The First Two Rounds of MD4 are Not One-Way

Extended Abstract

Hans Dobbertin

German Information Security Agency
P. O. Box 20 03 63
D-53133 Bonn

dobbertin@skom.rhein.de

In [1] it was shown that there are very effective attacks leading to collisions for the hash function MD4 designed by R. Rivest [3]. A summary of the status of hash functions of the MD4-family with respect to collision-resistence can be found in [2] and [4]. However, attacking the one-wayness of a hash function is a much more demanding challenge, and in case of success it has much more devastating consequences. No result along this line is known for MD4 and its successors. Therefore it is worth to explore how the recently developed new analytic methods for finding collisions can be applied to construct preimages or second preimages. As a first step, we state here the following partial result:

Denote by $\mathsf{MD4}^{[12]}$ the reduced version of MD4, where the third round of its underlying three-round compression function is cancelled, but everything else of its specification is kept (e.g. initial value, padding rule).

$$\boxed{\mathsf{MD4}^{[12]} \textit{ is not one-way.}}$$

It takes less than one hour to find preimages on a PC. Second preimages take a few minutes or even less than a millisecond if the initial value is free.

Example. Assume we want to find a preimage of

$$V = \texttt{0x00000000 0x00000000 0x00000000 0x00000000.}$$

We have constructed the following message $M = M_0||M_1||...||M_{28}$, which is hashed to this value V by $\mathsf{MD4}^{[12]}$:

$M_0 =$ 0xB5B6AC17	$M_8 =$ 0xFF9405C3	$M_{16} =$ 0x814F4825	$M_{24} =$ 0x847064AD
$M_1 =$ 0xB5B6AC17	$M_9 =$ 0xFF9405C3	$M_{17} =$ 0x814F4825	$M_{25} =$ 0x05DDD0F5
$M_2 =$ 0xB5B6AC17	$M_{10} =$ 0xFF9405C3	$M_{18} =$ 0x814F4825	$M_{26} =$ 0xD462FA71
$M_3 =$ 0x85574A58	$M_{11} =$ 0xC26EA1D5	$M_{19} =$ 0x814F4825	$M_{27} =$ 0x56A79DEC
$M_4 =$ 0x4353212D	$M_{12} =$ 0x015CB5D0	$M_{20} =$ 0x9919C508	$M_{28} =$ 0x00000080
$M_5 =$ 0x4353212D	$M_{13} =$ 0x81BBD193	$M_{21} =$ 0x9919C508	
$M_6 =$ 0x4353212D	$M_{14} =$ 0x1DEF9763	$M_{22} =$ 0x9919C508	
$M_7 =$ 0x3E30333E	$M_{15} =$ 0xADE9028B	$M_{23} =$ 0x2FD7B0F9	

We anticipate that a similar attack works for the last two rounds of MD4.

S. Vaudenay (Ed.): Fast Software Encryption – FSE'98, LNCS 1372, pp. 284–292, 1998.
© Springer-Verlag Berlin Heidelberg 1998

Technical Details for Checking the Example. According to the padding rule, before processing, a message has to be extended by a bit string

$$P = \texttt{100...0 (bin)} \,||\, \ell,$$

where ℓ is the 64-bit representation of the bit-length of the (unextended) message. In P, between `1` on the left and ℓ on the right side, the minimal number of zeros is placed such that the bit-length of the extended message becomes a multiple of $512 = 16 \times 32$ (to allow an iterative application of the compression function, which takes 16 words as input). The above M has bit-length 29×32. This means that

$$\ell = \texttt{00000000000003A0 (hex)},$$

and P is a 96-bit string with the little-endian representation $P = P_0 \,||\, P_1 \,||\, P_2$:

$$\begin{aligned} P_0 &= \texttt{0x00000080}, \\ P_1 &= \texttt{0x000003A0}, \\ P_2 &= \texttt{0x00000000}. \end{aligned}$$

Denote by $\mathsf{MD4}^{[12]}$-$\mathsf{Compress}$ the compression function of $\mathsf{MD4}^{[12]}$, i.e. the first two rounds of the $\mathsf{MD4}$ compression function. In order to compute the $\mathsf{MD4}^{[12]}$ hash value of M, first $\mathsf{MD4}^{[12]}$-$\mathsf{Compress}$ is applied to $M_0, ..., M_{15}$ with the following fixed initial value, which is a part of the specification of $\mathsf{MD4}$:

$$IV = \texttt{0x67452310 0xEFCDAB89 0x98BADCFE 0x10325476}.$$

This gives the output

$$\begin{aligned} C &= \mathsf{MD4}^{[12]}\text{-}\mathsf{Compress}(IV; M_0, ..., M_{15}) \\ &= \texttt{0xA86FDECC 0x25BF84C9 0xDB95C842 0xD0B260B9}. \end{aligned}$$

C is then the initial value for the second application of $\mathsf{MD4}^{[12]}$-$\mathsf{Compress}$ with $M_{16}, ..., M_{28}, P_0, P_1, P_2$ as input. The output is the hash value of M:

$$\begin{aligned} \mathsf{MD4}^{[12]}(M) &= \mathsf{MD4}^{[12]}\text{-}\mathsf{Compress}(C; M_{16}, ..., M_{28}, P_0, P_1, P_2) \\ &= \texttt{0x00000000 0x00000000 0x00000000 0x00000000}. \end{aligned}$$

How to invert the first two rounds of MD4 compression

Suppose $IV = (IV_0, IV_1, IV_2, IV_3)$ and the compress value $C = (C_0, C_1, C_2, C_3)$ are given. Set $H = (H_0, H_1, H_2, H_3) := C - IV$. Our approach is described in the following tables (the underlined intries are those which are up-dated in the repsective steps):

Round One of MD4 compression

input	register A	register B	register C	register D	step
	IV_0	IV_1	IV_2	IV_3	
X_0	$\underline{*}$	IV_1	IV_2	IV_3	step 0
X_1	$*$	IV_1	IV_2	$\underline{*}$	step 1
X_2	$*$	IV_1	$\underline{*}$	$*$	step 2
X_3	$*$	$\underline{*}$	$*$	$*$	step 3
X_4	$\underline{*}$	$*$	$*$	$*$	step 4
X_5	$*$	$*$	$*$	$\underline{*}$	step 5
X_6	$*$	$*$	$\underline{*}$	$*$	step 6
X_7	$*$	$\underline{*}$	$*$	$*$	step 7
X_8	$\underline{P_0}$	$*$	$*$	$*$	step 8
X_9	P_0	$*$	$*$	$\underline{P_3}$	step 9
X_{10}	P_0	$*$	$\underline{P_2}$	P_3	step 10
X_{11}	P_0	$\underline{P_1}$	P_2	P_3	step 11
X_{12}	$\underline{K}$	P_1	P_2	P_3	step 12
X_{13}	K	P_1	P_2	$\underline{K}$	step 13
X_{14}	K	P_1	$\underline{K}$	K	step 14
X_{15}	K	$\underline{B_3}$	K	K	step 15

Round Two of MD4 compression

input	register A	register B	register C	register D	step
X_0	$\underline{K}$	B_3	K	K	step 16
X_4	K	B_3	K	$\underline{K}$	step 17
X_8	K	B_3	$\underline{K}$	K	step 18
X_{12}	K	$\underline{B_2}$	K	K	step 19
X_1	$\underline{K}$	B_2	K	K	step 20
X_5	K	B_2	K	$\underline{K}$	step 21
X_9	K	B_2	$\underline{K}$	K	step 22
X_{13}	K	$\underline{B_1}$	K	K	step 23
X_2	$\underline{K}$	B_1	K	K	step 24
X_6	K	B_1	K	$\underline{K}$	step 25
X_{10}	K	B_1	$\underline{K}$	K	step 26
X_{14}	K	$\underline{B_0}$	K	K	step 27
X_3	$\underline{H_0}$	B_0	K	K	step 28
X_7	H_0	B_0	K	$\underline{H_3}$	step 29
X_{11}	H_0	B_0	$\underline{H_2}$	H_3	step 30
X_{15}	H_0	$\underline{H_1}$	H_2	H_3	step 31

That is, we assume that the contents of the A-, D-, and C-registers equals a constant K in steps 12,16,20,24, steps 13,17,21,25, and steps 14,18,22,26, respectively.

Idea. In round two the majority function g is applied, and by the described approach we can separate the contents in the B-registers from those in the other registers in round two.

Algorithm. Choose K and B_0 randomly. Now $X_0, ..., X_{11}$, and X_{15} are fixed by the steps 16,20, 24,28,17,21,25,29,18,22,26,30, and 31, respectively: for instance in step 16 we have the equation

$$K = (K + g(B3, K, K) + X_0 + K_1) << 3 = (K + K + X_0 + K_1) << 3$$

with $K_1 = \texttt{0x5A827999}$. Thus $X_0 = (K << 29) - 2K - K_1$, and so on.

Now compute P_0, P_1, P_2, P_3 by applying $X_0, ..., X_{11}$ in step 0,...,11. The values P_0, P_1, P_2, P_3 allow next to determine X_{12}, X_{13}, X_{14} from the steps 12,13,14.

Use X_{15} in step 15 to compute B_3, X_{12} in step 19 to compute B_2, X_{13} in step 23 to compute B_1, and finally X_{14} in step 27 to compute another value for B_0 which is "derived from above". If the latter B_0-value matches the chosen one, then we have found a preimage, otherwise try again.

Thus we need about 2^{32} trials to be successful. However, the algorithm can be speeded up by a factor of 100 or more if we use "continuous approximation" for the computation of B_0 (where K is fixed respectively, in order to have sufficient continuity; see second part of the below C-progam).

References

1. H. Dobbertin, *Cryptanalysis of MD4*, Fast Software Encryption (Third Workshop on Cryptographic Algorithms, Cambridge 1996), Lecture Notes in Computer Science, Springer-Verlag 1996, pp. 55-72.
2. H. Dobbertin, *The status of MD5 after a recent attack*, CryptoBytes, The technical newsletter of RSA Laboratories, vol. 2/2, Sommer 1996, pp. 1-6.
3. R. Rivest, *The MD4 message-digest algorithm*, Request for Comments (RFC) 1320, Internet Activities Board, Internet Privacy Task Force, April 1992.
4. M.J.B. Robshaw, *On recent results for MD2, MD4 and MD5*, Bulletin 4, RSA Laboratories, November 1996 (see http://www.rsa.com/PUBS/).

Appendix

C-program inverting the reduced MD4 with cancelled third compression round

The first part of the program below is a modifiction of the above algorithm, which allows to match the redundancy in the input required by the padding rule. On a Pentium PC the program finds a hash-preimage in about 15-20 minutes on the average.

```
#define UL unsigned long
#define shift(x,i) (UL)(((x)<<(i))^((x)>>(32-(i))))
#define f(x,y,z) ((x)&(y) & (~(x))&(z))
#define g(x,y,z) (UL)((x)&(y) | (x)&(z) | (y)&(z))
#include <stdio.h>

main(int ac,char *av[]){
  int i,k,sh,trials,Zeros,Ones,record,weight;
  UL KK,KKK,KKKK,K1,diff,test;
  UL AX,BX,CX,DX,P0,P1,P2,P3,AY,BY,CY,DY;
```

```
  UL X0,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15;
  UL Delta_A,Delta_B0,B0_basic,P2_basic;
  UL AA,B0,B1,B2,B3,C0,C1,C2,C3;
  UL H0,H1,H2,H3,HH0,HH1,HH2,HH3,HHH0,HHH1,HHH2,HHH3;
  UL Q0,Q1,Q2,Q3,IV0,IV1,IV2,IV3;
  UL M0,M1,M2,M3,M4,M5,M6,M7,M8,M9,M10;
  UL M11,M12,M13,M14,M15,M16,M17,M18,M19;
  UL M20,M21,M22,M23,M24,M25,M26,M27,M28;
  UL PP0,PP1,PP2;

  if(ac!=2){
    fprintf(stdout,"Usage: %s seed\n",av[0]);
    exit(1);
  }

  srand(atoi(av[1]));

  K1 = 0x5A827999;
  KK = 0x57902134;
  KKK= 0x57902134;

  /* We have here a special case of a more general algorithm. In
  general KK and KKK are different, but have only a small Hamming
  difference. How to choose these constants will be explained in
  the complete paper about this attack.*/

  Zeros=0;
  Ones=0;
  trials=0;

  /* Here you can specify the hash value  (HH0,HH1,HH2,HH3) */
  HH0=0x0;
  HH1=0x0;
  HH2=0x0;
  HH3=0x0;
  /*********************************************************/

  HHH0=HH0;
  HHH1=HH1;
  HHH2=HH2;
  HHH3=HH3;

  /* Here starts the first part: searching M16,...,M28 */

START_I:

  record = 33;

  AA = KK;
  H2 = rand();
  H3 = rand()&0xffffff7f;
  H0 = KK;
  X15 = 0x0;
  P1 = shift(KK,13)-KK-X15;
  X14= 0x3A0;
  KKKK = shift(KKK+KK+X14+K1,13);
  H1 = shift(KKKK+g(H2,H3,KK)+X15+K1,13);

  IV0=HH0-H0;
  IV1=HH1-H1;
  IV2=HH2-H2;
  IV3=HH3-H3;

  X0 = shift(KK,29)-AA-KK-K1;
  X1 = shift(KK,29)-KK-KK-K1;
  X2 = X1;
  X3 = X1;
  X4 = shift(KK,27)-KK-KK-K1;
```

```
  X5 = X4;
  X6 = X4;
  X7 = shift(H3,27)-KK-KK-K1;
  X12= shift(KK,19)-KK-KK-K1;
  X13= shift(KKK,19)-KK-KK-K1;
  X14= shift(KKKK,19)-KKK-KK-K1;

  P2_basic = rand();

  AX=IV0;BX=IV1;CX=IV2;DX=IV3;
  AX = shift(AX+f(BX,CX,DX)+ X0, 3);
  DX = shift(DX+f(AX,BX,CX)+ X1, 7);
  CX = shift(CX+f(DX,AX,BX)+ X2,11);
  BX = shift(BX+f(CX,DX,AX)+ X3,19);
  AX = shift(AX+f(BX,CX,DX)+ X4, 3);
  DX = shift(DX+f(AX,BX,CX)+ X5, 7);
  CX = shift(CX+f(DX,AX,BX)+ X6,11);
  BX = shift(BX+f(CX,DX,AX)+ X7,19);
  Q0=AX;Q1=BX;Q2=CX;Q3=DX;

  for(i=0; i<250; i++){

    trials=trials+1;

    sh=i&0x1f;
    diff=shift(1,sh);
    P2 = P2_basic^diff;
    C3 = shift(P2+f(KK,AA,P1)+X14,11);

    P3  = shift(KK,25)-f(AA,P1,P2)-X13;
    P0  = shift(AA,29)-f(P1,P2,P3)-X12;

    X8  = shift(P0,29)-f(Q1,Q2,Q3)-Q0;
    X9  = shift(P3,25)-f(P0,Q1,Q2)-Q3;
    X10 = shift(P2,21)-f(P3,P0,Q1)-Q2;
    X11 = shift(P1,13)-f(P2,P3,P0)-Q1;

    C2  = shift(C3+KK+X8+K1,9);
    C1  = shift(C2+KK+X9+K1,9);
    C0  = shift(C1+KK+X10+K1,9);

    Delta_A = shift(H2,23)-g(H3,KK,KKKK)-K1-C0;
    Delta_A = Delta_A^X11;

    weight=0;
    for(k=0; k<32; k++){Delta_A=shift(Delta_A,1);weight=weight+(Delta_A&1);}

    if(weight<record+2){
      P2_basic = P2;
    }
    if(weight<record){
      record=weight;
      if(record<2){
if(record==1){Ones=Ones+1;}
        fprintf(stdout,"Part I:   Hamming dist. %i  ",record);
        fprintf(stdout,"Trials %i  ",trials);
        fprintf(stdout,"Ones %i  Zeros %i\n",Ones,Zeros);
        fprintf(stdout,"%8.8X %i\n\n",Delta_A,i);
      }
    }

    if(weight==0){

      Zeros = Zeros+1;
      test = g(KKKK,KK,C0)^KK;
      if(test!=0){goto START_I;}
      test = g(KKK,KK,C1)^KK;
      if(test!=0){goto START_I;}
```

```
      M16=X0;
      M17=X1;
      M18=X2;
      M19=X3;
      M20=X4;
      M21=X5;
      M22=X6;
      M23=X7;
      M24=X8;
      M25=X9;
      M26=X10;
      M27=X11;
      M28=X12;
      PP0=X13;
      PP1=X14;
      PP2=X15;
      trials=0;

      H0=IV0;
      H1=IV1;
      H2=IV2;
      H3=IV3;
      IV0=0x67452301;
      IV1=0xefcdab89;
      IV2=0x98badcfe;
      IV3=0x10325476;
      H0=-IV0+H0;
      H1=-IV1+H1;
      H2=-IV2+H2;
      H3=-IV3+H3;
      goto START_II;
    }
   }
  goto START_I;



  /* Here starts the second part: searching M0,...,M15 */

START_II:
  record=33;
  KK = rand();
  B0_basic=rand();

  X0 = shift(KK,29)-KK-KK-K1;
  X1 = shift(KK,29)-KK-KK-K1;
  X2 = shift(KK,29)-KK-KK-K1;
  X3 = shift(H0,29)-KK-KK-K1;
  X4 = shift(KK,27)-KK-KK-K1;
  X5 = shift(KK,27)-KK-KK-K1;
  X6 = shift(KK,27)-KK-KK-K1;
  X8 = shift(KK,23)-KK-KK-K1;
  X9 = shift(KK,23)-KK-KK-K1;
  X10= shift(KK,23)-KK-KK-K1;

  for(i=0; i<250; i++){

    trials=trials+1;

    sh=i&0x1f;
    diff=shift(1,sh);
    B0=B0_basic^diff;

    X7 = shift(H3,27)-g(H0,B0,KK)-KK-K1;
    X11= shift(H2,23)-g(H3,H0,B0)-KK-K1;
    X15= shift(H1,19)-g(H2,H3,H0)-K1-B0;
```

```
AX=IV0;BX=IV1;CX=IV2;DX=IV3;
AX = shift(AX+f(BX,CX,DX)+ X0, 3);
DX = shift(DX+f(AX,BX,CX)+ X1, 7);
CX = shift(CX+f(DX,AX,BX)+ X2,11);
BX = shift(BX+f(CX,DX,AX)+ X3,19);
AX = shift(AX+f(BX,CX,DX)+ X4, 3);
DX = shift(DX+f(AX,BX,CX)+ X5, 7);
CX = shift(CX+f(DX,AX,BX)+ X6,11);
BX = shift(BX+f(CX,DX,AX)+ X7,19);
AX = shift(AX+f(BX,CX,DX)+ X8, 3);
DX = shift(DX+f(AX,BX,CX)+ X9, 7);
CX = shift(CX+f(DX,AX,BX)+X10,11);
BX = shift(BX+f(CX,DX,AX)+X11,19);
P0=AX;P1=BX;P2=CX;P3=DX;

X12 = shift(KK,29)-P0-f(P1,P2,P3);
X13 = shift(KK,25)-P3-f(KK,P1,P2);
X14 = shift(KK,21)-P2-f(KK,KK,P1);

B3 = shift(P1+KK+X15,19);
B2 = shift(B3+X12+KK+K1,13);
B1 = shift(B2+X13+KK+K1,13);
Delta_B0 = shift(B1+X14+KK+K1,13)-B0;

weight=0;
for(k=0; k<32; k++){Delta_B0=shift(Delta_B0,1);weight=weight+(Delta_B0&1);}

if(weight<record+2){
  B0_basic = B0;
}
if(weight<record){
  record=weight;
  if(record<2){
   if(record==1){Ones=Ones+1;}
    fprintf(stdout,"Part II:   Hamming dist. 1  ");
    fprintf(stdout,"Trials %i  ",trials);
     fprintf(stdout,"Ones %i\n",Ones);
    fprintf(stdout,"%8.8X %i\n\n",Delta_B0,i);
   }
}

if(weight==0){
  fprintf(stdout,"Cancel the third round ");
  fprintf(stdout,"of the MD4 compression function,\n");
  fprintf(stdout,"then the following message M=M0,...,M28 has ");
  fprintf(stdout,"the hash value\n");
  fprintf(stdout,"H  = 0x%8.8X ",HHH0);
  fprintf(stdout,"0x%8.8X ",HHH1);
  fprintf(stdout,"0x%8.8X ",HHH2);
  fprintf(stdout,"0x%8.8X:\n\n",HHH3);
  fprintf(stdout,"M0 = 0x%8.8X;  ",X0);
  fprintf(stdout,"M1 = 0x%8.8X;\n",X1);
  fprintf(stdout,"M2 = 0x%8.8X;  ",X2);
  fprintf(stdout,"M3 = 0x%8.8X;\n",X3);
  fprintf(stdout,"M4 = 0x%8.8X;  ",X4);
  fprintf(stdout,"M5 = 0x%8.8X;\n",X5);
  fprintf(stdout,"M6 = 0x%8.8X;  ",X6);
  fprintf(stdout,"M7 = 0x%8.8X;\n",X7);
  fprintf(stdout,"M8 = 0x%8.8X;  ",X8);
  fprintf(stdout,"M9 = 0x%8.8X;\n",X9);
  fprintf(stdout,"M10= 0x%8.8X;  ",X10);
  fprintf(stdout,"M11= 0x%8.8X;\n",X11);
  fprintf(stdout,"M12= 0x%8.8X;  ",X12);
  fprintf(stdout,"M13= 0x%8.8X;\n",X13);
  fprintf(stdout,"M14= 0x%8.8X;  ",X14);
  fprintf(stdout,"M15= 0x%8.8X;\n",X15);
  fprintf(stdout,"M16= 0x%8.8X;  ",M16);
  fprintf(stdout,"M17= 0x%8.8X;\n",M17);
```

```
      fprintf(stdout,"M18= 0x%8.8X;  ",M18);
      fprintf(stdout,"M19= 0x%8.8X;\n",M19);
      fprintf(stdout,"M20= 0x%8.8X;  ",M20);
      fprintf(stdout,"M21= 0x%8.8X;\n",M21);
      fprintf(stdout,"M22= 0x%8.8X;  ",M22);
      fprintf(stdout,"M23= 0x%8.8X;\n",M23);
      fprintf(stdout,"M24= 0x%8.8X;  ",M24);
      fprintf(stdout,"M25= 0x%8.8X;\n",M25);
      fprintf(stdout,"M26= 0x%8.8X;  ",M26);
      fprintf(stdout,"M27= 0x%8.8X;\n",M27);
      fprintf(stdout,"M28= 0x%8.8X;\n\n",M28);
      fprintf(stdout,"The corresponding padding string is P0,P1,P2:\n");
      fprintf(stdout,"P0 = 0x%8.8X;  ",PP0);
      fprintf(stdout,"P1 = 0x%8.8X;  ",PP1);
      fprintf(stdout,"P2 = 0x%8.8X;\n\n",PP2);
      exit(1);
    }
  }
  goto START_II;
}
```