. After that, in the remaining steps, all inputs coincide again. Thus it is natural to try to control the computation in such a way that after the second application of X_{i_0} and $\tilde{X}_{i_0}$ the contents in the registers coincide.

Therefore we first concentrate on the steps between the first and second application of X_{i_0} (resp. $\tilde{X}_{i_0}$). In this part of the attack we try to find suitable values for the contents of the registers after step $i_0 - 1$ and suitable values for X_{i_0}, $\tilde{X}_{i_0}$, and for the $X_i = \tilde{X}_i$ occurring in the considered segment of the compress function.

Note that the ordering, in which the variables X_i are applied, is exactly the same for compress$^{[12]}$ and compress$^{[23]}$. In both cases we take $i_0 = 13$, since for this choice we get the shortest corresponding segment of the compress function. We have enough variables to find a simultaneous collision for the considered segment of the left and right line ("inner collision"), and, on the other hand, enough variables remain free for the second part of the attack.

Part II: Backward Collisions. Suppose we have found an inner collision. Then, in order to find a collision of the two-round compress function, we have to determine values for the remaining free variables such that computing backward and starting with the values of the registers after step $i_0 - 1$ (these values are given according to Part I) leads to the same initial value for the left and right line ("backward collision").

Experience has shown that we can even find collisions with 2^{100} of the 2^{128} possible initial values (see Section 4).

Part III: Right Initial Value ("Meet-in-the-Middle"). Finally, the problem remains that the initial value IV_0 specified by the definition of RIPEMD$^{[12]}$ and RIPEMD$^{[23]}$ (see the Appendix) will usually not be among the admissible initial values found in Part II. Therefore we randomly choose 16 input words for the compress function starting with IV_0, until the output is admissible. This approach works, since we have a very fast test whether the output is admissible or not (checking a small set of equations). Thus nothing has to be stored and compared in lists. The computational effort necessary for this part is usually much smaller than for Part II.

Since two messages found by this attack both have length $2 \times 16 \times$ (*length of words*), they give a collision no matter whether the padding rule is applied or not.

The next sections contain a detailed description of Parts I–III of our attack.

3. Inner Collisions

In this section we show how to find inner collisions for compress$^{[23]}$. For compress$^{[12]}$ this is much more difficult and requires many technical tricks. (The difficulties seem to be caused by the asymmetry of the Boolean function F, which is involved here.) We restrict ourselves to the easy case, and we only give an inner collision for compress$^{[12]}$ at the end of this section without proof.

By L-compress$^{[23]}$ and R-compress$^{[23]}$ we denote the left and right line of compress$^{[23]}$, respectively. We now consider the sequence of those steps of L-compress$^{[23]}$ and R-compress$^{[23]}$ between which X_{13} occurs the first and the second time, i.e., steps 13–18. In these steps, X_{13}, X_{14}, X_{15} together with the Boolean vector function

$$G(X, Y, Z) = (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z),$$

and then X_7, X_4, and again X_{13} together with the Boolean vector function

$$H(X, Y, Z) = X \oplus Y \oplus Z$$

are applied. The constants used in compress$^{[23]}$ are

$$\begin{aligned} K_1 &= \texttt{0x5a827999}, \\ K_2 &= \texttt{0x6ed9eba1}, \\ K_3 &= \texttt{0x5c4dd124}. \end{aligned}$$

Lemma A. *We obtain a simultaneous collision for steps* 13–18 *of* L-compress$^{[23]}$ *and* R-compress$^{[23]}$ *for*

$$\begin{aligned} X_{13} &= \texttt{0xa57d8666} = -K_1 - 1, \\ X_{14} &= \texttt{0xa57d8666} = -K_1 - 1, \\ X_{15} &= \texttt{0xa57d8666} = -K_1 - 1, \\ X_7 &= \texttt{0x9126145f} = -K_2, \\ X_4 &= \textit{arbitrary}, \end{aligned}$$

$$\begin{aligned} \tilde{X}_{13} &= X_{13} + 1 = \texttt{0xa57d8667}, \\ \tilde{X}_j &= X_j \quad (j = 14, 15, 7, 4), \end{aligned}$$

if we use the following "initial values" for step 13 *of the left and right line, respectively:*

$$\begin{aligned} (A_{\mathrm{L}}, B_{\mathrm{L}}, C_{\mathrm{L}}, D_{\mathrm{L}}) &= (1, 0, 0, 0), \\ (A_{\mathrm{R}}, B_{\mathrm{R}}, C_{\mathrm{R}}, D_{\mathrm{R}}) &= (Q, Q + 1, K_1 - Q, K_1 - Q - 1), \end{aligned}$$

where Q is even and satisfies the equation

$$(2Q - K_1)^{\ll 1} + (2Q - K_1)^{\ll 12} - Q + K_2 - K_3 + 1 = 0.$$

In fact, this equation has precisely the even solutions

$$\begin{aligned} Q &= 2 * \texttt{0x110d8f04}, \\ Q &= 2 * \texttt{0x5dda5bd1}. \end{aligned}$$

Proof. First we consider the left line. Table 1 shows the contents A_i, B_i, C_i, D_i of the registers of the left line after the application of steps $i = 13, \ldots, 18$ for the inputs $X_{13}, X_{14}, X_{15}, X_7, X_4$ as specified above, and for

$$(A_{12}, B_{12}, C_{12}, D_{12}) = (A_{\mathrm{L}}, B_{\mathrm{L}}, C_{\mathrm{L}}, D_{\mathrm{L}}) = (1, 0, 0, 0).$$

Table 1

Step i	A_i	B_i	C_i	D_i	Z_j	X_j	constant	Boolean function	Shift
12	1	0	0	0	*	*	*	*	*
13	1	0	0	$\boxed{-1}$	-1	$-K_1-1$	K_1	G	11
14	1	0	$\boxed{0}$	-1	-1	$-K_1-1$	K_1	G	13
15	1	$\boxed{0}$	0	-1	-1	$-K_1-1$	K_1	G	12
16	$\boxed{0}$	0	0	-1	0	$-K_2$	K_2	H	11
17	0	0	0	$\boxed{D_{17}}$	X_4+K_2	X_4	K_2	H	13
18	0	0	$\boxed{C_{18}}$	D_{17}	K_2-K_1-1	$-K_1-1$	K_2	H	14

Here we set

$$Z_j = X_j + \text{constant} \qquad (j = 13, 14, 15, 7, 4, 13)$$

for the "actual input," where constant denotes the constant of the particular step to be used according to the definition of the algorithm (see the Appendix). The boxed entries are those which have been modified in the respective steps.

We have

$$\begin{aligned} D_{18} &= D_{17} = (X_4 + K_2 - 1)^{\ll 13}, \\ C_{18} &= (D_{18} + K_2 - K_1 - 1)^{\ll 14}. \end{aligned}$$

In order to verify Table 1 compute

$$\begin{aligned} D_{13} &= (0 + G(1,0,0) - 1)^{\ll 11} = (-1)^{\ll 11} = (\texttt{0xffffffff})^{\ll 11} = -1, \\ C_{14} &= (0 + G(-1,1,0) - 1)^{\ll 13} = 0^{\ll 13} = 0, \end{aligned}$$

and so on.

We have to compare this with Table 2 for the inputs $\tilde{X}_{13}, \tilde{X}_{14}, \tilde{X}_{15}, \tilde{X}_7, \tilde{X}_4$, where $\tilde{X}_{13} = X_{13} + 1$ and $\tilde{X}_j = X_j$ $(j = 14, 15, 7, 4)$.

Table 2

Step i	$\tilde{A}_i$	$\tilde{B}_i$	$\tilde{C}_i$	$\tilde{D}_i$	Z_j	$\tilde{X}_j$	constant	Boolean function	Shift
12	1	0	0	0	*	*	*	*	*
13	1	0	0	$\boxed{0}$	0	$-K_1$	K_1	G	11
14	1	0	$\boxed{-1}$	0	-1	$-K_1-1$	K_1	G	13
15	1	$\boxed{0}$	-1	0	-1	$-K_1-1$	K_1	G	12
16	$\boxed{0}$	0	-1	0	0	$-K_2$	K_2	H	11
17	0	0	-1	$\boxed{\tilde{D}_{17}}$	X_4+K_2	X_4	K_2	H	13
18	0	0	$\boxed{\tilde{C}_{18}}$	$\tilde{D}_{17}$	K_2-K_1	$-K_1$	K_2	H	14

Table 3

Step i	A_i	B_i	C_i	D_i	Z_j	X_j	const.	Boolean function	Shift
12	Q	$Q+1$	K_1-Q	K_1-Q-1	*	*	*	*	*
13	Q	$Q+1$	K_1-Q	$\boxed{-1}$	$-K_1-1$	$-K_1-1$	0	G	11
14	Q	$Q+1$	$\boxed{0}$	-1	$-K_1-1$	$-K_1-1$	0	G	13
15	Q	$\boxed{B_{15}}$	0	-1	$-K_1-1$	$-K_1-1$	0	G	12
16	$\boxed{A_{16}}$	B_{15}	0	-1	K_3-K_2	$-K_2$	K_3	H	11
17	A_{16}	B_{15}	0	$\boxed{D_{17}}$	X_4+K_3	X_4	K_3	H	13
18	A_{16}	B_{15}	$\boxed{C_{18}}$	D_{17}	K_3-K_1-1	$-K_1-1$	K_3	H	14

We have

$$\tilde{D}_{18} = \tilde{D}_{17} = (X_4 + K_2 - 1)^{\ll 13},$$
$$\tilde{C}_{18} = (\tilde{D}_{18} + K_2 - K_1 - 1)^{\ll 14}.$$

Thus we see that $(A_{18}, B_{18}, C_{18}, D_{18}) = (\tilde{A}_{18}, \tilde{B}_{18}, \tilde{C}_{18}, \tilde{D}_{18})$ as desired.

For the right line, Tables 3 and 4 show the contents of the registers for steps 12–18.

For verification of Tables 3 and 4 we need the assumption that Q is even and we use the fact that K_1 is odd, for instance:

$$\begin{aligned} D_{13} &= (K_1 - Q - 1 + G(Q, Q+1, K_1 - Q) - K_1 - 1)^{\ll 11} \\ &= (-Q - 2 + ((Q \wedge (Q+1)) \vee (Q \wedge (K_1 - Q)) \vee ((Q+1) \wedge (K_1 - Q))))^{\ll 11} \\ &= (-Q - 2 + (Q+1))^{\ll 11} = (-1)^{\ll 11} = -1. \end{aligned}$$

Further, we have

$$\begin{aligned} B_{18} &= B_{15} = (2Q - K_1)^{\ll 12}, \\ A_{18} &= A_{16} = (Q - B_{18} - 1 + K_3 - K_2)^{\ll 11}, \\ D_{18} &= D_{17} = (X_4 + K_3 - 1 + (A_{18} \oplus B_{18}))^{\ll 13}, \\ C_{18} &= (K_3 - K_1 - 1 + (A_{18} \oplus B_{18} \oplus D_{18}))^{\ll 14}, \end{aligned}$$

Table 4

Step i	$\tilde{A}_i$	$\tilde{B}_i$	$\tilde{C}_i$	$\tilde{D}_i$	Z_j	$\tilde{X}_j$	const.	Boolean function	Shift
12	Q	$Q+1$	K_1-Q	K_1-Q-1	*	*	*	*	*
13	Q	$Q+1$	K_1-Q	$\boxed{0}$	$-K_1$	$-K_1$	0	G	11
14	Q	$Q+1$	$\boxed{-1}$	0	$-K_1-1$	$-K_1-1$	0	G	13
15	Q	$\boxed{\tilde{B}_{15}}$	-1	0	$-K_1-1$	$-K_1-1$	0	G	12
16	$\boxed{\tilde{A}_{16}}$	$\tilde{B}_{15}$	-1	0	K_3-K_2	$-K_2$	K_3	H	11
17	$\tilde{A}_{16}$	$\tilde{B}_{15}$	-1	$\boxed{\tilde{D}_{17}}$	X_4+K_3	X_4	K_3	H	13
18	$\tilde{A}_{16}$	$\tilde{B}_{15}$	$\boxed{\tilde{C}_{18}}$	$\tilde{D}_{17}$	K_3-K_1	$-K_1$	K_3	H	14

$$\begin{aligned}
\tilde{B}_{18} &= \tilde{B}_{15} = (2Q - K_1)^{\ll 12},\\
\tilde{A}_{18} &= \tilde{A}_{16} = (Q - \tilde{B}_{18} - 1 + K_3 - K_2)^{\ll 11},\\
\tilde{D}_{18} &= \tilde{D}_{17} = (X_4 + K_3 - 1 - (\tilde{A}_{18} \oplus \tilde{B}_{18}))^{\ll 13},\\
\tilde{C}_{18} &= (K_3 - K_1 - 1 + (\tilde{A}_{18} \oplus \tilde{B}_{18} \oplus \tilde{D}_{18}))^{\ll 14}.
\end{aligned}$$

Thus we conclude that $(A_{18}, B_{18}, C_{18}, D_{18}) = (\tilde{A}_{18}, \tilde{B}_{18}, \tilde{C}_{18}, \tilde{D}_{18})$ if and only if

$$A_{18} \oplus B_{18} = -(A_{18} \oplus B_{18}),$$

i.e., $A_{18} = B_{18}$ or $A_{18} = B_{18} \oplus 2^{31}$. The first case leads to the equation

$$(Q - (2Q - K_1)^{\ll 12} - 1 + K_3 - K_2)^{\ll 11} = (2Q - K_1)^{\ll 12},$$

or equivalently

$$(2Q - K_1)^{\ll 1} + (2Q - K_1)^{\ll 12} - Q + K_2 - K_3 + 1 = 0.$$

Simply testing all even $Q < 2^{32}$ shows that this equation has precisely the two solutions given in the lemma. It takes about 30 minutes on a 486-PC. (The corresponding equation for $A_{18} = B_{18} \oplus 2^{31}$ has no solution.) This completes the proof. □

Lemma B. *We obtain a simultaneous collision for steps* 13–18 *of* L-compress$^{[12]}$ *and* R-compress$^{[12]}$ *for*

$$\begin{aligned}
X_{13} &= \texttt{0xb6c474bc},\\
X_{14} &= \texttt{0x1e575831},\\
X_{15} &= \texttt{0x767f3bbb},\\
X_7 &= \texttt{0x3a456372},\\
X_4 &= \textit{arbitrary},
\end{aligned}$$

$$\begin{aligned}
\tilde{X}_{13} &= X_{13} + \Delta \qquad \textit{with} \quad \Delta = \texttt{0xa954a955},\\
\tilde{X}_j &= X_j \quad (j = 14, 15, 7, 4),
\end{aligned}$$

if we use the following "initial values" for step 13 *of the left and right line, respectively*:

$$\begin{aligned}
(A_L, B_L, C_L, D_L) &= (\texttt{0x50421004}, \texttt{0xbff56adf}, \texttt{0x00000000}, \texttt{0x00000000}),\\
(A_R, B_R, C_R, D_R) &= (\texttt{0x1021040a}, \texttt{0xfb5eaffd}, \texttt{0x3830a91b}, \texttt{0x21485a45}).
\end{aligned}$$

As already mentioned we do not describe the approach leading to the inner collision given in Lemma B, since it is too complicated to be presented here. The computational effort for finding these inner collisions is, also in this case, much smaller than the effort for finding backward collisions.

Remark on almost collisions. For the inner collision of compress$^{[23]}$ given in Lemma A, we have $\Delta = \tilde{X}_{13} - X_{13} = 1$. This Δ has Hamming weight 1. Hence the corresponding input pairs, which yield collisions of compress$^{[23]}$, give "almost collisions" for compress$^{[231]}$ (first round of compress put at the end, see end of the Appendix), in the sense that their outputs differ only at few bit positions. The reason is that the difference Δ between two inputs affects only the very end of the last round, namely steps 45, 46, and 47. About a portion of 1/70 (empirical observation) of the collisions for compress$^{[23]}$ found by our attack even have outputs under compress$^{[231]}$ different only at two bits. (See the end of Example A; unfortunately, a two-bit difference is the best result we can achieve in this way.)

Thus it is an interesting question whether Lemma B can also be shown for some Δ with Hamming weight 1. In that case, corresponding collisions of compress$^{[12]}$ would form almost collisions for the full compress function of RIPEMD. However, the best result that could be achieved in this way is a seven-bit difference in the hash values. The reason for this restriction is that the XOR operation H, which is applied in the last round, inherits differences.

4. Backward Collisions

The way to find backward collisions is the same for compress$^{[12]}$ and compress$^{[23]}$. Just to fix the notation, we consider compress$^{[23]}$ in what follows.

X_{13} does not occur in steps 0–12 and 19–31. Thus, in order to derive collisions of compress$^{[23]}$ from Lemma A we have to find "backward collisions" for the first 13 steps of the left and right line. That is, we have to compute $X_0, X_1, \ldots, X_6, X_8, \ldots, X_{12}$ and *common* initial values A, B, C, D such that the computation of L-compress$^{[23]}$ and R-compress$^{[23]}$ arrive after step 12 at (A_L, B_L, C_L, D_L) and (A_R, B_R, C_R, D_R) as specified in Lemma A, respectively, where $X_7 = -K_2$.

Lemma 1. *There is an algorithm which allows the computation of backward collisions in the above sense. When the algorithm is successful, about* 2^{36} *classes consisting of backward collisions, for which the initial values A and D can be chosen freely, are obtained. The algorithm requires on average about the same computational effort as* 2^{31} *computations of a (two-round) compress function.*

Proof. First we suppose that we have chosen $X_8, \ldots, X_{12}$. (Recall that X_7 is already fixed.) Then the contents of the registers after the application of step 6 for the left and the right line, respectively, say

$$(A_L^*, B_L^*, C_L^*, D_L^*) \quad \text{and} \quad (A_R^*, B_R^*, C_R^*, D_R^*),$$

are fixed and can be found easily by computing backward. It remains to determine $X_0, \ldots, X_6$ and A, B, C, D in order to find a backward collision. Table 5 shows the computation of steps 0–6.

Let s_i denote the number of bit positions to be shifted in step i. Moreover we use the notations

$$\Delta U = U_R - U_L,$$

Table 5

Initial values	A	B	C	D	Initial values	A	B	C	D
Step 0	U_L	B	C	D	Step 0	U_R	B	C	D
Step 1	U_L	B	C	V_L	Step 1	U_R	B	C	V_R
Step 2	U_L	B	W_L	V_L	Step 2	U_R	B	W_R	V_R
Step 3	U_L	B_L^*	W_L	V_L	Step 3	U_R	B_R^*	W_R	V_R
Step 4	A_L^*	B_L^*	W_L	V_L	Step 4	A_R^*	B_R^*	W_R	V_R
Step 5	A_L^*	B_L^*	W_L	D_L^*	Step 5	A_R^*	B_R^*	W_R	D_R^*
Step 6	A_L^*	B_L^*	C_L^*	D_L^*	Step 6	A_R^*	B_R^*	C_R^*	D_R^*

$$\Delta V = V_R - V_L,$$
$$\Delta W = W_R - W_L.$$

The following equations are easily derived:

$$0 = U_R^{\ll(-s_0)} - U_L^{\ll(-s_0)} + K_1, \tag{1}$$
$$G(U_R, B, C) - G(U_L, B, C) = V_R^{\ll(-s_1)} - V_L^{\ll(-s_1)} + K_1, \tag{2}$$
$$G(V_R, U_R, B) - G(V_L, U_L, B) = W_R^{\ll(-s_2)} - W_L^{\ll(-s_2)} + K_1, \tag{3}$$
$$G(W_R, V_R, U_R) - G(W_L, V_L, U_L) = B_R^{*\ll(-s_3)} - B_L^{*\ll(-s_3)} + K_1, \tag{4}$$
$$G(B_R^*, W_R, V_R) - G(B_L^*, W_L, V_L) = A_R^{*\ll(-s_4)} - A_L^{*\ll(-s_4)} - \Delta U + K_1, \tag{5}$$
$$G(A_R^*, B_R^*, W_R) - G(A_L^*, B_L^*, W_L) = D_R^{*\ll(-s_5)} - D_L^{*\ll(-s_5)} - \Delta V + K_1, \tag{6}$$
$$G(D_R^*, A_R^*, B_R^*) - G(D_L^*, A_L^*, B_L^*) = C_R^{*\ll(-s_6)} - C_L^{*\ll(-s_6)} - \Delta W + K_1. \tag{7}$$

In fact, these equations follow by elimination of X_i from the two equations defining step i. As an example, step 1 is defined by the equations

$$V_L = (D + G(U_L, B, C) + X_1 + K_1)^{\ll s_1},$$
$$V_R = (D + G(U_R, B, C) + X_1)^{\ll s_1},$$

which imply (2).

Equations (1)–(7) must hold when we have found a backward collision. Conversely, if $B, C, U_L, V_L, W_L, U_R, V_R, W_R$ satisfy (1)–(7), then we obtain 2^{64} backward collisions by setting, for arbitrary A and D:

$$X_0 = -A - G(B, C, D) + U_R^{\ll(-s_0)}, \tag{8}$$
$$X_1 = -D - G(U_R, B, C) + V_R^{\ll(-s_1)}, \tag{9}$$
$$X_2 = -C - G(V_R, U_R, B) + W_R^{\ll(-s_2)}. \tag{10}$$
$$X_3 = -B - G(W_R, V_R, U_R) + B_R^{*\ll(-s_3)}, \tag{11}$$
$$X_4 = -U_R - G(B_R^*, W_R, V_R) + A_R^{*\ll(-s_4)}, \tag{12}$$
$$X_5 = -V_R - G(A_R^*, B_R^*, W_R) + D_R^{*\ll(-s_5)}, \tag{13}$$
$$X_6 = -W_R - G(D_R^*, A_R^*, B_R^*) + C_R^{*\ll(-s_6)}. \tag{14}$$

Hence we try to find solutions $B, C, U_L, V_L, W_L, U_R, V_R, W_R$ of (1)–(7). To this end we make use of the following obvious fact:

There is a very fast algorithm which allows the computation of all solutions of an equation of the form

$$G(a_0, b_0, x + \Delta x) - G(a_1, b_1, x) = c \quad \text{with given } \Delta x, a_0, b_0, a_1, b_1, \text{ and } c. \tag{15}$$

In fact, simply solve this equation recursively and bitwise starting from the lowest bit. Empirical observations have shown that there exist solutions of (15) with a probability of about 2^{-11}, and in this case we obtain an average of 2^{11} solutions.

The basic idea is to derive a sequence of equations from (1)–(7), such that whenever the preceding equations are solved, the next equation is of the form (15). To manage this we make two additional settings, which in many cases will reduce the set of solutions, but on the other hand allow us to determine all remaining solutions very quickly:

$$\Delta U = -K_1^{\ll s_0} \quad (\text{or } (-K_1)^{\ll s_0}), \tag{16}$$

$$\Delta V = -K_1^{\ll s_1} \quad (\text{or } (-K_1)^{\ll s_1}). \tag{17}$$

The reason for (16) is (1), which suggests these guesses for the value of ΔU without knowing U_L and U_R. Similarly (17) hopefully implies

$$V_R^{\ll(-s_1)} - V_L^{\ll(-s_1)} + K_1 = 0,$$

such that (2) would have at least the trivial solution $B = C$. Further observe that ΔW is fixed by (7):

$$\Delta W = G(D_R^*, A_R^*, B_R^*) - G(D_L^*, A_L^*, B_L^*) - C_R^{*\ll(-s_6)} + C_L^{*\ll(-s_6)} + K_1.$$

Equations (6), (5), ..., (2), in this ordering, are now used to compute W_L, V_L, U_L, B, and finally C:

$$G(A_R^*, B_R^*, W_L + \Delta W) - G(A_L^*, B_L^*, W_L) = D_R^{*\ll(-s_5)} - D_L^{*\ll(-s_5)} - \Delta V + K_1, \quad \text{(I)}$$

$$G(B_R^*, W_R, V_L + \Delta V) - G(B_L^*, W_L, V_L) = A_R^{*\ll(-s_4)} - A_L^{*\ll(-s_4)} - \Delta U + K_1, \quad \text{(II)}$$

$$G(W_R, V_R, U_L + \Delta U) - G(W_L, V_L, U_L) = B_R^{*\ll(-s_3)} - B_L^{*\ll(-s_3)} + K_1, \quad \text{(III)}$$

$$G(V_R, U_R, B) - G(V_L, U_L, B) = W_R^{\ll(-s_2)} - W_L^{\ll(-s_2)} + K_1, \quad \text{(IV)}$$

$$G(U_R, B, C) - G(U_L, B, C) = V_R^{\ll(-s_1)} - V_L^{\ll(-s_1)} + K_1. \quad \text{(V)}$$

After these preparations we can describe how the algorithm for the search of backward collisions works in its simplest form:

1. Choose $X_8, \ldots, X_{12}$ randomly and compute the associated values for $A_L^*, B_L^*, C_L^*, D_L^*$ and $A_R^*, B_R^*, C_R^*, D_R^*$. Determine the solutions W_L of (I). If they exist, insert each of them into (II) in order to determine all associated V_L. In case of success, the next step is to consider (III) in order to compute U_L, and so on. Refuse each possibly obtained U_L if it does not satisfy (1).
2. If $X_8, \ldots, X_{12}$ have been found such that there are solutions up to (II), then take these as "basic values." Change randomly one bit of each of the basic values for $X_8, \ldots, X_{12}$. (The idea is that if a set of values of $X_8, \ldots, X_{12}$ gives solutions to

steps (I) and (II), then values close to them again give solutions to steps (I) and (II) with a relatively high probability.) Only if you again come to solutions up to (II) take the changed value as new basic values and continue until you reach (III), and so on.

Practice has shown that this leads to solutions up to (IV), and then, in most cases, automatically to solutions of (V).

Observe that the set of solutions has the structure of a tree. We have a set of solutions W_L of (I), such that to each W_L there corresponds a set of solutions V_L of (II), and so on. This "avalanche" effect has the consequence that in case we can reach (V), we even find 2^{36} or more solutions. □

The combination of Lemmas A and B with Lemma 1 implies directly:

Lemma 2. *There is an algorithm which allows the computation of collisions of* compress$^{[12]}$ *and* compress$^{[23]}$ *with about* 2^{100} *different initial values. It requires on average about the same computational effort as* 2^{31} *computations of a (two-round) compress function.*

In next example one class of 2^{64} collisions for compress$^{[23]}$ is given. It has been found by the previously described attack (Lemmas A and 1).

Example A (Class of Collisions for compress$^{[23]}$). For arbitrary A and D set

$$(B, C) = (\texttt{0x828a0950}, \texttt{0x98080110}),$$

$$\begin{aligned}
X_0 &= \texttt{0x12481059} + G(B, C, 0) - G(B, C, D) - A,\\
X_1 &= \texttt{0xa9368b18} - D,\\
X_2 &= \texttt{0xdf3dae71},\\
X_3 &= \texttt{0x88bacd2a},\\
X_4 &= \texttt{0x618ec5d2},\\
X_5 &= \texttt{0x53b59054},\\
X_6 &= \texttt{0x16b396fc},\\
X_7 &= \texttt{0x9126145f},\\
X_8 &= \texttt{0x972a229d},\\
X_9 &= \texttt{0xe9098eee},\\
X_{10} &= \texttt{0xf0f1721d},\\
X_{11} &= \texttt{0xf8dbb766},\\
X_{12} &= \texttt{0x753ed5cb},\\
X_{13} &= \texttt{0xa57d8666},\\
X_{14} &= \texttt{0xa57d8666},\\
X_{15} &= \texttt{0xa57d8666},
\end{aligned}$$

$$\tilde{X}_i = X_i \qquad \text{for } i < 16, \quad i \neq 13,$$
$$\tilde{X}_{13} = X_{13} + 1 = \texttt{0xa57d8667}.$$

Then we have

$$\text{compress}^{[23]}(A, B, C, D; X_0, \ldots, X_{15}) = \text{compress}^{[23]}(A, B, C, D; \tilde{X}_0, \ldots, \tilde{X}_{15}).$$

Explicitly, if for instance we set $A = \texttt{0x03d06da3}$ and $D = \texttt{0x6a645c74}$, then the common output under $\text{compress}^{[23]}$ is

$$\texttt{0x7f733e3b 0x3bac842a 0x061c7cac 0x0ca41089}.$$

Moreover, we have

$$\text{compress}^{[231]}(A, B, C, D; X)$$
$$= \texttt{0x9b888272 0x66483211 0x0692b464 0x758f548b},$$
$$\text{compress}^{[231]}(A, B, C, D; \tilde{X})$$
$$= \texttt{0x9b8882f2 0x66483291 0x0692b464 0x758f548b}.$$

That is, the outputs under $\text{compress}^{[231]}$ differ only at two bits.

5. Right Initial Value

By Lemma 2 we find large classes of collisions for the compress functions of RIPEMD[12] and RIPEMD[23], but not yet for these hash functions, because the required fixed initial value IV_0 will usually not be among the admissible initial values for collisions found. However, as we shall see below, this problem is easily avoided by a straightforward meet-in-the-middle attack.

Example B (Collision for RIPEMD[12]). The first message M is the sequence of the following words:

$$\begin{aligned}
M_0 &= \texttt{0x179ee2b9}, & M_{16} &= \texttt{0x61dfb182},\\
M_1 &= \texttt{0x6e14b784}, & M_{17} &= \texttt{0x1808647d},\\
M_2 &= \texttt{0x2fa520b7}, & M_{18} &= \texttt{0xfd84f2e1},\\
M_3 &= \texttt{0x09e5c84a}, & M_{19} &= \texttt{0xb8647a90},\\
M_4 &= \texttt{0x6a81e3b5}, & M_{20} &= \texttt{0xc3dd8441},\\
M_5 &= \texttt{0x50c8fbfe}, & M_{21} &= \texttt{0xa1fef1e1},\\
M_6 &= \texttt{0x6d390c47}, & M_{22} &= \texttt{0x7d15c061},\\
M_7 &= \texttt{0x7f5292b4}, & M_{23} &= \texttt{0x3a456372},\\
M_8 &= \texttt{0x50f05934}, & M_{24} &= \texttt{0x020828dd},\\
M_9 &= \texttt{0x2acd6dd7}, & M_{25} &= \texttt{0x2be014ea},\\
M_{10} &= \texttt{0x4afbae78}, & M_{26} &= \texttt{0x321373f7},\\
M_{11} &= \texttt{0x28eda5c3}, & M_{27} &= \texttt{0x0b2266e7},\\
M_{12} &= \texttt{0x3217765e}, & M_{28} &= \texttt{0x87e9ddd2},\\
M_{13} &= \texttt{0x16f6e1a4}, & M_{29} &= \texttt{0xb6c474bc},\\
M_{14} &= \texttt{0x54b57bb0}, & M_{30} &= \texttt{0x1e575831},\\
M_{15} &= \texttt{0x46cbe2b6}, & M_{31} &= \texttt{0x767f3bbb}.
\end{aligned}$$

The second message $\tilde{M}$ is defined by setting $\tilde{M}_i = M_i$ $(i < 32, i \neq 29)$ and

$$\tilde{M}_{29} = \texttt{0x60191e11}.$$

RIPEMD[12] associates to M and $\tilde{M}$ the same hash value, namely

$$\texttt{0x7b4d3b7f 0x792ae282 0xfee3cfe0 0xb8cee276},$$

where the padding rule is also considered. It requires that the 16 words

$$\texttt{0x80, 0, 0, ..., 0, 0x400, 0}$$

have to be added to M and $\tilde{M}$ (see [1]).

This example has been found as follows. First, based on Lemma B, the algorithm described in the proof of Lemma 1 is applied (i.e., $X_{13} = M_{29}$, $X_{14} = M_{30}$, $X_{15} = M_{31}$, $X_7 = M_{23}$ are given according to Lemma B). In this way we find $X_8 = M_{24}, \ldots, X_{12} = M_{28}$ such that we have a complete tree of solutions for W_L, W_R, V_L, V_R, U_L, U_R, B, and C satisfying (I)–(V). (Note that here we have to modify the equations in Section 4 by replacing K_1 by $-K_0$, because we are now dealing with the first two instead of the last two rounds.)

From this tree we take a collection of fixed values for W_L, W_R, V_L, V_R, U_L, and U_R, i.e., (I)–(III) are satisfied. In view of (12), (13), (14) the words $X_4 = M_{20}$, $X_5 = M_{21}$, $X_6 = M_{22}$ are now fixed.

Finally choose values for $M_0, \ldots, M_{15}$ randomly, compute

$$\mathsf{compress}^{[12]}(IV_0; M_0, \ldots, M_{15}) = (A, B, C, D)$$

with

$$IV_0 = \texttt{0x67452301 0xefcdab89 0x98badcfe 0x10325476}$$

(see the Appendix) and test, for B and C, the following equations:

$$F(V_R, U_R, B) - F(V_L, U_L, B) = W_R^{\ll(-s_2)} - W_L^{\ll(-s_2)} - K_0, \quad \text{(IV)}$$
$$F(U_R, B, C) - F(U_L, B, C) = V_R^{\ll(-s_1)} - V_L^{\ll(-s_1)} - K_0. \quad \text{(V)}$$

If the test is passed, then (A, B, C, D) is an admissible initial value for the second application of $\mathsf{compress}^{[12]}$, and we define $X_0 = M_{16}$, $X_1 = M_{17}$, $X_2 = M_{18}$, and $X_3 = M_{19}$ using (8)–(11).

Concretely in our above example, we have chosen

$$\begin{aligned} W_L &= \texttt{0xcef895d6}, \\ W_R &= \texttt{0x140279c4}, \\ V_L &= \texttt{0x0d5e7745}, \\ V_R &= \texttt{0xb0580b6d}, \\ U_L &= \texttt{0x83901a28}, \\ U_R &= \texttt{0x97ef4cad} \end{aligned}$$

from a complete tree of solutions of (I)–(V) which have been found. There are 2^{32} pairs (B, C) satisfying (IV) and (V) in this case.

We can improve this attack by selecting not only one, but several collections of values for W_L, W_R, V_L, V_R, U_L, U_R from the tree of solutions of (I)–(III). Another trick reducing the effort is taking $M_0, \ldots, M_{14}$ fixed and changing only M_{15}. Then only one round of the compress function has to be computed.

6. MD4 Is Not Collision-Free

We have found that the methods developed in this note can be applied to MD4 very effectively [3]. In fact, collisions can be found for the full MD4 in less than 1 minute on a PC.

Example C (Collision for MD4). The first message M is the sequence of the following 32-bit words:

$$\begin{aligned}
M_0 &= \texttt{0x13985e12}, & M_8 &= \texttt{0xabe17be0},\\
M_1 &= \texttt{0x748a810b}, & M_9 &= \texttt{0xed1ed4b3},\\
M_2 &= \texttt{0x4d1df15a}, & M_{10} &= \texttt{0x4120abf5},\\
M_3 &= \texttt{0x181d1516}, & M_{11} &= \texttt{0x20771029},\\
M_4 &= \texttt{0x2d6e09ac}, & M_{12} &= \texttt{0x20771027},\\
M_5 &= \texttt{0x4b6dbdb9}, & M_{13} &= \texttt{0xfdfffbff},\\
M_6 &= \texttt{0x6464b0c8}, & M_{14} &= \texttt{0xffffbffb},\\
M_7 &= \texttt{0xfba1c097}, & M_{15} &= \texttt{0x6774bed2}.
\end{aligned}$$

The second message $\tilde{M}$ is defined by setting $\tilde{M}_i = M_i$ $(i < 16, i \neq 12)$ and

$$\tilde{M}_{12} = M_{12} + 1.$$

MD4 associates to M and $\tilde{M}$ the same hash value

$$\texttt{0x711ad51b 0xbbab5e22 0x618b1c76 0x17c15892}.$$

Acknowledgment

The author would like to thank Antoon Bosselaers for checking the attack presented and giving several useful hints improving the exposition.

Appendix

The hash function RIPEMD is defined as the iteration of a certain compress function, which we specify below. The computation starts with the initial value

$$IV_0 = \texttt{0x67452301 0xefcdab89 0x98badcfe 0x10325476}.$$

Each application of the compress function uses four words as initial values and 16 words of the message as input, and it gives four words output, which are then used as initial

values for the next application. The final output is the hash value. This works, since there is a padding rule (addition of bits to the message such that its length is a multiple of $512 = 16\times$ (*length of words*)). A complete description of RIPEMD can be found in [1].

A.1. *Compress Function of RIPEMD*

Define the constants

$$\begin{aligned} K_0 &= \texttt{0x50a28be6}, \\ K_1 &= \texttt{0x5a827999}, \\ K_2 &= \texttt{0x6ed9eba1}, \\ K_3 &= \texttt{0x5c4dd124} \end{aligned}$$

and the Boolean vector functions

$$\begin{aligned} F(X, Y, Z) &= (X \wedge Y) \vee (\neg X \wedge Z), \\ G(X, Y, Z) &= (X \wedge Y) \vee (X \wedge Z) \vee (Y \wedge Z), \\ H(X, Y, Z) &= X \oplus Y \oplus Z. \end{aligned}$$

Further, let $FF(a, b, c, d, Z, s)$, $GG(a, b, c, d, Z, s)$, and $HH(a, b, c, d, Z, s)$ denote the operations

$$\begin{aligned} a &:= (a + F(b, c, d) + Z)^{\ll s}, \\ a &:= (a + G(b, c, d) + Z)^{\ll s}, \\ a &:= (a + H(b, c, d) + Z)^{\ll s}, \end{aligned}$$

respectively. In order to define compress for RIPEMD suppose now that initial values A, B, C, D and inputs $X_0, \ldots, X_{15}$ are given. Copy A, B, C, D into the registers aa, bb, cc, dd of the left line and the registers aaa, bbb, ccc, ddd of the right line. Then apply the following steps:

First round

step 0	$FF(aa, bb, cc, dd, X_0, 11)$	$FF(aaa, bbb, ccc, ddd, X_0 + K_0, 11)$
step 1	$FF(dd, aa, bb, cc, X_1, 14)$	$FF(ddd, aaa, bbb, ccc, X_1 + K_0, 14)$
step 2	$FF(cc, dd, aa, bb, X_2, 15)$	$FF(ccc, ddd, aaa, bbb, X_2 + K_0, 15)$
step 3	$FF(bb, cc, dd, aa, X_3, 12)$	$FF(bbb, ccc, ddd, aaa, X_3 + K_0, 12)$
step 4	$FF(aa, bb, cc, dd, X_4, 5)$	$FF(aaa, bbb, ccc, ddd, X_4 + K_0, 5)$
step 5	$FF(dd, aa, bb, cc, X_5, 8)$	$FF(ddd, aaa, bbb, ccc, X_5 + K_0, 8)$
step 6	$FF(cc, dd, aa, bb, X_6, 7)$	$FF(ccc, ddd, aaa, bbb, X_6 + K_0, 7)$
step 7	$FF(bb, cc, dd, aa, X_7, 9)$	$FF(bbb, ccc, ddd, aaa, X_7 + K_0, 9)$
step 8	$FF(aa, bb, cc, dd, X_8, 11)$	$FF(aaa, bbb, ccc, ddd, X_8 + K_0, 11)$

step 9	$FF(dd, aa, bb, cc, X_9, 13)$	$FF(ddd, aaa, bbb, ccc, X_9 + K_0, 13)$
step 10	$FF(cc, dd, aa, bb, X_{10}, 14)$	$FF(ccc, ddd, aaa, bbb, X_{10} + K_0, 14)$
step 11	$FF(bb, cc, dd, aa, X_{11}, 15)$	$FF(bbb, ccc, ddd, aaa, X_{11} + K_0, 15)$
step 12	$FF(aa, bb, cc, dd, X_{12}, 6)$	$FF(aaa, bbb, ccc, ddd, X_{12} + K_0, 6)$
step 13	$FF(dd, aa, bb, cc, X_{13}, 7)$	$FF(ddd, aaa, bbb, ccc, X_{13} + K_0, 7)$
step 14	$FF(cc, dd, aa, bb, X_{14}, 9)$	$FF(ccc, ddd, aaa, bbb, X_{14} + K_0, 9)$
step 15	$FF(bb, cc, dd, aa, X_{15}, 8)$	$FF(bbb, ccc, ddd, aaa, X_{15} + K_0, 8)$

Second round

step 16	$GG(aa, bb, cc, dd, X_7 + K_1, 7)$	$GG(aaa, bbb, ccc, ddd, X_7, 7)$
step 17	$GG(dd, aa, bb, cc, X_4 + K_1, 6)$	$GG(ddd, aaa, bbb, ccc, X_4, 6)$
step 18	$GG(cc, dd, aa, bb, X_{13} + K_1, 8)$	$GG(ccc, ddd, aaa, bbb, X_{13}, 8)$
step 19	$GG(bb, cc, dd, aa, X_1 + K_1, 13)$	$GG(bbb, ccc, ddd, aaa, X_1, 13)$
step 20	$GG(aa, bb, cc, dd, X_{10} + K_1, 11)$	$GG(aaa, bbb, ccc, ddd, X_{10}, 11)$
step 21	$GG(dd, aa, bb, cc, X_6 + K_1, 9)$	$GG(ddd, aaa, bbb, ccc, X_6, 9)$
step 22	$GG(cc, dd, aa, bb, X_{15} + K_1, 7)$	$GG(ccc, ddd, aaa, bbb, X_{15}, 7)$
step 23	$GG(bb, cc, dd, aa, X_3 + K_1, 15)$	$GG(bbb, ccc, ddd, aaa, X_3, 15)$
step 24	$GG(aa, bb, cc, dd, X_{12} + K_1, 7)$	$GG(aaa, bbb, ccc, ddd, X_{12}, 7)$
step 25	$GG(dd, aa, bb, cc, X_0 + K_1, 12)$	$GG(ddd, aaa, bbb, ccc, X_0, 12)$
step 26	$GG(cc, dd, aa, bb, X_9 + K_1, 15)$	$GG(ccc, ddd, aaa, bbb, X_9, 15)$
step 27	$GG(bb, cc, dd, aa, X_5 + K_1, 9)$	$GG(bbb, ccc, ddd, aaa, X_5, 9)$
step 28	$GG(aa, bb, cc, dd, X_{14} + K_1, 7)$	$GG(aaa, bbb, ccc, ddd, X_{14}, 7)$
step 29	$GG(dd, aa, bb, cc, X_2 + K_1, 11)$	$GG(ddd, aaa, bbb, ccc, X_2, 11)$
step 30	$GG(cc, dd, aa, bb, X_{11} + K_1, 13)$	$GG(ccc, ddd, aaa, bbb, X_{11}, 13)$
step 31	$GG(bb, cc, dd, aa, X_8 + K_1, 12)$	$GG(bbb, ccc, ddd, aaa, X_8, 12)$

Third round

step 32	$HH(aa, bb, cc, dd, X_3 + K_2, 11)$	$HH(aaa, bbb, ccc, ddd, X_3 + K_3, 11)$
step 33	$HH(dd, aa, bb, cc, X_{10} + K_2, 13)$	$HH(ddd, aaa, bbb, ccc, X_{10} + K_3, 13)$
step 34	$HH(cc, dd, aa, bb, X_2 + K_2, 14)$	$HH(ccc, ddd, aaa, bbb, X_2 + K_3, 14)$
step 35	$HH(bb, cc, dd, aa, X_4 + K_2, 7)$	$HH(bbb, ccc, ddd, aaa, X_4 + K_3, 7)$
step 36	$HH(aa, bb, cc, dd, X_9 + K_2, 14)$	$HH(aaa, bbb, ccc, ddd, X_9 + K_3, 14)$
step 37	$HH(dd, aa, bb, cc, X_{15} + K_2, 9)$	$HH(ddd, aaa, bbb, ccc, X_{15} + K_3, 9)$
step 38	$HH(cc, dd, aa, bb, X_8 + K_2, 13)$	$HH(ccc, ddd, aaa, bbb, X_8 + K_3, 13)$
step 39	$HH(bb, cc, dd, aa, X_1 + K_2, 15)$	$HH(bbb, ccc, ddd, aaa, X_1 + K_3, 15)$
step 40	$HH(aa, bb, cc, dd, X_{14} + K_2, 6)$	$HH(aaa, bbb, ccc, ddd, X_{14} + K_3, 6)$
step 41	$HH(dd, aa, bb, cc, X_7 + K_2, 8)$	$HH(ddd, aaa, bbb, ccc, X_7 + K_3, 8)$
step 42	$HH(cc, dd, aa, bb, X_0 + K_2, 13)$	$HH(ccc, ddd, aaa, bbb, X_0 + K_3, 13)$
step 43	$HH(bb, cc, dd, aa, X_6 + K_2, 6)$	$HH(bbb, ccc, ddd, aaa, X_6 + K_3, 6)$
step 44	$HH(aa, bb, cc, dd, X_{11} + K_2, 12)$	$HH(aaa, bbb, ccc, ddd, X_{11} + K_3, 12)$
step 45	$HH(dd, aa, bb, cc, X_{13} + K_2, 5)$	$HH(ddd, aaa, bbb, ccc, X_{13} + K_3, 5)$
step 46	$HH(cc, dd, aa, bb, X_5 + K_2, 7)$	$HH(ccc, ddd, aaa, bbb, X_5 + K_3, 7)$
step 47	$HH(bb, cc, dd, aa, X_{12} + K_2, 5)$	$HH(bbb, ccc, ddd, aaa, X_{12} + K_3, 5)$

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

$$\begin{aligned} AA &= B + cc + ddd, \\ BB &= C + dd + aaa, \\ CC &= D + aa + bbb, \\ DD &= A + bb + ccc. \end{aligned}$$

That is one sets

$$\mathsf{compress}(A, B, C, D; X_0, X_1, \ldots, X_{15}) = (AA, BB, CC, DD).$$

A.2. Compress Functions of RIPEMD[12] and RIPEMD[23]

By $\mathsf{compress}^{[12]}$ we denote the reduced version of compress, where the last round (steps 32–47) is omitted. Similarly, $\mathsf{compress}^{[23]}$ denotes the reduced version of compress, where the first round (steps 0–15) is omitted and—for the sake of convenience—the application of the X_i is permuted such that they occur in their natural ordering in the first round of $\mathsf{compress}^{[23]}$ (i.e., the second round of compress). Explicitly the steps of $\mathsf{compress}^{[23]}$ are therefore:

First round of $\mathsf{compress}^{[23]}$

step 0	$GG(aa, bb, cc, dd, X_0 + K_1, 7)$	$GG(aaa, bbb, ccc, ddd, X_0, 7)$
step 1	$GG(dd, aa, bb, cc, X_1 + K_1, 6)$	$GG(ddd, aaa, bbb, ccc, X_1, 6)$
step 2	$GG(cc, dd, aa, bb, X_2 + K_1, 8)$	$GG(ccc, ddd, aaa, bbb, X_2, 8)$
step 3	$GG(bb, cc, dd, aa, X_3 + K_1, 13)$	$GG(bbb, ccc, ddd, aaa, X_3, 13)$
step 4	$GG(aa, bb, cc, dd, X_4 + K_1, 11)$	$GG(aaa, bbb, ccc, ddd, X_4, 11)$
step 5	$GG(dd, aa, bb, cc, X_5 + K_1, 9)$	$GG(ddd, aaa, bbb, ccc, X_5, 9)$
step 6	$GG(cc, dd, aa, bb, X_6 + K_1, 7)$	$GG(ccc, ddd, aaa, bbb, X_6, 7)$
step 7	$GG(bb, cc, dd, aa, X_7 + K_1, 15)$	$GG(bbb, ccc, ddd, aaa, X_7, 15)$
step 8	$GG(aa, bb, cc, dd, X_8 + K_1, 7)$	$GG(aaa, bbb, ccc, ddd, X_8, 7)$
step 9	$GG(dd, aa, bb, cc, X_9 + K_1, 12)$	$GG(ddd, aaa, bbb, ccc, X_9, 12)$
step 10	$GG(cc, dd, aa, bb, X_{10} + K_1, 15)$	$GG(ccc, ddd, aaa, bbb, X_{10}, 15)$
step 11	$GG(bb, cc, dd, aa, X_{11} + K_1, 9)$	$GG(bbb, ccc, ddd, aaa, X_{11}, 9)$
step 12	$GG(aa, bb, cc, dd, X_{12} + K_1, 7)$	$GG(aaa, bbb, ccc, ddd, X_{12}, 7)$
step 13	$GG(dd, aa, bb, cc, X_{13} + K_1, 11)$	$GG(ddd, aaa, bbb, ccc, X_{13}, 11)$
step 14	$GG(cc, dd, aa, bb, X_{14} + K_1, 13)$	$GG(ccc, ddd, aaa, bbb, X_{14}, 13)$
step 15	$GG(bb, cc, dd, aa, X_{15} + K_1, 12)$	$GG(bbb, ccc, ddd, aaa, X_{15}, 12)$

Second round of $\mathsf{compress}^{[23]}$

step 16	$HH(aa, bb, cc, dd, X_7 + K_2, 11)$	$HH(aaa, bbb, ccc, ddd, X_7 + K_3, 11)$
step 17	$HH(dd, aa, bb, cc, X_4 + K_2, 13)$	$HH(ddd, aaa, bbb, ccc, X_4 + K_3, 13)$
step 18	$HH(cc, dd, aa, bb, X_{13} + K_2, 14)$	$HH(ccc, ddd, aaa, bbb, X_{13} + K_3, 14)$

step 19	$HH(bb, cc, dd, aa, X_1 + K_2, 7)$	$HH(bbb, ccc, ddd, aaa, X_1 + K_3, 7)$
step 20	$HH(aa, bb, cc, dd, X_{10} + K_2, 14)$	$HH(aaa, bbb, ccc, ddd, X_{10} + K_3, 14)$
step 21	$HH(dd, aa, bb, cc, X_6 + K_2, 9)$	$HH(ddd, aaa, bbb, ccc, X_6 + K_3, 9)$
step 22	$HH(cc, dd, aa, bb, X_{15} + K_2, 13)$	$HH(ccc, ddd, aaa, bbb, X_{15} + K_3, 13)$
step 23	$HH(bb, cc, dd, aa, X_3 + K_2, 15)$	$HH(bbb, ccc, ddd, aaa, X_3 + K_3, 15)$
step 24	$HH(aa, bb, cc, dd, X_{12} + K_2, 6)$	$HH(aaa, bbb, ccc, ddd, X_{12} + K_3, 6)$
step 25	$HH(dd, aa, bb, cc, X_0 + K_2, 8)$	$HH(ddd, aaa, bbb, ccc, X_0 + K_3, 8)$
step 26	$HH(cc, dd, aa, bb, X_9 + K_2, 13)$	$HH(ccc, ddd, aaa, bbb, X_9 + K_3, 13)$
step 27	$HH(bb, cc, dd, aa, X_5 + K_2, 6)$	$HH(bbb, ccc, ddd, aaa, X_5 + K_3, 6)$
step 28	$HH(aa, bb, cc, dd, X_{14} + K_2, 12)$	$HH(aaa, bbb, ccc, ddd, X_{14} + K_3, 12)$
step 29	$HH(dd, aa, bb, cc, X_2 + K_2, 5)$	$HH(ddd, aaa, bbb, ccc, X_2 + K_3, 5)$
step 30	$HH(cc, dd, aa, bb, X_{11} + K_2, 7)$	$HH(ccc, ddd, aaa, bbb, X_{11} + K_3, 7)$
step 31	$HH(bb, cc, dd, aa, X_8 + K_2, 5)$	$HH(bbb, ccc, ddd, aaa, X_8 + K_3, 5)$

The reduced version of RIPEMD, where compress is replaced by compress$^{[12]}$ (resp. compress$^{[23]}$), is denoted by RIPEMD$^{[12]}$ (resp. RIPEMD$^{[23]}$).

Because we refer to it in a remark concerning almost collisions (see the end of Section 3), we finally define the three-round compress function compress$^{[231]}$ as a modification of the original RIPEMD compress, where the first round is put at the end. The first two rounds of compress$^{[231]}$ coincide with compress$^{[23]}$. The ordering in which the blocks X_i are applied in the single steps of compress$^{[231]}$ is the same as for compress, since this ordering is a fundamental design principle of RIPEMD. The last round of compress$^{[231]}$ is therefore defined as follows:

step 32	$FF(aa, bb, cc, dd, X_3, 11)$	$FF(aaa, bbb, ccc, ddd, X_3 + K_0, 11)$
step 33	$FF(dd, aa, bb, cc, X_{10}, 14)$	$FF(ddd, aaa, bbb, ccc, X_{10} + K_0, 14)$
step 34	$FF(cc, dd, aa, bb, X_2, 15)$	$FF(ccc, ddd, aaa, bbb, X_2 + K_0, 15)$
step 35	$FF(bb, cc, dd, aa, X_4, 12)$	$FF(bbb, ccc, ddd, aaa, X_4 + K_0, 12)$
step 36	$FF(aa, bb, cc, dd, X_9, 5)$	$FF(aaa, bbb, ccc, ddd, X_9 + K_0, 5)$
step 37	$FF(dd, aa, bb, cc, X_{15}, 8)$	$FF(ddd, aaa, bbb, ccc, X_{15} + K_0, 8)$
step 38	$FF(cc, dd, aa, bb, X_8, 7)$	$FF(ccc, ddd, aaa, bbb, X_8 + K_0, 7)$
step 39	$FF(bb, cc, dd, aa, X_1, 9)$	$FF(bbb, ccc, ddd, aaa, X_1 + K_0, 9)$
step 40	$FF(aa, bb, cc, dd, X_{14}, 11)$	$FF(aaa, bbb, ccc, ddd, X_{14} + K_0, 11)$
step 41	$FF(dd, aa, bb, cc, X_7, 13)$	$FF(ddd, aaa, bbb, ccc, X_7 + K_0, 13)$
step 42	$FF(cc, dd, aa, bb, X_0, 14)$	$FF(ccc, ddd, aaa, bbb, X_0 + K_0, 14)$
step 43	$FF(bb, cc, dd, aa, X_6, 15)$	$FF(bbb, ccc, ddd, aaa, X_6 + K_0, 15)$
step 44	$FF(aa, bb, cc, dd, X_{11}, 6)$	$FF(aaa, bbb, ccc, ddd, X_{11} + K_0, 6)$
step 45	$FF(dd, aa, bb, cc, X_{13}, 7)$	$FF(ddd, aaa, bbb, ccc, X_{13} + K_0, 7)$
step 46	$FF(cc, dd, aa, bb, X_5, 9)$	$FF(ccc, ddd, aaa, bbb, X_5 + K_0, 9)$
step 47	$FF(bb, cc, dd, aa, X_{12}, 8)$	$FF(bbb, ccc, ddd, aaa, X_{12} + K_0, 8)$

Note Added in Proof (July 1996). As a reaction to the presented attack, RIPEMD has meanwhile been replaced in the ISO/IEC Draft 10118-3 by its strengthened successors RIPEMD-160 and RIPEMD-128 (H. Dobbertin, A. Bosselaers, and B. Preneel, RIPEMD-160: A strengthened version of RIPEMD, *Fast Software Encryption* (*Proceed-*

ings of the 1996 *Cambridge Workshop on Cryptographic Algorithms*), Lecture Notes in Computer Science, vol. 1039, Springer-Verlag, Berlin, 1996, pp. 71–82).

Very recently it has been demonstrated that collisions of the compress function of MD5 can be found (H. Dobbertin, The status of MD5 after a recent attack, *CryptoBytes, The technical newsletter of RSA Laboratories*, vol. 2, Summer issue, 1996).

References

[1] A. Bosselaers and B. Preneel (eds.), *Integrity Primitives for Secure Information Systems, Final Report of RACE Integrity Primitives Evaluation* (*RIPE-RACE* 1040), Chapter 3: RIPEMD, Lecture Notes in Computer Science, vol. 1007, Springer-Verlag, Berlin, 1995, pp. 69–111.

[2] B. den Boer and A. Bosselaers, An attack on the last two rounds of MD4, *Advances in Cryptology, CRYPTO '91*, Lecture Notes in Computer Science, vol. 576, Springer-Verlag, Berlin, 1992, pp. 194–203.

[3] H. Dobbertin, Cryptanalysis of MD4, *Fast Software Encryption* (*Proceedings of the* 1996 *Cambridge Workshop on Cryptographic Algorithms*), Lecture Notes in Computer Science, vol. 1039, Springer-Verlag, Berlin, 1996, pp. 53–69. (An extended version will appear in this journal.)

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

[5] R. Rivest, the MD5 message-digest algorithm, *Request for Comments* (*RFC*) 1321, Internet Activities Board, Internet Privacy Task Force, April 1992.

[6] S. Vaudenay, On the need of multipermutations: Cryptanalysis of MD4 and SAFER, *Fast Software Encryption* (*Proceedings of the* 1994 *Leuven Workshop on Cryptographic Algorithms*), Lecture Notes in Computer Science, vol. 1008, Springer-Verlag, Berlin, 1995, pp. 286–297.