

Cryptanalysis of MD4

Hans Dobbertin

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

Communicated by Ivan Damgård

Received 23 October 1995 and revised 31 August 1997

Abstract. In 1990 Rivest introduced the hash function MD4. Two years later RIPEMD, a European proposal, was designed as a stronger mode of MD4. In 1995 the author found an attack against two of three rounds of RIPEMD. As we show in the present note, the methods developed to attack RIPEMD can be modified and supplemented such that it is possible to break the full MD4, while previously only partial attacks were known. An implementation of our attack allows us to find collisions for MD4 in a few seconds on a PC. An example of a collision is given demonstrating that our attack is of practical relevance.

Key words. Dedicated hash functions, Collisions, MD4, MD5, RIPEMD.

1. Introduction

Rivest [8] introduced the hash function MD4 in 1990. The MD4 algorithm is defined as an iterative application of a three-round compression function. After an unpublished attack on the first two rounds of MD4 due to Merkle and an attack against the last two rounds by den Boer and Bosselaers [3], Rivest introduced the strengthened version MD5 [9]. The most important difference to MD4 is the addition of a fourth round.

On the other hand, the stronger mode RIPEMD [1] of MD4 was designed as a European proposal in 1992. The compression function of RIPEMD consists of two parallel lines of a modified version of the MD4 compression function. In [5] we have shown that if the first or the last round of its compression function is omitted, then RIPEMD is not collision-free.

Vaudenay [11] described another attack against the first two rounds of MD4. The two round collisions he found form almost-collisions for the full MD4. However, none of the previously known partial attacks can be generalized to the three-round MD4. Thus, the original conjecture that MD4 is collision-free still remained to be disproved. This is done in this paper.

We show that the methods developed to attack RIPEMD can be applied to MD4 very

effectively. We derive an algorithm which allows us to compute collisions for the full MD4 in a few seconds on a PC with a Pentium processor. Finally it is demonstrated that a further development of our attack allows us to find collisions for meaningful messages. A recent result sets even the one-wayness of MD4 in question [6]. Therefore we suggest that MD4 should no longer be applied in practice.

A more sophisticated attack, based on the same techniques as described here for MD4, can be successfully applied to MD5. In [4] it is explained how collisions for the compression function of MD5 can be found. We recommend RIPEMD-160 [7] as a replacement for MD4, MD5, and RIPEMD.

Terminology and Basic Notations

Using the term “collision of a compression function” we assume that the corresponding initial values coincide for both inputs. For “pseudocollisions” 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. Accordingly the value of an expression is its remainder modulo 2^{32} , and equations are understood to be modulo 2^{32} . The symbols \wedge , \vee , \oplus , and \neg are used for bitwise AND, OR, XOR, and complement, respectively. For a 32-bit word W , let $W^{\ll s}$ denote the 32-bit value obtained by circularly shifting (rotation) W left by s bit positions for $0 \leq s < 32$. If W is an expression then, of course, evaluate it before shifting. Further, we agree that $-W^{\ll s}$ stands for $-(W^{\ll s})$.

A definition of the compression function of MD4 can be found in the Appendix.

2. Main Result and Plan of the Attack

The main result of this paper is:

MD4 is not collision-free.

There is an algorithm such that the finding of collisions for MD4 requires the same computational effort as about 2^{20} computations of the MD4-compression function.

The most direct way of trying to get a collision for an iterated hash function like MD4 is by trying to find a collision for the compression function with the fixed initial value required at the beginning of the computation of hash values. This is precisely what will be done in this paper for MD4.

Throughout this note $X = (X_i)_{i < 16}$ denotes a collection of 16 words, and the collection $\tilde{X} = (\tilde{X}_i)_{i < 16}$ is defined by setting

$$\begin{aligned}\tilde{X}_i &= X_i & \text{for } i \neq 12, \\ \tilde{X}_{12} &= X_{12} + 1.\end{aligned}$$

In what follows we demonstrate how to choose X such that its MD4 hash value coincides with that of \tilde{X} , i.e.,

$$\text{compress}(IV_0; X) = \text{compress}(IV_0; \tilde{X}).$$

As in [5] the basic idea is that a (small) difference between only one of the input variables can be controlled in such a way that the differences occurring in the computations of the two associated hash values are compensated for at the end.

Our attack is separated into three parts. Each part considers a certain segment of the compression function. For $n < m < 48$ we therefore introduce the notation

$$\text{compress}_m^n((A, B, C, D); X_{\varphi(n)}, \dots, X_{\varphi(m)}) = (A', B', C', D')$$

for the segment of **compress** from step n to step m , where the mapping φ is defined such that $X_{\varphi(i)}$ is applied at the i th step of **compress**. This means that the computation of compress_m^n starts with the “initial value” (A, B, C, D) , then steps n to m are applied with the corresponding input words $X_{\varphi(n)}, \dots, X_{\varphi(m)}$, and the output (A', B', C', D') of compress_m^n is the contents of the registers after step m .

Sometimes, in the above notation, we write simply X instead of $X_{\varphi(n)}, \dots, X_{\varphi(m)}$. (However, thereby we do not necessarily assume that all X_i are actually defined.) We introduce another basic notation which is used in what follows. For $n \leq i \leq m$ let

$$(A_i, B_i, C_i, D_i) \text{ (resp. } (\tilde{A}_i, \tilde{B}_i, \tilde{C}_i, \tilde{D}_i))$$

be the contents of the registers after step i has been applied computing compress_m^n for the input X (resp. \tilde{X}). Further, we set

$$\Delta_i = (A_i - \tilde{A}_i, B_i - \tilde{B}_i, C_i - \tilde{C}_i, D_i - \tilde{D}_i).$$

Note that in each step only one of the registers is changed and that, for instance, $A_{4k} = A_{4k+1} = A_{4k+2} = A_{4k+3}$ ($k = 0, 1, \dots, 11$).

Part I: Inner Almost-Collisions (Steps 12–19)

For the attack against two-round RIPEMD [5], X_{13} was the selected variable instead of X_{12} and the first part was to find “inner collisions,” i.e., an initial value and inputs for (both lines of) $\text{compress}_{18}^{13}$ such that $\Delta_{18} = 0$. Since X_{13} occurs precisely in steps 13 and 18 of two-round RIPEMD-compress, this was a suitable approach to find collisions for two rounds. The main problem in the case of RIPEMD has been to handle its two parallel lines simultaneously.

However, here we deal with the three rounds of MD4. X_{12} appears in each round exactly once, namely, in steps 12, 19, and 35. X and \tilde{X} give a collision if (and only if) $\Delta_{35} = 0$, because X_{12} appears in step 35 the last time. To achieve this we require a certain well-chosen value for Δ_{19} , namely,

$$\Delta_{19} = (0, 1^{\ll 25}, -1^{\ll 5}, 0).$$

This means that the outputs of $\text{compress}_{19}^{12}$ for X and \tilde{X} are close but not equal. We are not looking for inner collisions, but for inner “almost”-collisions with precisely the output difference specified above.

Part II: Differential Attack Modulo 2^{32} (Steps 20–35)

The value for Δ_{19} of Part I has been carefully selected such that, with relatively high probability, this difference inherits to step 35 in a way that is compensated for by the difference between the inputs for this step, i.e., X_{12} and $X_{12} + 1$. (It should be emphasized that here the considered differences are not meant with respect to XOR but modulo 2^{32} .) Based on Part I we can therefore find collisions for the compression function of MD4.

Part III: Right Initial Value (Steps 0–11)

In the derived algorithm for finding collisions for the compression function there are still many variables free. Therefore it is very easy to get even collisions with an arbitrary prescribed initial value.

The next sections contain a detailed description of the three parts of our attack.

3. Inner Almost-Collisions (Steps 12–19)

In this section we consider $\text{compress}_{19}^{12}$, i.e., steps 12–19 of compress . Let (A, B, C, D) be the initial value of $\text{compress}_{19}^{12}$. Recall that $(A_{19}, B_{19}, C_{19}, D_{19})$ denotes the output of $\text{compress}_{19}^{12}$, i.e., the contents of the registers after step 19 for the inputs $X_{12}, X_{13}, X_{14}, X_{15}, X_0, X_4, X_8, X_{12}$ (resp. $(\tilde{A}_{19}, \tilde{B}_{19}, \tilde{C}_{19}, \tilde{D}_{19})$ for the inputs $X_{12} + 1, X_{13}, X_{14}, X_{15}, X_0, X_4, X_8, X_{12} + 1$).

We want to find inner almost-collisions that are explicit values for A, B, C, D and $X_{12}, X_{13}, X_{14}, X_{15}, X_0, X_4, X_8$ such that

$$\Delta_{19} = (0, 1^{25}, -1^5, 0).$$

The reason for this requirement will become clear in the next section. Tables 1 and 2 show the contents of the registers after application of steps $i = 12, \dots, 19$ for X_{12}, X_{13}, \dots and for $X_{12} + 1, X_{13}, \dots$, respectively. To simplify the notations we set $A_* = A_{19}$, $B_* = B_{19}, \dots, U = A_{12}, V = D_{13}, W = C_{14}, Z = B_{15}$, and $\tilde{U} = \tilde{A}_{12}, \tilde{V} = \tilde{D}_{13}, \tilde{W} = \tilde{C}_{14}, \tilde{Z} = \tilde{B}_{15}$.

Here we require $\tilde{B}_* + 1^{25} = B_*$ and $C_* + 1^5 = \tilde{C}_*$. The boxed entries are those

Table 1

Step	A	B	C	D	Input	Shift	Function	Constant
12	U	B	C	D	X_{12}	3	F	0
13	U	B	C	V	X_{13}	7	F	0
14	U	B	W	V	X_{14}	11	F	0
15	U	Z	W	V	X_{15}	19	F	0
16	A_*	Z	W	V	X_0	3	G	K_1
17	A_*	Z	W	D_*	X_4	5	G	K_1
18	A_*	Z	C_*	D_*	X_8	9	G	K_1
19	A_*	B_*	C_*	D_*	X_{12}	13	G	K_1

Table 2

Step	A	B	C	D	Input	Shift	Function	Constant
12	\tilde{U}	B	C	D	$X_{12} + 1$	3	F	0
13	\tilde{U}	B	C	\tilde{V}	X_{13}	7	F	0
14	\tilde{U}	B	\tilde{W}	\tilde{V}	X_{14}	11	F	0
15	\tilde{U}	\tilde{Z}	\tilde{W}	\tilde{V}	X_{15}	19	F	0
16	A_*	\tilde{Z}	\tilde{W}	\tilde{V}	X_0	3	G	K_1
17	A_*	\tilde{Z}	\tilde{W}	D_*	X_4	5	G	K_1
18	A_*	\tilde{Z}	\tilde{C}_*	D_*	X_8	9	G	K_1
19	A_*	\tilde{B}_*	\tilde{C}_*	D_*	$X_{12} + 1$	13	G	K_1

which have been modified in the particular steps, and the Boolean functions F and G are “selection” and “majority”:

$$\begin{aligned} F(U, V, W) &= (U \wedge V) \vee (\neg U \wedge W), \\ G(U, V, W) &= (U \wedge V) \vee (U \wedge W) \vee (V \wedge W), \end{aligned}$$

and $K_1 = 0x5a827999$ (see the Appendix). Finding an inner almost-collision is equivalent to finding a collection of solutions $B, C, A_*, B_*, C_*, D_*, U, V, W, Z, \tilde{U}, \tilde{V}, \tilde{W}, \tilde{Z}$ for the following system of equations:

$$1 = \tilde{U}^{\ll 29} - U^{\ll 29}, \quad (1)$$

$$F(\tilde{U}, B, C) - F(U, B, C) = \tilde{V}^{\ll 25} - V^{\ll 25}, \quad (2)$$

$$F(\tilde{V}, \tilde{U}, B) - F(V, U, B) = \tilde{W}^{\ll 21} - W^{\ll 21}, \quad (3)$$

$$F(\tilde{W}, \tilde{V}, \tilde{U}) - F(W, V, U) = \tilde{Z}^{\ll 13} - Z^{\ll 13}, \quad (4)$$

$$G(\tilde{Z}, \tilde{W}, \tilde{V}) - G(Z, W, V) = U - \tilde{U}, \quad (5)$$

$$G(A_*, \tilde{Z}, \tilde{W}) - G(A_*, Z, W) = V - \tilde{V}, \quad (6)$$

$$G(D_*, A_*, \tilde{Z}) - G(D_*, A_*, Z) = W - \tilde{W} + \tilde{C}_*^{\ll 23} - C_*^{\ll 23}, \quad (7)$$

$$G(\tilde{C}_*, D_*, A_*) - G(C_*, D_*, A_*) = Z - \tilde{Z} + \tilde{B}_*^{\ll 19} - B_*^{\ll 19} - 1, \quad (8)$$

where \tilde{B}_* stands for $B_* - 1^{\ll 25}$ and \tilde{C}_* stands for $C_* + 1^{\ll 5}$. Equations (1)–(8) simply follow by elimination of $X_{\varphi(i)}$ from the two equations defining steps $i = 12, \dots, 19$ of $\text{compress}_{19}^{12}$ for the inputs X and \tilde{X} . As an example, by the definition of step 15 we have

$$\begin{aligned} Z &= (B + F(W, V, U) + X_{15})^{\ll 19}, \\ \tilde{Z} &= (B + F(\tilde{W}, \tilde{V}, \tilde{U}) + X_{15})^{\ll 19}, \end{aligned}$$

implying (4). Conversely, if a collection of solutions of (1)–(8) is given, then we obtain

an inner almost-collision by setting

$$X_{13} = \text{arbitrary}, \quad (9)$$

$$X_{14} = W^{\ll 21} - C - F(V, U, B), \quad (10)$$

$$X_{15} = Z^{\ll 13} - B - F(W, V, U), \quad (11)$$

$$X_0 = A_*^{\ll 29} - U - G(Z, W, V) - K_1, \quad (12)$$

$$X_4 = D_*^{\ll 27} - V - G(A_*, Z, W) - K_1, \quad (13)$$

$$X_8 = C_*^{\ll 23} - W - G(D_*, A_*, Z) - K_1, \quad (14)$$

$$X_{12} = B_*^{\ll 19} - Z - G(C_*, D_*, A_*) - K_1, \quad (15)$$

$$D = V^{\ll 25} - F(U, B, C) - X_{13}, \quad (16)$$

$$A = U^{\ll 19} - F(B, C, D) - X_{12}. \quad (17)$$

The system (1)–(8) has 14 variables. Thus it is a natural idea to make settings for some of the variables such that finding a solution for the remaining variables is feasible. Therefore we set

$$\tilde{U} = -1 = 0xffffffff, \quad U = 0, \quad B = 0.$$

Then (1) is satisfied and (2), (3), (6), (7), (8) can be transformed and reordered as follows:

$$\tilde{Z} = Z - G(\tilde{C}_*, D_*, A_*) + G(C_*, D_*, A_*) + \tilde{B}_*^{\ll 19} - B_*^{\ll 19} - 1, \quad (18)$$

$$\tilde{W} = W - G(D_*, A_*, \tilde{Z}) + G(D_*, A_*, Z) + \tilde{C}_*^{\ll 23} - C_*^{\ll 23}, \quad (19)$$

$$V = W^{\ll 21} - \tilde{W}^{\ll 21}, \quad (20)$$

$$\tilde{V} = V - G(A_*, \tilde{Z}, \tilde{W}) + G(A_*, Z, W), \quad (21)$$

$$C = V^{\ll 25} - \tilde{V}^{\ll 25}. \quad (22)$$

For this system the variables A_* , B_* , C_* , D_* , Z , and W form free parameters for the set of all solutions. The two remaining equations, (4) and (5), are now

$$G(Z, W, V) - G(\tilde{Z}, \tilde{W}, \tilde{V}) = 1, \quad (23)$$

$$F(\tilde{W}, \tilde{V}, -1) - F(W, V, 0) - \tilde{Z}^{\ll 13} + Z^{\ll 13} = 0. \quad (24)$$

Algorithm Searching for Inner Almost-Collisions

After these preparations we can give an example of an algorithm leading to solutions of (1)–(8), that is, to an inner almost-collision, in about 1 second on a PC. The basic idea can be described as a kind of “continuous approximation” (see Section 4 of [5]).

1. Choose A_* , B_* , C_* , D_* , Z , W randomly, compute \tilde{Z} , \tilde{W} , V , \tilde{V} according to (18)–(21) and test (23). If the test is passed goto 2. (Since W and \tilde{W} (resp. Z and \tilde{Z}) are close with respect to Hamming distance, there is a relatively high probability that (23) is satisfied.)
2. Take A_* , B_* , C_* , D_* , Z , W found in 1 as “basic values.” Change one random bit in each of these variables, compute the associated \tilde{Z} , \tilde{W} , V , \tilde{V} , and test if (23) is still satisfied and if, moreover, the right four bits of

$$F(\tilde{W}, \tilde{V}, -1) - F(W, V, 0) - \tilde{Z}^{\ll 13} + Z^{\ll 13} \quad (25)$$

are zero. If this test is passed take the corresponding values A_*, B_*, C_*, D_*, Z, W as the new “basic values.” Next do the same as before. If it appears that even the eight right bits of (25) are zero, then the next phase of the algorithm is reached. Now new basic values are taken only if the eight right bits of (25) are zero. Continue with making the 12, 16, . . . right bits zero until (24) is fulfilled.

3. Now (23) and (24) are satisfied, and we obtain an inner almost-collision by setting $B = 0$ and defining A, C, D , and X_i ($i = 0, 4, 8, 12, 13, 14, 15$) according to (9)–(17) and (22).

In order that the inner almost-collision can be used for the differential attack explained in the next section, the following additional equation has to be satisfied:

$$G(B_*, C_*, D_*) = G(\tilde{B}_*, \tilde{C}_*, D_*). \quad (26)$$

Since \tilde{B}_* and B_* (resp. \tilde{C}_* and C_*) are close, there is a high probability that this condition is true. Thus, to achieve (26) also, step 2 above has to be repeated a few times. (To be more precise, nine times on average, as we shall see in the next section.)

We call an inner almost-collision *admissible* if (26) is satisfied. Using again our original notation in (26) we can summarize the result of this section as follows:

Lemma 1. *There is a practical algorithm, which allows us to compute an admissible inner almost-collision, i.e., an initial value (A, B, C, D) and inputs $X_{12}, X_{13}, X_{14}, X_{15}, X_0, X_4, X_8$ for $\text{compress}_{19}^{12}$ such that we have*

$$\begin{aligned} \Delta_{19} &= (0, 1^{\ll 25}, -1^{\ll 5}, 0), \\ G(B_{19}, C_{19}, D_{19}) &= G(\tilde{B}_{19}, \tilde{C}_{19}, D_{19}). \end{aligned}$$

The computation requires less than 1 second on a PC.

4. Differential Attack Modulo 2^{32} (Steps 20–35)

The main part of the work has been done in the preceding section. We are now well prepared for a routine differential attack, which will allow us to find collisions for the compress function of MD4. Using the notation introduced in Section 2 we can state this result as follows:

Lemma 2. *Suppose that an admissible inner almost-collision, i.e., an initial value (A, B, C, D) for step 12 and variables $X_{12}, X_{13}, X_{14}, X_{15}, X_0, X_4, X_8$ are given according to Lemma 1. Choose the remaining X_i ’s randomly and determine the corresponding initial value by computing compress_{11}^0 backward starting with*

$$(A_{11}, B_{11}, C_{11}, D_{11}) = (A, B, C, D).$$

Then the probability that X and \tilde{X} form a collision for the compression function of MD4 (i.e., $\Delta_{35} = 0$) is about 2^{-22} .

Table 3

Step i	Δ_i^*			Function	Shift	p_i^{i-1}	Input	Constant
19	0	$1 \ll 25$	$-1 \ll 5$	0	*	*	*	*
20	0	$1 \ll 25$	$-1 \ll 5$	0	G	3	1	X_1
21	0	$1 \ll 25$	$-1 \ll 5$	0	G	5	$1/9$	X_5
22	0	$1 \ll 25$	$-1 \ll 14$	0	G	9	$1/3$	X_9
23	0	$1 \ll 6$	$-1 \ll 14$	0	G	13	$1/3$	X_{13}
24	0	$1 \ll 6$	$-1 \ll 14$	0	G	3	$1/9$	X_2
25	0	$1 \ll 6$	$-1 \ll 14$	0	G	5	$1/9$	X_6
26	0	$1 \ll 6$	$-1 \ll 23$	0	G	9	$1/3$	X_{10}
27	0	$1 \ll 19$	$-1 \ll 23$	0	G	13	$1/3$	X_{14}
28	0	$1 \ll 19$	$-1 \ll 23$	0	G	3	$1/9$	X_3
29	0	$1 \ll 19$	$-1 \ll 23$	0	G	5	$1/9$	X_7
30	0	$1 \ll 19$	-1	0	G	9	$1/3$	X_{11}
31	0	1	-1	0	G	13	$1/3$	X_{15}
32	0	1	-1	0	H	3	$1/3$	X_0
33	0	1	-1	0	H	9	$1/3$	X_8
34	0	1	0	0	H	11	$1/3$	X_4
35	0	0	0	0	H	15	1	$X_{12}(+1)$

Proof. Let p be the probability that $\Delta_{35} = 0$ under the given assumption. We have to confirm that

$$p \approx 2^{-22}.$$

Table 3 defines a sequence of fixed values Δ_i^* ($i = 19, \dots, 35$) for differences starting with $\Delta_{19}^* = (0, 1 \ll 25, -1 \ll 5, 0)$ and ending with $\Delta_{35}^* = 0$. The boxed entries correspond to those variables which are modified in the particular steps. The Boolean functions G and H are majority and XOR, respectively. Here p_i^j ($i > j$) denotes the probability that $\Delta_i = \Delta_i^*$ under the assumption that $\Delta_j = \Delta_j^*$. The asterisk entries for step 19 mean that we do not refer to these values in our argumentation. Note that $\Delta_{19} = \Delta_{19}^*$, since an inner almost-collision is given. We have $\Delta_{20} = \Delta_{20}^*$ and therefore $p_{20}^{19} = 1$, because the given inner almost-collision is admissible. To verify $p_{35}^{34} = 1$ note that $\Delta_{34} = (0, 1, 0, 0) = \Delta_{34}^*$ implies

$$\begin{aligned}
 B_{35} &= (B_{34} + H(C_{34}, D_{34}, A_{34}) + X_{12} + K_2)^{\ll 15} \\
 &= ((\tilde{B}_{34} + 1) + H(\tilde{C}_{34}, \tilde{D}_{34}, \tilde{A}_{34}) + X_{12} + K_2)^{\ll 15} \\
 &= (\tilde{B}_{34} + H(\tilde{C}_{34}, \tilde{D}_{34}, \tilde{A}_{34}) + (X_{12} + 1 + K_2))^{\ll 15} \\
 &= \tilde{B}_{35}.
 \end{aligned}$$

Also each of the other given values for p_i^{i-1} can be proved easily. As an example, for $i = 32$ we have to show that $(R + 1) \oplus S = R \oplus (S + 1)$ holds with probability $1/3$ for independent random words R, S . This equation is satisfied if and only if exactly one of the following conditions for the binary representations of R and S is given:

$$\begin{array}{llll} R = *0 & \text{and} & S = *0, \\ R = *01 & \text{and} & S = *01, \\ R = *011 & \text{and} & S = *011, \\ \vdots & & \vdots \\ R = 01 \cdots 11 & \text{and} & S = 01 \cdots 11, \\ R = 1 \cdots 11 & \text{and} & S = 1 \cdots 11. \end{array}$$

Here an asterisk marks an arbitrary bit sequence of suitable length. (These sequences do not have to coincide for R and S in the particular cases.) Thus we conclude

$$p_{32}^{31} = \frac{1}{2^2} + \frac{1}{4^2} + \frac{1}{8^2} + \cdots + \frac{1}{2^{62}} + \frac{1}{2^{64}} + \frac{1}{2^{64}} = \frac{1}{3} \left(1 + \frac{1}{2^{63}} \right).$$

The above table yields

$$\prod_{i=20}^{35} p_i^{i-1} = 2^{-30.11}.$$

This already indicates that p is large enough for a practical attack. Since the conditions $\Delta_i = \Delta_i^*$ are strongly dependent, we obtain a much more realistic approximation for p if we consider four steps at once. The values for p_i^{i-4} can certainly be computed similarly as p_i^{i-1} . However, this seems to require lengthy considerations of various cases. The values in Table 4 have been found by a simple Monte Carlo method.

Now we get

$$p_{23}^{19} p_{27}^{23} p_{31}^{27} p_{35}^{31} = 2^{-24.54}.$$

This is a much better approximation for p . Experimental observations suggest that p is in fact still larger. We found the estimation $p \approx 2^{-22}$. \square

Thus we have shown that a random choice of the nine free X_i 's gives a collision of the compression function with probability 2^{-22} . Therefore, in principle, each given inner

Table 4

Step i		Δ_i^*		p_i^{i-4}
19	0	$1 \ll 25$	$-1 \ll 5$	0
23	0	$1 \ll 6$	$-1 \ll 14$	0
27	0	$1 \ll 19$	$-1 \ll 23$	0
31	0	1	-1	0
35	0	0	0	1/7

almost-collision allows us to find on average about 2^{266} collisions for the compression functions.

Actually we need much less than 2^{22} trials to find a collision for the compression function. The reason is that we do not have to start each trial from the beginning. If X_1, X_5, X_9 have been found such that together with the already fixed X_{13} we have reached the required difference for step 23, then keep X_1, X_5, X_9 . Next choose suitable values for X_2, X_6, X_{10} , and so on. In this way we can find a collision for the compression function in a small fraction of a second on a PC.

5. Right Initial Value (Steps 0–11)

It remains to compute collisions with the initial value IV_0 required by the definition of MD4. By Lemma 2 there are enough variables free to manage this easily. (The following argumentation does of course not depend on the particular choice of IV_0 .)

Suppose an admissible inner almost-collision with initial value (A, B, C, D) is given. Take random X_1, X_2, X_3 , and X_5 . Recall that X_0, X_4 , and X_8 are already fixed. Compute $\text{compress}_5^0(IV_0; X_0, \dots, X_5)$. Now $A_5 = A_4, B_5 = B_4 = B_3, C_5 = C_4 = C_3 = C_2, D_5$ are fixed.

Next we define X_6, X_7, X_9, X_{10} , and X_{11} such that the output of

$$\text{compress}_{11}^0(IV_0; X_0, \dots, X_{11})$$

matches with (A, B, C, D) , or, in other words,

$$\text{compress}_{11}^6((A_4, B_3, C_2, D_5); X_6, \dots, X_{11}) = (A, B, C, D).$$

Matching B, C , and D can be done directly by associating suitable values to the free variables X_{11}, X_{10} , and X_9 , respectively. It remains to match A in step 8. This cannot be done as before, since X_8 is already fixed. Step 8 is defined by the equation

$$A_8 = (A_4 + F(B_7, C_6, D_5) + X_8)^{\ll 3}.$$

According to the definition of F as a selection function, we achieve $A_8 = A$ if $B_7 = -1 = 0xffffffff$ and $C_6 = A^{\ll 29} - A_4 - X_8$. These values for C_6 and B_7 can be obtained by a suitable choice of X_6 and X_7 .

Explicitly, this simple idea leads to the settings:

$$\begin{aligned} X_6 &:= -C_2 - F(D_5, A_4, B_3) + (A^{\ll 29} - A_4 - X_8)^{\ll 21}, \\ C_6 &= (C_2 + F(D_5, A_4, B_3) + X_6)^{\ll 11} = A^{\ll 29} - A_4 - X_8, \\ X_7 &:= -B_3 - F(C_6, D_5, A_4) - 1, \\ B_7 &= (B_3 + F(C_6, D_5, A_4) + X_7)^{\ll 19} = -1, \\ A_8 &= (A_4 + F(-1, C_6, D_5) + X_8)^{\ll 3} = (A_4 + C_6 + X_8)^{\ll 3} = A, \end{aligned}$$

$$\begin{aligned}
X_9 &:= D^{\ll 25} - D_5 - F(A, -1, C_6), \\
D_9 &= (D_5 + F(A, -1, C_6) + X_9)^{\ll 7} = D, \\
X_{10} &:= C^{\ll 21} - C_6 - F(D, A, -1), \\
C_{10} &= (C_6 + F(D, A, -1) + X_{10})^{\ll 11} = C, \\
X_{11} &:= B^{\ll 13} + 1 - F(C, D, A), \\
B_{11} &= (-1 + F(C, D, A) + X_{11})^{\ll 19} = B.
\end{aligned}$$

This means we obtain

$$\begin{aligned}
\text{compress}_{11}^0(IV_0; X_0, \dots, X_{11}) &= (A_{11}, B_{11}, C_{11}, D_{11}) = (A_8, B_11, C_{10}, D_9) \\
&= (A, B, C, D),
\end{aligned}$$

i.e., as desired, we have reached the connection to the given inner almost-collision.

6. Collision Search Algorithm

As we have now described all parts of the attack, we give an overview summarizing the single steps of the derived algorithm searching for collisions:

1. *Compute A, B, C, D and $X_0, X_4, X_8, X_{12}, X_{13}, X_{14}, X_{15}$, which give an inner almost-collision (from steps 12 to 19). The technical details of a suitable algorithm have been explained in Section 3. It also fixes values for $A_{19}, B_{19}, C_{19}, D_{19}$ and $\tilde{A}_{19}, \tilde{B}_{19}, \tilde{C}_{19}, \tilde{D}_{19}$.*

2. *According to Sections 4 and 5 choose X_1, X_2, X_3, X_5 randomly and compute*

$$(A_5, B_5, C_5, D_5) = \text{compress}_5^0(IV_0; X_0, \dots, X_5), \quad (27)$$

$$t = A^{\ll 29} - A_5 - X_8, \quad (28)$$

$$X_6 = t^{\ll 21} - C_5 - F(D_5, A_5, B_5), \quad (29)$$

$$X_7 = -1 - B_5 - F(t, D_5, A_5), \quad (30)$$

$$X_9 = D^{\ll 25} - D_5 - F(A, -1, t), \quad (31)$$

$$X_{10} = C^{\ll 21} - t - F(D, A, -1), \quad (32)$$

$$X_{11} = B^{\ll 13} + 1 - F(C, D, A), \quad (33)$$

$$(A_{35}, B_{35}, C_{35}, D_{35}) = \text{compress}_{35}^{20}(A_{19}, B_{19}, C_{19}, D_{19}; X), \quad (34)$$

$$(\tilde{A}_{35}, \tilde{B}_{35}, \tilde{C}_{35}, \tilde{D}_{35}) = \text{compress}_{35}^{20}(\tilde{A}_{19}, \tilde{B}_{19}, \tilde{C}_{19}, \tilde{D}_{19}; \tilde{X}), \quad (35)$$

$$\Delta_{35} = (A_{35}, B_{35}, C_{35}, D_{35}) - (\tilde{A}_{35}, \tilde{B}_{35}, \tilde{C}_{35}, \tilde{D}_{35}). \quad (36)$$

3. *If $\Delta_{35} = 0$, then we have found a collision. Otherwise make a new trial by going to 2.*

Tuning, Computational Effort, and Example

To make 2 more effective do not compute the compression function from steps 20 to 35 completely. Instead, as a condition to break up the trial, test immediately after each step i if $\Delta_i \neq \Delta_i^*$ ($i = 21, 22, \dots$); see Table 3.

In this way for the largest part of all trials the computation will already be broken up after step 21, i.e., it is restricted to (27)–(33) and two steps of (34) and (35), respectively. (On the other hand, it is unlikely that $\Delta_{35} = 0$ but $\Delta_i \neq \Delta_i^*$ for some i . The sequence $\Delta_{19}^*, \dots, \Delta_{34}^*$ in Section 3 is the “almost unique” way leading to $\Delta_{35} = 0$. If we leave this way, then it is very likely that the avalanche effect brings everything out of control. Hence it is unlikely that we lose a successful trial by the proposed selection.) Thus mainly a trial requires about the same effort as 16 steps (one-third) of MD4-compress.

In view of Lemma 2 this means that finding a collision takes on average the same computational effort as about 2^{20} computations of MD4-compress. This estimation has been confirmed by an implementation of the attack.

The algorithm sometimes runs into a dead end. For instance, this happens if the values $A_{19}, B_{19}, C_{19}, D_{19}$ and $\tilde{A}_{19}, \tilde{B}_{19}, \tilde{C}_{19}, \tilde{D}_{19}$ coming from the inner almost-collision are badly conditioned with respect to the differential attack. This effect can be taken into account by controlling the success of the algorithm and making a new start if necessary.

Beside the complexity of a collision search algorithm, the “variety” of collisions which, at least theoretically, can be found is another important aspect. In particular, the number of obtainable collisions is of interest. We therefore mention that by Lemma 2 for each found inner almost-collision, in principle, about 2^{106} collisions of MD4 can be computed by applying the above algorithm, since there are four words free in 2 (i.e., 2^{128} trials), and the probability for success is about 2^{-22} .

For the sake of readability of our exposition we have restricted ourselves to the description of the most direct version of our attack. However, there are still many further technical tricks to improve it considerably. In this way we can even get collisions of practical relevance. This is demonstrated in the next section. First, however, we give a collision which has been found by the previously described search algorithm:

$$\begin{aligned} X_0 &= 0x13985e12, & X_8 &= 0xabe17be0, \\ X_1 &= 0x748a810b, & X_9 &= 0xed1ed4b3, \\ X_2 &= 0x4d1df15a, & X_{10} &= 0x4120abf5, \\ X_3 &= 0x181d1516, & X_{11} &= 0x20771029, \\ X_4 &= 0x2d6e09ac, & X_{12} &= 0x20771027, \\ X_5 &= 0x4b6dbdb9, & X_{13} &= 0xfdffffbfff, \\ X_6 &= 0x6464b0c8, & X_{14} &= 0xfffffbff, \\ X_7 &= 0xfba1c097, & X_{15} &= 0x6774bed2. \end{aligned}$$

Recall that \tilde{X} is defined by setting $\tilde{X}_i = X_i$ ($i < 16, i \neq 12$) and

$$\tilde{X}_{12} = X_{12} + 1 = 0x20771028.$$

X and \tilde{X} have the same MD4-compression value with respect to the initial value IV_0 (see the Appendix). The complete MD4 algorithm, including the padding rule, associates to X and \tilde{X} the common hash value

$$0x711ad51b 0xbbab5e22 0x618b1c76 0x17c15892.$$

7. Collisions for Crooks

How to Swindle Ann (see [12])

Alf wanted to sell Ann his house, and Ann was interested. They agreed on a price of \$176,495. Alf asked Ann to sign a contract using a digital signature scheme which is based on some public-key algorithm and the hash function MD4. The contract read as follows:

CONTRACT
 At the price of \$176,495 Alf Blowfish
 sells his house to Ann Bonidea....

“The first 20 bytes (each of them is represented by an asterisk above) are random. They have been placed before the text for security reasons!” claimed Alf, and Ann signed the contract. Later, however, Alf substituted the contract file by another which read as follows:

CONTRACT
 At the price of \$276,495 Alf Blowfish
 sells his house to Ann Bonidea....

The contract had been prepared by him such that replacing \$176,495 by \$276,495 does not change the MD4 hash value!

How Alf Did It

We now explain the precise definition of the above digital contract. Its first sixteen 32-bit words are:

$$\begin{array}{ll}
 M_0 = 0x9074449b, & M_8 = 0x68742074, \\
 M_1 = 0x1089fc26, & M_9 = 0x72702065, \\
 M_2 = 0x8bf37fa2, & M_{10} = 0x20656369, \\
 M_3 = 0x1d630daf, & M_{11} = 0x2420666f, \\
 M_4 = 0x63247e24, & M_{12} = 0x2c363731, \\
 M_5 = 0x4e4f430a, & M_{13} = 0x20353934, \\
 M_6 = 0x43415254, & M_{14} = 0x20666c41, \\
 M_7 = 0x410a0a54, & M_{15} = 0x776f6c42.
 \end{array}$$

The 20 bytes of M_0 – M_4 are the above mentioned “random bytes.” The bytes of M_5 , in reverse ordering (according to the definition of MD4) and interpreted as ASCII read as follows:

0a 43 4f 4e = *Line feed ‘CON’*,

and so on to M_{15} which reads

42 6c 6f 77 = *‘Blow’*.

The sequence M_i ($i < 16$) has been chosen such that setting $M'_{12} = M_{12} + 1$ and $M'_i = M_i$ for $i < 16$, $i \neq 12$ gives a collision, i.e.,

compress($IV_0; M$) = **compress**($IV_0; M'$)

for the compression function of MD4 and its fixed initial value IV_0 . This collision has been found in less than 1 hour on a PC. Interpreting $M_{12} = 0x2c363731$ and $M'_{12} = 0x2c363732$ we get

$M_{12} = 31\ 37\ 36\ 2c = ‘176,’$
 $M'_{12} = 32\ 37\ 36\ 2c = ‘276,’$

In view of the definition of MD4 as the iterative application of **compress** we obtain a collision by taking any bit string and appending it to M and M' .

8. The 256-Bit Extension of MD4

Together with MD4 Rivest [8] also proposed an extension of MD4 with 256-bit hash values for highest security requirements. The compression function **compress-ext** of this Extended MD4 is designed as follows: Two copies of MD4 compress run in parallel. The first copy is standard MD4 compress. The second copy differs only in the choice of the constants. For the first (resp. second) copy the following constants K_i (resp. K'_i) are used in round i :

$$\begin{array}{ll} K_1 = 0, & K'_1 = 0, \\ K_2 = 0x5a827999, & K'_2 = 0x50a28be6, \\ K_3 = 0x6ed9eba1, & K'_3 = 0x5c4dd124. \end{array}$$

After computing each copy separately, the values of the A registers in the two copies are exchanged. Extended MD4 is defined as the iterative application of **compress-ext** starting with the initial value $IV^* = (IV_0^*, IV_1^*)$, where

$$\begin{aligned} IV_0^* &= 0x67452301\ 0xefcdab89\ 0x98badcfe\ 0x10325476, \\ IV_1^* &= 0x33221100\ 0x77665544\ 0xbbaa9988\ 0xffeeddcc. \end{aligned}$$

The following example of a collision for **compress-ext** was found by using methods as for the RIPEMD and MD4 attacks. *The reason for the weakness of Extended MD4 and RIPEMD can be clearly identified: The two parallel lines of the compression function*

are too similar. The ordering in which the words are applied in the single steps is the same. This allows both lines to be attacked in parallel.

Set $IV = (IV_0, IV_1)$ with

$$IV_0 = IV_1 = 0x3106724a 0x187c28f6 0x6db5f180 0xafdad375.$$

Define the inputs

$$\begin{aligned} X_0 &= 0x51737d99, & X_8 &= 0xfec3fc24, \\ X_1 &= 0x527507ef, & X_9 &= 0x74fdd294, \\ X_2 &= 0x69ea5e67, & X_{10} &= 0x28566835, \\ X_3 &= 0x6a7e3c3d, & X_{11} &= 0x0ec55879, \\ X_4 &= 0x8171ebe6, & X_{12} &= 0x9a213c15, \\ X_5 &= 0x453ef355, & X_{13} &= 0x2069ff64, \\ X_6 &= 0x0535803b, & X_{14} &= 0xfffffbff, \\ X_7 &= 0x2c885e93, & X_{15} &= 0x2fa86b00. \end{aligned}$$

Define \tilde{X} by setting $\tilde{X}_i = X_i$ ($i < 16, i \neq 12$) and $\tilde{X}_{12} = X_{12} + 1$. Then we have

$$\text{compress-ext}(IV; X) = \text{compress-ext}(IV; \tilde{X}).$$

Computation of collisions of the compression function of Extended MD4 with $IV_0 = IV_1$ requires about the same effort as 2^{26} compression computations (resp. 2^{40} compression computations if the value for $IV_0 = IV_1$ is prescribed). We anticipate that a more sophisticated attack allows us to find collisions with the prescribed initial value IV^* above (i.e., collisions for Extended MD4) by $O(2^{40})$ operations.

The design of RIPEMD and Extended MD4 are very similar. Therefore the demonstrated weakness of Extended MD4 supports the proposal to replace RIPEMD by a strengthened version (see [7]). However, this old version of RIPEMD should not be confused with its successor RIPEMD-160 [7] with 160-bit hash values or with the new RIPEMD-128, the plug-in substitute for RIPEMD with a 128-bit result. RIPEMD-128/160 were designed by taking account of the experiences made by the analysis of MD4, Extended MD4, MD5, and RIPEMD.

9. Conclusions

A dedicated hash function should be secure and fast at the same time. Everyone who comes up with a new design of a fast algorithm, especially if there is insufficient experience with related algorithms, runs a great risk of overlooking weaknesses and underestimating possibilities of finding new cryptanalytic methods. However, there is no other way than to start with concrete proposals, thereby pushing on an evolutionary process leading to better and better solutions. Therefore the introduction of MD4 by Rivest [8] in 1990 was a significant contribution. Today there is a whole family of hash functions based on MD4's design elements.

A short time after MD4 had been introduced, some weaknesses became apparent and Rivest introduced MD5 in 1991. He explained his reasons in [9]:

“The MD5 algorithm is an extension of the MD4 message-digest algorithm. MD5 is slightly slower than MD4, but is more conservative in design. MD5 was designed because it was felt that MD4 was perhaps being adopted for use more quickly than justified by the existing critical review; because MD4 was designed to be exceptionally fast, it is at the edge in terms of risking successful cryptanalytic attack. . . .”

The weaknesses of MD4 observed in [3] and [11], two-round attacks and almost collisions, were generally considered to be mainly of theoretical importance. In view of the presented attack this can no longer be assumed, as has been demonstrated.

Where MD4 is still in use, it should be replaced!

Even the one-wayness of MD4 is set in question by a recent result showing that preimages for the first two rounds can be found very fast [6].

The compression function of the 256-bit extension of MD4 (see [8]) is not collision-free as shown in Section 8. RIPEMD is another strengthened mode of MD4 proposed in 1992 [2]. The design of RIPEMD and that of Extended MD4 are very similar. We anticipate that, in addition to the already known two-round attacks [4], the compression function of RIPEMD is also not collision-free. Also the compression function of MD5 is not collision-free [4].

However, the consequence of our analysis of (Extended) MD4, MD5, and RIPEMD is not that any hash functions, whose design is based on MD4, is compromised (similar as, say, the fact that eight round DES is weak does not mean that all Feistel ciphers are weak). On the contrary, the basic design principles of MD4 are today well analyzed, we have learned how to avoid weaknesses and to estimate better how many rounds are needed. The result is the hash function RIPEMD-160 [7] (with 160-bit hash values), which we would suggest as a replacement for MD4, MD5, and RIPEMD. Another alternative is the revised version of the Secure Hash Algorithm (SHA-1), which was designed by NSA and published by NIST (National Institute of Standards and Technology) [1]. The design criteria of SHA-1 are secret. SHA-1 and RIPEMD-160 are also recommended by RSA Data Security Inc. for applications which require a collision-free hash function [10].

Acknowledgment

The author thanks Antoon Bosselaers for reading an earlier version of this paper very carefully and giving several hints improving the exposition.

Appendix

The hash function MD4 is defined as the iteration of a certain compression function, which we specify below. The computation starts with the initial value

$IV_0 = 0x67452301\ 0xefcdab89\ 0x98badcfe\ 0x10325476.$

Each application of the compression function uses a collection of four words as initial value and sixteen words of the message as input, and it gives four words output, which are then used as the initial value 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 $512 = 16 \times (\text{length of words})$). A description of MD4 including also the padding rule can be found in [8].

The compression function of MD4 uses the Boolean vector functions

$$\begin{aligned} F(U, V, W) &= (U \wedge V) \vee (\neg U \wedge W), \\ G(U, V, W) &= (U \wedge V) \vee (U \wedge W) \vee (V \wedge W), \\ H(U, V, W) &= U \oplus V \oplus W \end{aligned}$$

and the constants

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

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 the MD4 compression function suppose now that the initial value (A, B, C, D) and inputs X_0, X_1, \dots, X_{15} are given. Copy A, B, C, D into registers a, b, c, d , and apply the following steps:

First round

- | | |
|---------|------------------------------|
| step 0 | $FF(a, b, c, d, X_0, 3)$ |
| step 1 | $FF(d, a, b, c, X_1, 7)$ |
| step 2 | $FF(c, d, a, b, X_2, 11)$ |
| step 3 | $FF(b, c, d, a, X_3, 19)$ |
| step 4 | $FF(a, b, c, d, X_4, 3)$ |
| step 5 | $FF(d, a, b, c, X_5, 7)$ |
| step 6 | $FF(c, d, a, b, X_6, 11)$ |
| step 7 | $FF(b, c, d, a, X_7, 19)$ |
| step 8 | $FF(a, b, c, d, X_8, 3)$ |
| step 9 | $FF(d, a, b, c, X_9, 7)$ |
| step 10 | $FF(c, d, a, b, X_{10}, 11)$ |
| step 11 | $FF(b, c, d, a, X_{11}, 19)$ |
| step 12 | $FF(a, b, c, d, X_{12}, 3)$ |
| step 13 | $FF(d, a, b, c, X_{13}, 7)$ |
| step 14 | $FF(c, d, a, b, X_{14}, 11)$ |
| step 15 | $FF(b, c, d, a, X_{15}, 19)$ |

Second round

- step 16 $GG(a, b, c, d, X_0 + K_1, 3)$
- step 17 $GG(d, a, b, c, X_4 + K_1, 5)$
- step 18 $GG(c, d, a, b, X_8 + K_1, 9)$
- step 19 $GG(b, c, d, a, X_{12} + K_1, 13)$
- step 20 $GG(a, b, c, d, X_1 + K_1, 3)$
- step 21 $GG(d, a, b, c, X_5 + K_1, 5)$
- step 22 $GG(c, d, a, b, X_9 + K_1, 9)$
- step 23 $GG(b, c, d, a, X_{13} + K_1, 13)$
- step 24 $GG(a, b, c, d, X_2 + K_1, 3)$
- step 25 $GG(d, a, b, c, X_6 + K_1, 5)$
- step 26 $GG(c, d, a, b, X_{10} + K_1, 9)$
- step 27 $GG(b, c, d, a, X_{14} + K_1, 13)$
- step 28 $GG(a, b, c, d, X_3 + K_1, 3)$
- step 29 $GG(d, a, b, c, X_7 + K_1, 5)$
- step 30 $GG(c, d, a, b, X_{11} + K_1, 9)$
- step 31 $GG(b, c, d, a, X_{15} + K_1, 13)$

Third round

- step 32 $HH(a, b, c, d, X_0 + K_2, 3)$
- step 33 $HH(d, a, b, c, X_8 + K_2, 9)$
- step 34 $HH(c, d, a, b, X_4 + K_2, 11)$
- step 35 $HH(b, c, d, a, X_{12} + K_2, 15)$
- step 36 $HH(a, b, c, d, X_2 + K_2, 3)$
- step 37 $HH(d, a, b, c, X_{10} + K_2, 9)$
- step 38 $HH(c, d, a, b, X_6 + K_2, 11)$
- step 39 $HH(b, c, d, a, X_{14} + K_2, 15)$
- step 40 $HH(a, b, c, d, X_1 + K_2, 3)$
- step 41 $HH(d, a, b, c, X_9 + K_2, 9)$
- step 42 $HH(c, d, a, b, X_5 + K_2, 11)$
- step 43 $HH(b, c, d, a, X_{13} + K_2, 15)$
- step 44 $HH(a, b, c, d, X_3 + K_2, 3)$
- step 45 $HH(d, a, b, c, X_{11} + K_2, 9)$
- step 46 $HH(c, d, a, b, X_7 + K_2, 11)$
- step 47 $HH(b, c, d, a, X_{15} + K_2, 15)$

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

$$AA = A + a, \quad BB = B + b, \quad CC = C + c, \quad DD = D + d.$$

That is, one sets

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

References

- [1] FIPS 180-1, *Secure Hash Standard*, Federal Information Processing Standard, NIST, US Department of Commerce, Washington D.C., April 1995.
- [2] RIPE, *Integrity Primitives for Secure Information Systems. Final Report of RACE Integrity Primitives Evaluation (RIPE-RACE 1040)*, Lecture Notes in Computer Science, vol. 1007, Springer-Verlag, Berlin, 1995.
- [3] den Boer, B., and Bosselaers, A.: 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.
- [4] Dobbertin, H.: The status of MD5 after a recent attack, *CryptoBytes* (the Technical Newsletter of RSA Laboratories), vol. 2 (1996), pp. 1–6.
- [5] Dobbertin, H.: RIPEMD with two-round compress function is not collision-free, *J. Cryptology*, vol. 10 (1997), pp. 51–68.
- [6] Dobbertin, H.: The first two rounds of MD4 are not one-way, preprint, March 1997.
- [7] Dobbertin, H., Bosselaers, A., and Preneel, B.: RIPEMD-160: A strengthened version of RIPEMD, *Cambridge Workshop on Cryptographic Algorithms*, 1996, Lecture Notes in Computer Science, vol. 1039, Springer-Verlag, Berlin, 1996, pp. 71–82.
- [8] Rivest, R.: The MD4 message-digest algorithm, Request for Comments (RFC) 1320, Internet Activities Board, Internet Privacy Task Force, April 1992.
- [9] Rivest, R.: The MD5 message-digest algorithm, Request for Comments (RFC) 1321, Internet Activities Board, Internet Privacy Task Force, April 1992.
- [10] Robshaw, M.J.B.: On recent results on MD2, MD4 and MD5, Bulletin 4, RSA Laboratories, November 1996 (see <http://www.rsa.com/PUBS/>).
- [11] Vaudenay, S.: 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.
- [12] Yuval, G.: How to swindle Rabin, *Cryptologia*, vol. 3, no. 3 (1979), pp. 187–189.