APPENDIX E: SOURCE CODE: ANALYSIS OF MD5

E.1 SOURCE CODE: FIRST PHASE OF THE ATTACK ON MD5

This section of the appendix contains the C source code used in the first phase of the attack on MD5.

```
/* This program is a reconstruction of Dobbertin's attack on MD5.
 *
 * Author: P.R. Kasselman
 * Date: 16/06/1997
 * Filename: md5an04.c */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

unsigned int Rotate(unsigned int X, unsigned int s);
unsigned int RotateRight(unsigned int X, unsigned int s);
unsigned int F(unsigned int X, unsigned int Y, unsigned int Z);
unsigned int G(unsigned int X, unsigned int Y, unsigned int Z);
unsigned int GG(unsigned int x, unsigned int c1, unsigned int d1,
unsigned int c2, unsigned int d2, unsigned int a);
unsigned int GGG(unsigned int x, unsigned int a1, unsigned int c1,
unsigned int a2, unsigned int c2, unsigned int b);
unsigned int NG(unsigned int x, unsigned int d1, unsigned int a1,
unsigned int d2, unsigned int a2, unsigned int c);
unsigned int search1(unsigned int x, unsigned int c1, unsigned int d1,
  unsigned int c2, unsigned int d2, unsigned int a,
  unsigned int Target, unsigned int *data);
unsigned int search2(unsigned int x, unsigned int a1, unsigned int c1,
  unsigned int a2, unsigned int c2, unsigned int b,
  unsigned int Target, unsigned int *data);
unsigned int search3(unsigned int x, unsigned int d1, unsigned int a1,
  unsigned int d2, unsigned int a2, unsigned int c,
  unsigned int Target, unsigned int *data);

#define FS1 7
#define FS2 12
#define FS3 17
#define FS4 22

#define GS1 5
#define GS2 9
#define GS3 14
#define GS4 20

#define EPS1 0x08000000
#define EPS2 0xfffffe00

#define MAX 0x00100000

int main()
{
 unsigned int i,j,k,l;
 unsigned int x,konst;
```

```
unsigned int data1[MAX], data2[MAX], data3[MAX];
unsigned int b2,c2,a,b1,c1;
unsigned int A15,B15,C15,D15,A19,B19,C19,D19,A23,B23,C23,D23;
unsigned int Bt15,Ct15,At19,Ct19,Dt19,Dt23;
unsigned int DeltaX14;
unsigned int DeltaA19, DeltaC19, DeltaD19, DeltaC15, DeltaB15;
unsigned int BasicA15, BasicB15, BasicC19;
unsigned int Stop, Phase, Test, Funny, MaxPhase, Y, alt;
unsigned int SearchCount1, SearchCount2, SearchCount3, SearchCount4;
unsigned int Diff1, Diff2, Diff3;
unsigned int Count1, Count2, Count3;
unsigned int Target1, Target2, Target3;
unsigned int M[8];
time_t TheTime;

TheTime = time(NULL);
srandom(TheTime);

/* Specify the difference for X14 */
DeltaX14 = Rotate(1,9);

M[0] = 0x0000000f;
M[1] = 0x000000ff;
M[2] = 0x00000fff;
M[3] = 0x0000ffff;
M[4] = 0x000fffff;
M[5] = 0x00ffffff;
M[6] = 0x0fffffff;
M[7] = 0xffffffff;

SearchCount1 = 0;
SearchCount2 = 0;
SearchCount3 = 0;
SearchCount4 = 0;

/* Make initial choices for D19, Dt19, D15, C15 and Ct15 that is guaranteed
   to satisfy the conditions imposed. */

D19 = 0x00000000;
Dt19 = -1-D19;
DeltaD19 = Dt19-D19;

printf("Delta D19: %8.8X\n", DeltaD19);

C19 =  random() ^ Rotate(random(),1);
Ct19 = C19+EPS1;
DeltaC19 = Ct19-C19;

printf("Delta C19: %8.8X\n", DeltaC19);

D15 = Rotate(1,16)-Rotate(1,25);
C15 = D15 - 1;
Ct15 = Rotate(Rotate(C15-D15,15)+DeltaX14,17)+D15;
DeltaC15 = Ct15-C15;
```

```
 Phase = 0;

 printf("Delta C15: %8.8X\n", DeltaC15);

 Stop = 0;
 Count3 = 0;
 while(Stop == 0)
  {
if(Phase == 0)
 {
   MaxPhase = 0;
SearchCount1 = 0;
SearchCount2 = 0;
  SearchCount3 = 0;
  SearchCount4 = 0;

  /* Choose A15 randomly */
  A15 = random() ^ Rotate(random(),1);

  /* Determine Delta B15 based from (2) */


    DeltaB15 = Rotate(F(Ct15,D15,A15)-F(C15,D15,A15),22)
  + Rotate(Ct15-C15,0);

  /*printf("Delta B15: %8.8X\n", DeltaB15);*/

  /* Choose B15 randomly */
  B15 = random() ^ Rotate(random(),1);

  /* Compute Bt15 based on the random choice of B15 */
  Bt15 = DeltaB15 + B15;

  /* Determine Delta A19 from (3) */
  DeltaA19 = Rotate(G(Bt15,Ct15,D15)-G(B15,C15,D15),5)
  + Rotate(Bt15-B15,0);

  /*printf("Delta A19: %8.8X\n", DeltaA19);*/

  /* Choose C19 randomly */
  C19 = random() ^ Rotate(random(),1);;
Ct19 = C19+EPS1;
 }

if(Phase > 0)
 {
  /* Choose A15 to be close to the previous "good" value */
  Diff1 = random()&random()&random()&random();
  A15 = BasicA15 ^ Diff1;

  /* Determine Delta B15 based from (2) */
  DeltaB15 = Rotate(F(Ct15,D15,A15)-F(C15,D15,A15),22)
  + Rotate(Ct15-C15,0);

  /*printf("Delta B15: %8.8X\n", DeltaB15);*/
```

```
  /* Choose B15 to be close to the previous "good" value */
  Diff2 = random()&random()&random()&random();
  B15 = BasicB15 ^ Diff2;

  /* Compute Bt15 based on the random choice of B15 */
  Bt15 = DeltaB15 + B15;

  /* Determine Delta A19 from (3) */
  DeltaA19 = Rotate(G(Bt15,Ct15,D15)-G(B15,C15,D15),5)
  + Rotate(Bt15-B15,0);

  /*printf("Delta A19: %8.8X\n", DeltaA19);*/

  /* Choose C19 randomly */
  Diff3 = random()&random()&random()&random();
  C19 = BasicC19 ^ Diff3;
Ct19 = C19+EPS1;
 }

   Funny = 0;

/* Set a target */
Target1 =  Rotate(-DeltaA19,12) - Rotate(-DeltaA19-EPS1,12) + DeltaB15;

/* For the given target determine if any solutions exists, and if so,
 * how many solutions exists. */

x = random();

/* Find valid  values for A19 and implicitly At19 */
Count1 = 0;
Count1 = search1(x,C19,D19,Ct19,Dt19,DeltaA19,Target1,data1);

/* The array data1 now contains all valid values of A19 */
for(i=0; i<Count1; i++)
 {
A19  = data1[i];
At19 = data1[i] + DeltaA19;

Target2 = Rotate(D19-A19, 23) - Rotate(Dt19-At19, 23);

  x = random();

/* Find valid  values for B15 and implicitly Bt15 */
Count2 = 0;
Count2 = search2(x,A19,C15,At19,Ct15,DeltaB15,Target2,data2);
for(j=0; j<Count2; j++)
 {
B15  = data2[j];
Bt15 = data2[j] + DeltaB15;

/* Confirm if equation (3) holds */
Test = G(B15,C15,D15)-G(Bt15,Ct15,D15)
- Rotate(A19 - B15, 27)
```

```
+ Rotate(At19 - Bt15, 27);
if(Test == 0)
 {
/* Confirm if equation (2) holds */
Test = F(C15,D15,A15) - F(Ct15,D15,A15)
- Rotate(B15-C15,10)
+ Rotate(Bt15-Ct15,10);

if(Test == 0)
   {
    /* Now determine if equation (5) holds */
  Test = G(D19,A19,B15)-G(Dt19,At19,Bt15)
- Rotate(C19 - D19, 18)
+ Rotate(Ct19 - Dt19, 18)
+ C15 - Ct15;

if(((Test&M[0])==0)&&(Phase<2))
 {
   Phase = 1;
   Funny = 1;
   /*printf("Hit %d\n", Phase);*/
 }

if(((Test&M[1])==0)&&(Phase<3))
 {
   Phase = 2;
   Funny = 2;
   /* printf("Hit %d\n", Phase);*/
 }
if(((Test&M[2])==0)&&(Phase<4))
 {
   Phase = 3;
   Funny = 3;
   /*printf("Hit %d\n", Phase);*/
 }
if(((Test&M[3])==0)&&(Phase<5))
 {
   Phase = 4;
   Funny = 4;
   /*printf("Hit %d\n", Phase);*/
   SearchCount1++;

   if(SearchCount1 > 10000)
    {
     SearchCount1 = 0;
Phase = 0;
printf("Resetting Process\n");
    }

 }

if(((Test&M[4])==0)&&(Phase<6))
 {
   Phase = 5;
   Funny = 5;
```

```
    SearchCount2++;
    /*printf("Hit %d\n", Phase);*/

    if(SearchCount2 > 10000)
     {
      SearchCount2 = 0;
Phase = 0;
printf("Resetting Process\n");
     }

 }

if(((Test&M[5])==0)&&(Phase<7))
 {
    Phase = 6;
    Funny = 6;
    SearchCount3++;
        /*printf("Hit %d\n", Phase);*/
    if(SearchCount3 > 10000)
     {
      SearchCount3 = 0;
Phase = 0;
printf("Resetting Process\n");
     }
 }

if(((Test&M[6])==0)&&(Phase<8))
 {
    Phase = 7;
    Funny = 7;
    SearchCount4++;

    /*printf("Hit %d\n", Phase);*/
    if(SearchCount4 > 16000)
     {
      SearchCount4 = 0;
Phase = 0;
printf("Resetting Process\n");
     }
 }

if(Test == 0)
 {
  printf("Checking Results\n");
  Test = F(Ct15,D15,A15)
  - F(C15,D15,A15)
  - Rotate(Bt15-Ct15,10)
  + Rotate(B15-C15,10);

  if(Test!=0)
   {
    printf("Error 1\n");
    exit(1);
   }
```

```
  Test = G(B15,C15,D15)
  - G(Bt15,Ct15,D15)
  - Rotate(A19-B15,27)
  + Rotate(At19-Bt15,27);

  if(Test!=0)
   {
    printf("Error 2\n");
    exit(1);
   }
Test = G(A19,B15,C15)
  - G(At19,Bt15,Ct15)
  - Rotate(D19-A19,23)
  + Rotate(Dt19-At19,23);

  if(Test!=0)
   {
    printf("Error 3\n");
    exit(1);
   }

Test = G(D19,A19,B15)
  - G(Dt19,At19,Bt15)
  - Rotate(C19-D19,18)
  + Rotate(Ct19-Dt19,18)
  + C15-Ct15;

  if(Test!=0)
   {
    printf("Error 4\n");
    exit(1);
   }

  /* Now that equations (1)-(7) are
   * satisfied, determine if (8) is
   * satisfied and if the error
   * propagates as expected */

  Target3 =  Rotate(-DeltaA19,12)
        - Rotate(-DeltaA19-EPS1,12)
        + DeltaB15;

  x = random();

  /* Find valid  values for C19 and
   * implicitly Ct19 */

  Count3 = 0;
  Count3 = search3(x,D19,A19,Dt19,At19,DeltaC19,Target3,data3);

  for(k=0; k<Count3; k++)
   {
    C19 = data3[k];
    Ct19 = data3[k] + EPS1;
    B19 = C19 - DeltaA19;
```

```
/* Confirm if (6) holds */
Test = G(C19,D19,A19)
     - G(Ct19,Dt19,At19)
     + B15 - Bt15
     - Rotate(B19-C19,12)
     + Rotate(B19-Ct19,12);
if(Test != 0)
 {
  printf("Error 4\n");
  exit(1);
 }

/* Confirm if (7) holds */
Test = G(B19,C19,D19)
     - G(B19,Ct19,Dt19)
     + A19 - At19;
if(Test != 0)
 {
  printf("Error 5\n");
  exit(1);
 }

/* Determine if propagation of
 * differences are as expected */

 alt = 0;
 if(alt==0)
  {
   for(l=0; l<10; l++)
    {
     A23 = random()
      ^ Rotate(random(),1);
     D23 = random()
      ^ Rotate(random(),1);
     Dt23 = D23 + EPS2;
     Y = random()
              ^ Rotate(random(),1);

     Test = Rotate(Ct19
      + G(Dt23,A23,B19)
       + Y,14)
      + Dt23;

     Test = Test - (Rotate(C19
      + G(D23,A23,B19)
      + Y,14)
      + D23);

     if(Test != 0)
      {
       Phase = 0;
      }

     if(Test == 0)
```

```
            {
    Test = Rotate(Dt23-A23,23)
       - Rotate(D23-A23,23);
    Test = Test
       - G(A23,B19,Ct19)
       + G(A23,B19,C19);
    Test = Test - (Dt19-D19);

    if(Test == 0)
     {
        Stop++;
        printf("A15  = 0x%8.8X;\n", A15);
        printf("D15  = 0x%8.8X;\n", D15);
        printf("C15  = 0x%8.8X;\n", C15);
        /*printf("Ct15 = 0x%8.8X;\n", Ct15);*/
        printf("B15  = 0x%8.8X;\n", B15);
        /*printf("Bt15 = 0x%8.8X;\n", Bt15);*/
        printf("A19  = 0x%8.8X;\n", A19);
        /*printf("At19 = 0x%8.8X;\n", At19);*/
        printf("D19  = 0x%8.8X;\n", D19);
        /*printf("Dt19 = 0x%8.8X;\n", Dt19);*/
        printf("C19  = 0x%8.8X;\n", C19);
        /*printf("Ct19 = 0x%8.8X;\n", Ct19);*/
        printf("B19  = 0x%8.8X;\n", B19);
        printf("A23  = 0x%8.8X;\n", A23);
        /*printf("B23  = 0x%8.8X;\n", B23);
        printf("C23  = 0x%8.8X;\n", C23);
        printf("D23  = 0x%8.8X;\n", D23);
        printf("Dt23 = 0x%8.8X;\n", Dt23);*/

        exit(0);
             }

           }
         }
         if(Stop == 1)
          {
           printf("Found a Solution\n");
          }
       }
    }

 }

/* Preserve the basic values */
if(Funny == Phase)
 {
  BasicA15 = A15;
     BasicB15 = B15;
     BasicC19 = C19;
 }
 }
 }
 }
 }
```

```
if(Phase > MaxPhase)
 {
  MaxPhase = Phase;
printf("Best Current Phase: %d\n", MaxPhase);
printf("%d %d %d %d\n", SearchCount1, SearchCount2,
SearchCount3,SearchCount4);
 }
  }

 printf("Target: %8.8X\n", Target1);
 printf("Target: %8.8X\n", Target2);
 printf("Number of solutions (7): %d\n", Count1);
 printf("Number of solutions (4a): %d\n", Count2);
 printf("Number of solutions (4b): %d\n", Count3);
 printf("Number of Valid Solutions (3) %d\n", Stop);
 return(0);
}

/* Hierdie deel kan so klein bietjie meer elegant gedoen word,
 * maar a.g.v. tydbeperkings gaan ek nie nou daaraan karring nie */

unsigned int search1(unsigned int x, unsigned int c1, unsigned int d1,
  unsigned int c2, unsigned int d2, unsigned int a,
  unsigned int Target, unsigned int *data)
{
 int i;
 static int index, count;

 i = index;

 if(i==0)
  {
count = 0;
  }


 /* This is where you check the depth of the tree */
 if(i==32)
  {
   data[count] = x;
count++;
return(count);
  }

 index++;

 if(count >= MAX)
  {
   index--;
   return(count);
  }

 if( (((GG(x,c1,d1,c2,d2,a) - Target) >> i) & 0x00000001) == 0)
  {
```

```
search1(x,c1,d1,c2,d2,a,Target,data);
  }

  x = x^(1<<i);

 if( (((GG(x,c1,d1,c2,d2,a)-Target) >> i) & 0x00000001) == 0)
  {
search1(x,c1,d1,c2,d2,a,Target,data);
  }

 x = x^(1<<i);

 index--;
 return(count);
}

/* Soek oplossings vir vergelyking 4 */
unsigned int search2(unsigned int x, unsigned int a1, unsigned int c1,
  unsigned int a2, unsigned int c2, unsigned int b,
  unsigned int Target, unsigned int *data)
{
 int i;
 static int index, count;

 i = index;

 if(i==0)
  {
count = 0;
  }


 /* This is where you check the depth of the tree */
 if(i==32)
  {
   data[count] = x;
count++;
return(count);
  }

 index++;

 if(count >= MAX)
  {
   index--;
   return(count);
  }

 if( (((GGG(x,a1,c1,a2,c2,b)-Target) >> i) & 0x00000001) == 0)
  {
search2(x,a1,c1,a2,c2,b,Target,data);
  }

  x = x^(1<<i);
```

```
 if( (((GGG(x,a1,c1,a2,c2,b)-Target) >> i) & 0x00000001) == 0)
  {
search2(x,a1,c1,a2,c2,b,Target,data);
  }

 x = x^(1<<i);

 index--;
 return(count);
}

/* Soek oplossings vir vergelyking (6) en (7) gekombineer */
unsigned int search3(unsigned int x, unsigned int d1, unsigned int a1,
  unsigned int d2, unsigned int a2, unsigned int c,
  unsigned int Target, unsigned int *data)
{
 int i;
 static int index, count;

 i = index;

 if(i==0)
  {
count = 0;
  }


 /* This is where you check the depth of the tree */
 if(i==32)
  {
   data[count] = x;
count++;
return(count);
  }

 index++;

 if(count >= MAX)
  {
   index--;
   return(count);
  }

 if( (((NG(x,d1,a1,d2,a2,c) - Target) >> i) & 0x00000001) == 0)
  {
search3(x,d1,a1,d2,a2,c,Target,data);
  }

  x = x^(1<<i);

 if( (((NG(x,d1,a1,d2,a2,c)-Target) >> i) & 0x00000001) == 0)
  {
search3(x,d1,a1,d2,a2,c,Target,data);
  }
```

```
 x = x^(1<<i);

 index--;
 return(count);
}


unsigned int Rotate(unsigned int X, unsigned int s)
{
   unsigned int temp;

   temp = X;
   X = (X << s) | (temp >> (32-s));
   return(X);
}

unsigned int RotateRight(unsigned int X, unsigned int s)
{
   unsigned int temp;

   temp = X;
   X = (X >> s) | (temp << (32-s));
   return(X);
}

unsigned int F(unsigned int X, unsigned int Y, unsigned int Z)
{
   return((X&Y) | ((~X)&Z));
}

unsigned int G(unsigned int X, unsigned int Y, unsigned int Z)
{
   return((X&Z) | (Y&(~Z)));
}

unsigned int GG(unsigned int x, unsigned int c1, unsigned int d1,
unsigned int c2, unsigned int d2, unsigned int a)
{
 /* a is the difference DeltaA19. x is the basic value */
 return(G(c1,d1,x)-G(c2,d2,x+a));
}

unsigned int GGG(unsigned int x, unsigned int a1, unsigned int c1,
unsigned int a2, unsigned int c2, unsigned int b)
{
 /* b is the difference DeltaB15. x is the basic value */
 return(G(a1,x,c1)-G(a2,x+b,c2));
}

unsigned int NG(unsigned int x, unsigned int d1, unsigned int a1,
unsigned int d2, unsigned int a2, unsigned int c)
{
 /* a is the difference DeltaA19. x is the basic value */
 return(G(x,d1,a1)-G(x+c,d2,a2));
}
```

E.2 SOURCE CODE: SECOND PHASE OF THE ATTACK ON MD5

This section of the appendix contains the C source code used in the second phase of the attack on MD5.

```
/* This is the program written by Hans Dobbertin to implement the attack on
 * the last two rounds of MD5. */

#include <stdio.h>
#include <stdlib.h>

#define UL unsigned long
#define MAX 10000
#define shift(x,i) (UL)(((x)<<(i))^((x)>>(32-(i))))
#define f(x,y,z) ((x)^(y)^(z))
#define F(x) (UL)(f(x+a,b2,c2)-f(x,b1,c1))
#define H(x) (UL)(((b1)^(x+a))+((b2)^(x+a))+((c1)^(x))+((c2)^(x)))

UL a,b1,c1,b2,c2,konst;
int *max,max_test;
UL **feld;
int suche(),suchh(),sucheb();

main(int ac,char *av[])
{
   int rr,kk,nn,jj,ii,inn,out,maxi2;
   int count1,count2,count3,count4,count5,count6;
   UL A0,A1,A2,B0,B1,B2,C0,C1,C2,D0,D1,D2,K,K0,U1,U2,V1,V2,W1,W2,Z1,Z2,anfang;
   UL A2_basic,B2_basic,C2_basic,D2_basic,C,D;
   UL X0,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,XX,dX;
   UL Y7,Y8,Y9,Y10,Y11,Y12,X,dC10,Y,Z,U,UU,delta;
   UL test0,test1,test2,test3,test4,test5,test6,test7,test8,test9;
   UL eps1,eps2,eps3,eps4,tem,test,test10,test11;
   int phase,erfolg,sh1,sh2,sh3,diff1,diff2,diff3,tem1,tem2;
   UL dA1,dB1,dD1,dC1,AA1,BB1,CC1,DD1,AA2,BB2,CC2,DD2,dXnull;
   UL *feld0,*feld1,*feld2,*feld3,*feld4,*feld5,*feld6;
   int Count1,Count2,Count3,versuch,flag,ind,maxi,ERFOLG;
   int max0=MAX,max1=MAX,max2=MAX,max3=MAX,max4=MAX,max5=MAX,max6=MAX;
   float quotient;

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


   srand(atoi(av[1]));

   if((feld0=(UL*)malloc(MAX*sizeof(UL)))==NULL ||
      (feld1=(UL*)malloc(MAX*sizeof(UL)))==NULL ||
      (feld2=(UL*)malloc(MAX*sizeof(UL)))==NULL ||
```

```
      (feld3=(UL*)malloc(MAX*sizeof(UL)))==NULL ||
      (feld4=(UL*)malloc(MAX*sizeof(UL)))==NULL ||
      (feld5=(UL*)malloc(MAX*sizeof(UL)))==NULL ||
      (feld6=(UL*)malloc(MAX*sizeof(UL)))==NULL)
     {
      fprintf(stderr,"Nicht genuegend speicher\n");
      exit(1);
     }


   out=0x100;
   inn=0x1000;
   maxi=0x10;
   maxi2=1000;
   ERFOLG=0;
   versuch=0;

   delta=shift(1,9);
   BB1=0;
   B1=-1;
   dB1=BB1-B1;

   test10 =  0x80044000;
   test11 =  0x40040000;
   eps1   = -0x40004000;
   eps2   = -0x80084000;
   eps3   = -0xFFFBFE00;
   eps4   = -0x40000200;

LABEL_A:

   max_test=out;
   erfolg=0;

LABEL_NEU:
   A2=(rand()^shift(rand(),1));
   C2=(rand()^shift(rand(),1));
   C2=C2&0xbffbbdff;
   A2=A2&0xbffbbdff;
   D2=(A2+shift(eps1-eps4,31))^(rand()&rand()&rand()&rand());
   C2=C2^0x40004000;
   D2=D2^0x00000200;
   A2=A2^0x40004200;


   test=f(C2+eps3,D2+eps4,0)-f(C2,D2,0)-test11;
   if(test!=0)
     {
goto LABEL_NEU;
     }

   test=f(C2+eps3,D2+eps4,A2+eps1)-f(C2,D2,A2)-test10;
   if(test!=0)
     {
goto LABEL_NEU;
```

```
      }

    B2=shift(-eps2-1,31)^(rand()&rand()&rand());

    UU=(A2+eps1)^(B2+eps2);
    U =A2^B2;
    dC1=shift(C2-D2+eps3-eps4,16)-shift(C2-D2,16);
    dC1=dC1-((D2+eps4)^UU)+(D2^U);
    C1=(rand()^shift(rand(),1));
    CC1 = C1+dC1;
    dD1 = -(CC1^UU)+(C1^U)+shift(D2-A2+eps4-eps1,21)-shift(D2-A2,21);

    dXnull=(shift(A2-B2+eps1-eps2,28)-shift(A2-B2,28)+1+eps2)&1;
    test=dXnull^((shift(B2+eps2-CC1,9)-shift(B2-C1,9))&1)^(dC1&1);

    if(test==0)
      {
konst=shift(D2-A2+eps4-eps1,21)-shift(D2-A2,21)-dD1;
b2=UU;
c2=0;
b1=U;
c1=0;
a=dC1;
feld=&feld1;
max=&max1;

if(count1=suche())
  {
     for(nn=0; nn<maxi; nn++)
       {
  ind=(unsigned)rand()%count1;
  C1=feld1[ind];
  CC1=C1+dC1;
  if(dXnull==((shift(dC1-dD1,16))&1))
    {
       dX = shift(dC1-dD1,16);
       /*dX = shift(dC1-dD1,16)-shift(1,16);*/
    }
  if(dXnull!=((shift(dC1-dD1,16))&1))
    {
       dX = shift(dC1-dD1,16)+1;
    }

  konst=shift(B2+eps2-CC1,9)-shift(B2-C1,9)-dB1;
  b2=CC1^BB1;
  c2=0;
  b1=C1^B1;
  c1=0;
  a = dX;
  feld=&feld2;
  max=&max2;
  max_test=inn;
  if(count2=suche())
    {
       if(erfolg==0)
```

```
 {
    fprintf(stderr,"** %8d  %8d  %8d",count1,nn,count2);
    fprintf(stderr,"** %8.8X  %8.8X  %8d\n",dC1,dD1,ERFOLG);
    erfolg=1;
 }
       for(kk=0; kk<count2; kk++)
 {
    ind=(unsigned)rand()%count2;
    X=feld2[ind];
    XX=X+dX;
    konst=shift(A2-B2+eps1-eps2,28)-shift(A2-B2,28)-2;
    b1=(B2+eps2)^CC1;
    b2=XX^BB1;
    c1=(-B2-1)^C1;
    c2=(-X-1)^B1;
    a=dD1;
    feld=&feld3;
    max=&max3;
    max_test=inn;
    if(count3=sucheh())
      {
 ERFOLG=ERFOLG+1;
 fprintf(stderr,"LABEL_B %d\n",ERFOLG);
 goto LABEL_B;
      }
 }
       max_test=out;
    }
       }
  }
     }

   goto LABEL_A;

LABEL_B:

   fprintf(stdout,"%d\n",ERFOLG);
   fprintf(stdout,"A2 = 0x%8.8X;\n",A2);
   fprintf(stdout,"B2 = 0x%8.8X;\n",B2);
   fprintf(stdout,"C2 = 0x%8.8X;\n",C2);
   fprintf(stdout,"D2 = 0x%8.8X;\n",D2);
   fprintf(stdout,"C1 = 0x%8.8X;\n",C1);
   fprintf(stdout,"dC1= 0x%8.8X;\n",dC1);
   fprintf(stdout,"dD1= 0x%8.8X;\n\n",dD1);

   erfolg=0;
   flag=0;
   max_test=0x200000;
   konst=shift(D2-A2+eps4-eps1,21)-shift(D2-A2,21)-dD1;
   b2=UU;
   c2=0;
   b1=U;
   c1=0;
   a =dC1;
   feld=&feld1;
```

```
   max=&max1;
   if(count1=suche())
     {
for(nn=0; nn<maxi2; nn++)
  {
     ind=(unsigned)rand()%count1;
     C1=feld1[ind];
     CC1=C1+dC1;
     if(dXnull==((shift(dC1-dD1,16))&1))
       {
  dX = shift(dC1-dD1,16);
  /*dX = shift(dC1-dD1,16)-shift(1,16);*/
       }

     if(dXnull!=((shift(dC1-dD1,16))&1))
       {
  dX = shift(dC1-dD1,16)+1;
       }

     konst=shift(B2+eps2-CC1,9)-shift(B2-C1,9)-dB1;
     b2=CC1^BB1;
     c2=0;
     b1=C1^B1;
     c1=0;
     a=dX;
     feld=&feld2;
     max=&max2;
     if(count2=suche())
       {
  /*fprintf(stderr,"II. ** %8d %8d %8d %8.8X\n",count1,nn,count2,dD1);*/
  for(kk=0; kk<count2; kk++)
    {
       ind=(unsigned)rand()%count2;
       X=feld2[ind];
       XX=X+dX;
       konst=shift(A2-B2+eps1-eps2,28)-shift(A2-B2,28)-2;
       b1=(B2+eps2)^CC1;
       b2=XX^BB1;
       c1=(-B2-1)^C1;
       c2=(-X-1)^B1;
       a=dD1;
       feld=&feld3;
       max=&max3;
       if(count3=sucheh())
 {
    for(rr=0; rr<count3; rr++)
      {
 ind=(unsigned)rand()%count3;
 D1=feld3[ind];
 DD1=D1+dD1;
 test=dX-(shift(CC1-DD1,16)-shift(C1-D1,16));
 if(test==0)
   {
      A1=D1^X^B1;
      AA1=DD1^XX^BB1;
```

```
      dA1=AA1-A1;
      test=(shift(AA1-BB1,28)-shift(A1-B1,28)-1)&1;
      if(test==0)
{
   flag=1;
   konst=shift(DD1-AA1,21)-shift(D1-A1,21);
   b2=AA1;
   c2=BB1;
   b1=A1;
   c1=B1;
   a=0;
   feld=&feld4;
   max=&max4;
   C=rand();
   test = f(AA1,BB1,C)-f(A1,B1,C);
   test = test-shift(DD1-AA1,21)+shift(D1-A1,21);
   if((test&0xffff)==0)
     {
fprintf(stderr,"def = %8.8x %d\n",test,ERFOLG);
     }

   if(count4=sucheb())
     {
test = f(D2+eps4,A2+eps1,B2+eps2)-f(D2,A2,B2);
test = test-shift(C2-D2+eps3-eps4,16)+shift(C2-D2,16)+dC1;
if(test!=0)
  {
     fprintf(stderr,"ERROR1\n");
     exit(1);
  }

test = f(A2+eps1,B2+eps2,CC1)-f(A2,B2,C1);
test = test-shift(D2-A2+eps4-eps1,21)+shift(D2-A2,21)+dD1;
if(test!=0)
  {
     fprintf(stderr,"ERROR2\n");
     exit(1);
  }

test = f(B2+eps2,CC1,DD1)-f(B2,C1,D1);
test = test-shift(A2-B2+eps1-eps2,28)+shift(A2-B2,28)+dA1;
if(test!=0)
  {
     fprintf(stderr,"ERROR3\n");
     exit(1);
  }

test = f(CC1,DD1,AA1)-f(C1,D1,A1);
test = test-shift(B2-CC1+eps2,9)+shift(B2-C1,9)+dB1;
if(test!=0)
  {
     fprintf(stderr,"ERROR4\n");
     exit(1);
  }
```

```
test = f(DD1,AA1,BB1)-f(D1,A1,B1);
test = test-shift(CC1-DD1,16)+shift(C1-D1,16);
if(test!=0)
  {
     fprintf(stderr,"ERROR5\n");
     exit(1);
  }


C=feld4[0];
test = f(AA1,BB1,C)-f(A1,B1,C);
test = test-shift(DD1-AA1,21)+shift(D1-A1,21);
if(test!=0)
  {
     fprintf(stderr,"ERROR6\n");
     exit(1);
  }

D=shift(shift(AA1-BB1,28)-shift(A1-B1,28)-1,31)^C;
test = f(BB1,C,D)-f(B1,C,D);
test = test-shift(AA1-BB1,28)+shift(A1-B1,28);
if(test!=0)
  {
     fprintf(stderr,"ERROR7\n");
  }

fprintf(stderr,"A2 = Ox%8.8X;\n",A2);
fprintf(stderr,"B2 = Ox%8.8X;\n",B2);
fprintf(stderr,"C2 = Ox%8.8X;\n",C2);
fprintf(stderr,"D2 = Ox%8.8X;\n\n",D2);
fprintf(stderr,"eps1= Ox%8.8X;\n",eps1);
fprintf(stderr,"eps2= Ox%8.8X;\n",eps2);
fprintf(stderr,"eps3= Ox%8.8X;\n",eps3);
fprintf(stderr,"eps4= Ox%8.8x;\n",eps4);
fprintf(stderr,"dC1 = Ox%8.8X;\n",dC1);
fprintf(stderr,"dd1 = Ox%8.8X;\n",dD1);
fprintf(stderr," **** gef. ****\n");
fprintf(stderr,"AA1 = Ox%8.8x;\n",AA1);
fprintf(stderr,"A1  = Ox%8.8x;\n",A1);
fprintf(stderr,"BB1 = Ox%8.8X;\n",BB1);
fprintf(stderr,"B1  = Ox%8.8x;\n",B1);
fprintf(stderr,"CC1 = Ox%8.8X;\n",CC1);
fprintf(stderr,"C1  = Ox%8.8X;\n",C1);
fprintf(stderr,"DD1 = Ox%8.8X;\n",DD1);
fprintf(stderr,"D1  = Ox%8.8x;\n",D1);
fprintf(stderr,"C   = Ox%8.8X;\n",C);
fprintf(stderr,"D   = Ox%8.8X;\n",D);

fprintf(stdout,"A2  =   Ox%8.8X;\n",A2);
fprintf(stdout,"B2  =   Ox%8.8X;\n",B2);
fprintf(stdout,"C2  =   Ox%8.8X;\n",C2);
fprintf(stdout,"D2  =   Ox%8.8X;\n",D2);
fprintf(stdout,"eps1= Ox%8.8x;\n",eps1);
fprintf(stdout,"eps2= Ox%8.8X;\n",eps2);
fprintf(stdout,"eps3= Ox%8.8X;\n",eps3);
```

```
fprintf(stdout,"eps4= Ox%8.8X;\n",eps4);
fprintf(stdout,"dC1 = Ox%8.8X;\n",dC1);
fprintf(stdout,"dD1 = Ox%8.8X;\n",dD1);
fprintf(stdout,"**** gef. ****\n");
fprintf(stdout,"AA1 = Ox%8.8X;\n",AA1);
fprintf(stdout,"A1  = Ox%8.8X;\n",A1);
fprintf(stdout,"BB1 = Ox%8.8X;\n",BB1);
fprintf(stdout,"B1  = Ox%8.8X;\n",B1);
fprintf(stdout,"CC1 = Ox%8.8X;\n",CC1);
fprintf(stdout,"C1  = Ox%8.8X;\n",C1);
fprintf(stdout,"DD1 = Ox%8.8X;\n",DD1);
fprintf(stdout,"D1  = Ox%8.8x;\n",D1);
fprintf(stdout,"C   = Ox%8.8x;\n",C);
fprintf(stderr,"D   = Ox%8.8X;\n",D);
exit(1);
      }
}
    }
       }
     }
  }
     }
        }
   }
      }


    if(flag==0)
      {
fprintf(stderr,"nichts gefunden\n");
      }
    if(flag==1)
      {
fprintf(stderr,"nichts gefunden +\n");
      }

    goto LABEL_A;
} /* End Main */


int suche()
{
    int i;
    static int index,count;
    static UL x;

    i=index;

    if(i==0)
      {
count=0;
      }

    if(i==32)
      {
if(count>=*max)
```

```
  {
     (*max)*=2;
     if((*feld=(UL*)realloc(*feld,(*max)*sizeof(UL)))==NULL)
       {
  printf("Nicht genuegend Speicher 1\n");
  exit(1);
       }

  }

(*feld)[count]=x;
count++;
return count;
     }

   index++;

/*   if(count >= MAX)
     {
index--;
return(count);
     }*/


   if((((F(x)-konst)>>i)&1)==0)
     {
suche();
     }

   if(count >= max_test)
     {
index--;
return count;
     }


   x^=1ul<<i;
   if((((F(x)-konst)>>i)&1)==0)
     {
suche();
     }

   x^=1ul<<i;
   index--;
   return count;
}

int sucheh()
{
   int i;
   static int index,count;
   static UL x;

   i=index;
```

```
   if(i==0)
     {
count=0;
     }

   if(i==32)
     {
if(count>=*max)
  {
     (*max)*=2;
     if((*feld=(UL*)realloc(*feld,(*max)*sizeof(UL)))==NULL)
       {
  printf("Nicht genuegend Speicher 2\n");
  exit(1);
       }
      }
(*feld)[count]=x;
count++;
return count;
     }

   index++;

   /*if(count >= MAX)
     {
index--;
return(count);
     }*/

   if((((H(x)-konst)>>i)&1)==0)
     {
sucheh();
     }

   if(count >= max_test)
     {
index--;
return count;
     }


   x=1ul<<i;

   if((((H(x)-konst)>>i)&1) == 0)
     {
sucheh();
     }

   x^=1ul<<i;
   index--;
   return count;
}

int sucheb()
{
```

```
   int i;
   static int index,count;
   static UL x;

   i=index;

   if(i==0)
     {
count=0;
     }

   if(i==32)
     {
if(count>=*max)
  {
     (*max)*=2;
     if((*feld=(UL*)realloc(*feld,(*max)*sizeof(UL)))==NULL)
       {
  printf("Nicht genuegend Speicher 3\n");
  exit(1);
       }

  }

(*feld)[count]=x;
count++;
return count;
     }

   index++;

   /*if(count >= MAX)
     {
index--;
return(count);
     }*/

   if((((F(x)-konst)>>i)&1)==0)
     {
sucheb();
     }

   if(count >= max_test)
     {
index--;
return count;
     }


   x^=1ul<<i;
   if((((F(x)-konst)>>i)&1)==0)
     {
sucheb();
     }
```

```
    x^=1ul<<i;
    index--;
    return count;
}
```

E.3 THIRD PHASE OF THE ATTACK ON MD5

This section of the appendix contains the C source code used in the third phase of the attack on MD5.

```
/* This program is a modification of the original program for connecting the
 * first two stages of the MD5 attack. The modification ensures that the
 * connection established is a valid connection.
 *
 * Original Author: Hans Dobbertin?
 * Modifications  : P.R. Kasselman
 * Date of Modifications: 30 September 1996
 * Filename: md5an13.c */

#include <stdio.h>
#include <stdlib.h>
#include <time.h>
#include "libtiming.h"

#define ulong unsigned long
#define shift(x,i) (ulong)((((ulong)x)<<(i))^(((ulong)x)>>(32-(i))))
#define f(x,y,z) ((x)&(y) | (~(x))&(z))
#define g(x,y,z) ((x)&(z) | (~(z))&(y))
#define h(x,y,z) ((x)^(y)^(z))
#define i(x,y,z) ((y)^((x)|(~(z))))

static FILE *f;

/* random number generator from Numerical Recipes in C p. 282 */
#define IM1 2147483563
#define IM2 2147483399
#define IA1 40014
#define IA2 40692
#define IQ1 53668
#define IQ2 52774
#define IR1 12211
#define IR2 3791
#define NTAB 32
#define NDIV (1+(IM1-1)/NTAB)

double ran2(long *idum)
{
    int j;
    long k;
    static long idum2 = 123456789;
    static long iy = 0;
    static long iv[NTAB];
    double temp;
```

```
   /* Initialization */

   if (*idum <= 0)
     {
/* make sure *idum != 0 */
if (*idum == 0)
  {
     *idum = 1;
  }

printf("Random generator initialized with %ld\n", *idum);
fprintf(f, "Random generator initialized with %ld\n", *idum);
idum2 = *idum;
for (j=NTAB+7; j>=0; j--)
  {
     /* idum = (IAllidum) % IM1*/
     k = *idum/IQ1;
     *idum = IA1*(*idum-k*IQ1)-k*IR1;
     if (*idum < 0)
       {
  *idum += IM1;
       }
     if (j < NTAB)
       {
  iv[j] = *idum;
       }
  }

iy = iv[0];
     }

   /* idum = (IA1*idum) % IM1 */

   k = *idum/IQ1;

   *idum = IA1*(*idum-k*IQ1)-k*IR1;

   if (*idum < 0)
     {
*idum += IM1;
     }

   /* idum2 = (IA2*idum) % IM2 */
   k = idum2/IQ2;
   idum2 = IA2*(idum2-k*IQ2)-k*IR2;

   if (idum2 < 0)
     {
idum2 += IM2;
     }

   j = iy/NDIV;
   iy = iv[j] - idum2;
   iv[j] = *idum;
```

```
   if (iy < 1)
     {
iy += IM1-1;
     }

   return (double)iy/IM1;
}

static long randstate;
void srand32(long seed)
{
   if (seed>0)
     {
randstate = -seed;
     } else {
randstate = seed;
     }
}

unsigned long rand32(void)
{
   return (unsigned long)(4294967296.0*ran2(&randstate));
}

main(int ac,char *av[])
{
   ulong A,B,B_basic,C,D;
   ulong A0,B0,C0,D0;
   ulong C31,B31;
   ulong A39,B35,C39,D39;
   ulong A43,B39,D43;
   ulong A35,C35,D35;
   ulong A19,B15,C15,D19;
   ulong A23,B19,C19,D23;
   ulong C27,B27,A31,D31;
   ulong        BB_12,CC_12,C2_34,BB_34;
   ulong A15,D15;
   ulong A27,B23,C23,D27;
   ulong A1,B1,C1,D1;
   ulong At0,Bt0,Ct0,Dt0,At1,Bt1,Ct1,Dt1,At2,Bt2,Ct2,Dt2;
   ulong test0,test1;
   ulong X0,X1,X2,X3,X4,X5,X6,X7,X8,X9,X10,X11,X12,X13,X14,X15,X0_12;
   ulong X8A,X8B;
   ulong konst1,konst2,konst3,konst4,konst5;
   int         gew=32,gew0,k,Versuch,vvv;
   unsigned int iterations34, count1, sum34_vvv;
   unsigned int ConnectionFlag;
   double       t1, t2, time_tot;

   ulong K0=0xd76aa478, K1=0xe8c7b756, K2=0x242070db, K3=0xc1bdceee;
   ulong K4=0xf57c0faf, K5=0x4787c62a, K6=0xa8304613, K7=0xfd469501;
   ulong K8=0x698098d8, K9=0x8b44f7af, K10=0xffff5bb1, K11=0x895cd7be;
   ulong K12=0x6b901122, K13=0xfd987193, K14=0xa679438e, K15=0x49b40821;
   ulong K16=0xf61e2562, K17=0xc040b340, K18=0x265e5a51, K19=0xe9b6c7aa;
```

```
    ulong K20=0xd62f105d, K21=0x02441453, K22=0xd8a1e681, K23=0xe7d3fbc8;
    ulong K24=0x21e1cde6, K25=0xc33707d6, K26=0xf4d50d87, K27=0x455a14ed;
    ulong K28=0xa9e3e905, K29=0xfcefa3f8, K30=0x676f02d9, K31=0x8d2a4c8a;
    ulong K32=0xfffa3942, K33=0x8771f681, K34=0x6d9d6122, K35=0xfde5380c;
    ulong K36=0xa4beea44, K37=0x4bdecfa9, K38=0xf6bb4b60, K39=0xbebfbc70;
    ulong K40=0x289b7ec6, K41=0xeaa127fa, K42=0xd4ef3085, K43=0x04881d05;
    ulong K44=0xd9d4d039, K45=0xe6db99e5, K46=0x1fa27cf8, K47=0xc4ac5665;
    ulong K48=0xf4292244, K49=0x432aff97, K50=0xab9423a7, K51=0xfc93a039;
    ulong K52=0x655b59c3, K53=0x8f0ccc92, K54=0xffeff47d, K55=0x85845dd1;

    int s0 = 7,s1 =12,s2 =17,s3 =22;
    int s4 = 7,s5 =12,s6 =17,s7 =22;
    int s8 = 7,s9 =12,s10=17,s11=22;
    int s12= 7,s13=12,s14=17,s15=22;
    int s16= 5,s17= 9,s18=14,s19=20;
    int s20= 5,s21= 9,s22=14,s23=20;
    int s24= 5,s25= 9,s26=14,s27=20;
    int s28= 5,s29= 9,s30=14,s31=20;
    int s32= 4,s33=11,s34=16,s35=23;
    int s36= 4,s37=11,s38=16,s39=23;
    int s40= 4,s41=11,s42=16,s43=23;
    int s44= 4,s45=11,s46=16,s47=23;
    int s48= 6,s49=10,s50=15,s51=21;
    int s52= 6,s53=10,s54=15,s55=21;

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


    srand32(atol(av[1]));
    f = fopen("md5_dob.dat", "w");

    /* Daten    Runde 12 */
    A15 = 0XFA08A191;
    D15 = 0xFE010000;
    A19 = 0x3079FC64;
    B15 = 0x7C091F7C;
    C15 = 0xFE00FFFF;
    D19 = 0x00000000;
    B19 = 0x007DE57C;
    C19 = 0xA83AE412;
    A23 = 0x001003E0;
    A23 = 0x20200380;
    A23 = 0x20001380;

    A15  = 0x45A9B039;
    D15  = 0xFE010000;
    C15  = 0xFE00FFFF;
    B15  = 0x681702E8;
    A19  = 0xA27F00A0;
    D19  = 0x00000000;
    C19  = 0x5E4D2843;
```

```
   B19  = 0xCC7E11F4;
   A23  = 0x8A14C72D;

   /* Daten Runnde 34 */
   A43 = 0x7DC2498C;
   B39 = 0x40040E86;

   /* These chaining variables are not used in this program and may therefore
    * be ommitted */
   /*   C2_34 = 0x12F37166;
   D43 = 0xBDC18273; */
   A39 = 0x579DA695;
   B35 = 0xFFFFFFFF;
   C39 = 0xC783FA3D;
   D39 = 0x8DE34058;
   C35 = 0x00028800;
   D35 = 0x3420f162;

   /* Determine message words from the first part of the attack */
   X1 = shift(A19-B15,32-s16)-K16-A15-g(B15,C15,D15);
   X6 = shift(D19-A19,32-s17)-K17-D15-g(A19,B15,C15);
   X11 = shift(C19-D19,32-s18)-K18-C15-g(D19,A19,B15);
   X0 = shift(B19-C19,32-s19)-K19-B15-g(C19,D19,A19);
   X5 = shift(A23-B19,32-s20)-K20-A19-g(B19,C19,D19);

   /* Determine message words from the second part of the attack */
   X4 = shift(D39-A39,32-s37)-K37-D35-h(A39,B35,C35);
   X7 = shift(C39-D39,32-s38)-K38-C35-h(D39,A39,B35);
   X10= shift(B39-C39,32-s39)-K39-B35-h(C39,D39,A39);
   X13= shift(A43-B39,32-s40)-K40-A39-h(B39,C39,D39);

   /* This is a constant to be used later on (when confirming a collision) */
   BB_12 = shift(B15-C15,32-s15)-K15-f(C15,D15,A15);
   /* This is a constant to be used later on for confirmation of collision-
s */
   CC_12 = shift(C15-D15,32-s14)-K14;

   /* Calculate chaining variables to be used when establishing a connection */
   D23 = shift(D19+g(A23,B19,C19)+X10+K21,s21)+A23;
   A35 = shift(A39-B35,32-s36)-K36-X1-h(B35,C35,D35);

   /* This constant is used at a later stage (X14 is not included in
      this expression) */
   BB_34 = shift(B35-C35,32-s35)-K35-h(C35,D35,A35);

   D43 = shift(D39+h(A43,B39,C39)+X0 +K41,s41)+A43;

   /* Data of inner collisions 1-2 and 3-4 should be chosen such that X0
    is the same in both inner collisions.  This was not the case in the first
    version of the attack, and the reason why K19 had to be adapted. */

   sum34_vvv = 0;
   iterations34 = 50;
   time_tot = 0;
```

```
   /* The purpose of this loop is to find the average time required
    * to find a collision */

/*   for (count1=0; count1<iterations34; count1++) */
     {
/* Number of inner collisions 1-2/connections to find a collision */
vvv = 0;

printf("\nSearch %u started\n", count1+1);
fflush(stdout);
fprintf(f, "State random generator = %ld\n", randstate);

while(1)
  {
     /* Look for a collision */
     do
       {
  /* Look for inner collision in round 1-2 */
  X15 = rand32();

  /* Determine the last four chaining variables that will
   * result in a internal collision for the first two rounds */

  C23 = shift(C19+g(D23,A23,B19)+X15+K22,s22)+D23;
  B23 = shift(B19+g(C23,D23,A23)+X4 + K23,s23)+C23;

  /*A27 = 0xffffffff;*/
  A27 = B23;
  X9 = shift(A27-B23,32-s24)-g(B23,C23,D23)-A23-K24;
  D27 = 0xffffffff;
  X14 = shift(D27-A27,32-s25)-g(A27,B23,C23)-D23-K25;

  /* Check whether inner collision 1-2 */
  A = A15;
  B = BB_12-X15;
  C = CC_12-X14-f(D15,A15,B);
  D = D15;
  C = shift(C+f(D,A,B)+X14+((ulong)1<<9)+K14,s14)+D;
  B = shift(B+f(C,D,A)+X15+K15,s15)+C;
  A = shift(A+g(B,C,D)+X1 +K16,s16)+B;
  D = shift(D+g(A,B,C)+X6 +K17,s17)+A;
  C = shift(C+g(D,A,B)+X11+K18,s18)+D;
  B = shift(B+g(C,D,A)+X0 +K19,s19)+C;
  A = shift(A+g(B,C,D)+X5 +K20,s20)+B;
  D = shift(D+g(A,B,C)+X10+K21,s21)+A;
  C = shift(C+g(D,A,B)+X15+K22,s22)+D;
  B = shift(B+g(C,D,A)+X4 +K23,s23)+C;
  A = shift(A+g(B,C,D)+X9 +K24,s24)+B;
  D = shift(D+g(A,B,C)+X14+((ulong)1<<9)+K25,s25)+A;

  /*if( A==A27 && B==B23 && C==C23 && D==D27 )
    {
       printf("%8.8X %8.8X %8.8X %8.8X\n", A-A27, B-B23, C-C23, D-D27);
    }*/
```

```
  /* Proceed if an internal collision for 1-2 was found */
  } while( A!=A27 || B!=B23 || C!=C23 || D!=D27 );

     /* begin connection */

     D35 = D35;
     A35 = A35;
     B31 = BB_34-X14;
     C31 = shift(C35-D35,32-s34)-h(D35,A35,B35)-X11-K34;

     Versuch=0;

     /* These are constants for the while loop that follows.
      * Pre-computation saves some time */

     B_basic = rand32();
     konst1 = A27+X13+K28;
     konst2 = shift(A35-B31,32-s32)-X5-K32;
     konst3 = B31^C31;
     konst4 = -X7-K30;

     ConnectionFlag = 0;
     while(ConnectionFlag == 0)
       {
/* Looking for a connection */

  B27 = B_basic^((ulong)1 << (rand32()&0x1f));
  A31 = shift(B27+konst1,s28)+B27;
  D31 = (konst2-A31)^konst3;
  C27 = shift(C31-D31,32-s30)-g(D31,A31,B27)+konst4;

  X8A = shift(B27-C27,32-s27)-B23-g(C,D27,A27)-K27;
  X8B = shift(D35-A35,32-s33)-D31-h(A35,B31,C31)-K33;
  test0 = X8B-X8A;

  if (test0==0)
    {
       /* you are really lucky */
       X12 = shift(B31-C31,32-s31)-B27-g(C31,D31,A31)-K31;
       X2 = shift(D31-A31,32-s29)-D27-g(A31,B27,C27)-K29;
       X3 = shift(C27-D27,32-s26)-C23-g(D27,A27,B23)-K26;
       X8 = shift(B27-C27,32-s27)-B23-g(C27,D27,A27)-K27;

       /*printf("Lucky You\n")*/
       /* Immediatle exit from the while loop */
       break;
    }

  /* Compute the Hamming Distance */
  /* Perform an iterative aproach */
  for (gew0=0, test1=test0; test1; test1>>=1)
    gew0 += test1&1;
  Versuch++;
  if (Versuch>30)     /*originally 150*/
    {
```

```
        /* Effectively restart the process */
        Versuch = 0;
        gew = 32;
        B_basic = rand32();

        /* Restart from the beginning of the loop */
        continue;
    }

  /* Continuous approximation techniques */
  if (gew0-2<gew) /* originally gew0-1 */
    {
        gew = gew0;
        if (gew>11) /* originally 7 */
 {
    /* If X8A and X8B are closer to each other than
     * before retain the value of B as the new
     * basic value */

    B_basic =B27;

    /* Restart from the beginning of the loop */
    continue;
 }

/* Verbesserung */
/* Check whether B23 is changed by same amount as CO */
        /*if( (C23 & A23) != ((C23-test0) & A23))
 {
    break;
 }*/

        /* Change C23 and consequently recalculate B23 */
        C23 = C23-test0;
        B23 = shift(B19+g(C23,D23,A23)+X4 + K23,s23)+C23;

        /* Recompute D27 */
        D27 = shift(D23+g(A27,B23,C23)+X14+K25,s25)+A27;

        /* A, B, C, D not changed */
        /* Check whether A still only depends on B and A27 */
if( g(B27,C27,D27) != B27)
 {
    break;
 }

        /* Sanity Check */
        X8A = shift(B27-C27,32-s27)-B23-g(C27,D27,A27)-K27;
        X8B = shift(D35-A35,32-s33)-D31-h(A35,B31,C31)-K33;

        if(test0 == 0)
 {
    /* Reset the condition and determine if a collision
     * for both round1 1-2 and 3-4 holds */
```

```
    test0 = 1;

    X15 = shift(C23-D23,32-s22)-g(D23,A23,B19)-C19-K22;
    X9 = shift(A27-B23,32-s24)-g(B23,C23,D23)-A23-K24;

    /* Good Fiddling */
    X12 = shift(B31-C31,32-s31)-B27-g(C31,D31,A31)-K31;
    X2 = shift(D31-A31,32-s29)-D27-g(A31,B27,C27)-K29;
    X3 = shift(C27-D27,32-s26)-C23-g(D27,A27,B23)-K26;
    X8 = shift(B27-C27,32-s27)-B23-g(C27,D27,A27)-K27;

    /*printf("Good Fiddle\n");*/

    /* check whether inner collision 1-2 holds */
    A = A15;
    B = BB_12-X15;
    C = CC_12-X14-f(D15,A15,B);
    D = D15;
    C = shift(C+f(D,A,B)+X14+((ulong)1<<9)+K14,s14)+D;
    B = shift(B+f(C,D,A)+X15+K15,s15)+C;
    A = shift(A+g(B,C,D)+X1 +K16,s16)+B;
    D = shift(D+g(A,B,C)+X6 +K17,s17)+A;
    C = shift(C+g(D,A,B)+X11+K18,s18)+D;
    B = shift(B+g(C,D,A)+X0 +K19,s19)+C;
    A = shift(A+g(B,C,D)+X5 +K20,s20)+B;
    D = shift(D+g(A,B,C)+X10+K21,s21)+A;
    C = shift(C+g(D,A,B)+X15+K22,s22)+D;
    B = shift(B+g(C,D,A)+X4 +K23,s23)+C;
    A = shift(A+g(B,C,D)+X9 +K24,s24)+B;
    D = shift(D+g(A,B,C)+X14+((ulong)1<<9)+K25,s25)+A;

    if( A!=A27 || B!=B23 || C!=C23 || D!=D27 )
      {
/* If an inner collision does not hold
 * restart by finding a new inner collision
 * for rounds1-2 */
break;
      }


    /* Check Propagation */
    C = shift(C+g(D,A,B)+X3+K26,s26)+D;
    B = shift(B+g(C,D,A)+X8 +K27,s27)+C;
    A = shift(A+g(B,C,D)+X13 +K28,s28)+B;
    D = shift(D+g(A,B,C)+X2+K29,s29)+A;
    C = shift(C+g(D,A,B)+X7+K30,s30)+D;
    B = shift(B+g(C,D,A)+X12 +K31,s31)+C;
    A = shift(A+h(B,C,D)+X5 +K32,s32)+B;
    D = shift(D+h(A,B,C)+X8+K33,s33)+A;

    /* Verify the existence of a connection */
    if( A!=A35 || B!=B31 || C!=C31 || D!=D35 )
      {
break;
      }
```

```
    test0 = 0;
    break;
}
    }
       }

     if(test0!=0)
       {
/* The Verbesserung failed, so start with a new inner
 * collision*/
continue;
       }

    /* connection found */
    vvv++;
    /* end connect */

    if (!(vvv%20))
      {
printf("Connection and inner collision 1-2 %d found\r", vvv);
fflush(stdout);
      }

    /* Check whether inner collision 3-4 */

    /* Daten Runde 34 */
    C = C39;
    B = B39;
    A = A43;
    D = D43;
    C = shift(C+h(D,A,B)+X3 +K42,s42)+D;
    B = shift(B+h(C,D,A)+X6 +K43,s43)+C;
    A = shift(A+h(B,C,D)+X9 +K44,s44)+B;
    D = shift(D+h(A,B,C)+X12+K45,s45)+A;
    C = shift(C+h(D,A,B)+X15+K46,s46)+D;
    B = shift(B+h(C,D,A)+X2 +K47,s47)+C;
    A = shift(A+i(B,C,D)+X0 +K48,s48)+B;
    D = shift(D+i(A,B,C)+X7 +K49,s49)+A;
    C = shift(C+i(D,A,B)+X14+K50,s50)+D;
    A0 = A;
    B0 = B;
    C0 = C;
    D0 = D;

    B = B31;
    A = A35;
    D = D35;
    C = C35;
    B = shift(B+h(C,D,A)+X14+((ulong)1<<9)+K35,s35)+C;
    A = shift(A+h(B,C,D)+X1 +K36,s36)+B;
    D = shift(D+h(A,B,C)+X4 +K37,s37)+A;
    C = shift(C+h(D,A,B)+X7 +K38,s38)+D;
    B = shift(B+h(C,D,A)+X10+K39,s39)+C;
    A = shift(A+h(B,C,D)+X13+K40,s40)+B;
```

```
   D = shift(D+h(A,B,C)+X0 +K41,s41)+A;
   C = shift(C+h(D,A,B)+X3 +K42,s42)+D;
   B = shift(B+h(C,D,A)+X6 +K43,s43)+C;
   A = shift(A+h(B,C,D)+X9 +K44,s44)+B;
   D = shift(D+h(A,B,C)+X12+K45,s45)+A;
   C = shift(C+h(D,A,B)+X15+K46,s46)+D;
   B = shift(B+h(C,D,A)+X2 +K47,s47)+C;
   A = shift(A+i(B,C,D)+X0 +K48,s48)+B;
   D = shift(D+i(A,B,C)+X7 +K49,s49)+A;
   C = shift(C+i(D,A,B)+X14+((ulong)1<<9)+K50,s50)+D;
   A ^= A0;
   B ^= B0;
   C ^= C0;
   D ^= D0;

   if(B==0)
     {
fprintf(f,"%8.8X %8.8X %8.8X %8.8X %5d\n",A,B,C,D,vvv);
fflush(f);
printf("%5d: %8.8X %8.8X %8.8X %8.8X\n", vvv, A, B, C, D);
fflush(stdout);
     }

   if( A==0 && B==0 && C==0 && D==0)
     {
/* Step backwards through round 1 to find IV */
A = A15;
B = B15;

C = C15;
D = D15;
B = shift(B-C,32-s15)-K15-X15-f(C,D,A);
C = shift(C-D,32-s14)-K14-X14-f(D,A,B);
D = shift(D-A,32-s13)-K13-X13-f(A,B,C);
A = shift(A-B,32-s12)-K12-X12-f(B,C,D);
B = shift(B-C,32-s11)-K11-X11-f(C,D,A);
C = shift(C-D,32-s10)-K10-X10-f(D,A,B);
D = shift(D-A,32-s9)-K9-X9-f(A,B,C);
A = shift(A-B,32-s8)-K8-X8-f(B,C,D);
B = shift(B-C,32-s7)-K7-X7-f(C,D,A);
C = shift(C-D,32-s6)-K6-X6-f(D,A,B);
D = shift(D-A,32-s5)-K5-X5-f(A,B,C);
A = shift(A-B,32-s4)-K4-X4-f(B,C,D);
B = shift(B-C,32-s3)-K3-X3-f(C,D,A);
C = shift(C-D,32-s2)-K2-X2-f(D,A,B);
D = shift(D-A,32-s1)-K1-X1-f(A,B,C);
A = shift(A-B,32-s0)-K0-X0-f(B,C,D);

fprintf(f,"**** %d Versuche ****\n", vvv);
fprintf(f,"**** Collision %u for MDS compress ****\n", count1+1);
fprintf(f,"**** IV = (%8.8X,%8.8X,%8.8X,%8.8X) ****\n",A,B,C,D);
fprintf(f,"***1 X14' = X14+2^9 ****\n");

fprintf(f,"X0 = 0x%8.8x;\n",X0);
fprintf(f,"X1 = 0x%8.8X;\n",X1);
```

```
  fprintf(f,"X2 = 0x%8.8X;\n",X2);
  fprintf(f,"X3 = 0x%8.8X;\n",X3);
  fprintf(f,"X4 = 0x%8.8X;\n",X4);
  fprintf(f,"X5 = 0x%8.8X;\n",X5);
  fprintf(f,"X6 = 0x%8.8X;\n",X6);
  fprintf(f,"X7 = 0x%8.8x;\n",X7);
  fprintf(f,"X8 = 0x%8.8X;\n",X8);
  fprintf(f,"X9 = 0x%8.8X;\n",X9);
  fprintf(f,"X10= 0x%8.8X;\n",X10);
  fprintf(f,"X11= 0x%8.8X;\n",X11);
  fprintf(f,"X12= 0x%8.8X;\n",X12);
  fprintf(f,"X13= 0x%8.8X;\n",X13);
  fprintf(f,"X14= 0x%8.8X;\n",X14);
  fprintf(f,"X15=       0x%8.8X;\n",X15);
  fflush(f);

  /* Break from the while(1) loop if a collision for 3-4 was
   * found */
  break;
       }

  }


/* solution found */

sum34_vvv += vvv;

fprintf(f, "\ncollision %u found after %d trials\n", count1+1, vvv);
fflush(f);
printf("\nCollision %u found after %d trials\n", count1+1, vvv);
fflush(stdout);
     }

   fprintf(f, "Mean number ic-12 for a ic-34: %u\n", sum34_vvv/iterations34);
   fprintf(f, "Mean time for a collision: %f seconds\n", time_tot/iterations34);
   fclose(f);

   PrintUserTime();

   return 0;

 }
```

APPENDIX F: SOURCE CODE: COLLISIONS FOR FIRST ROUND OF SHA

This Appendix contains an implementation of the attack on the first round of SHA. The implementation is written in ANSI-C.

```
/* This program is used to investigate the effect of the message expansion
 * algorithm used in SHA (not SHA-1). The results obtained from the analysis
 * in sha01 and sha02 is verified. In particular it is verified if a
 * collision can be obtained for the first round of SHA.
 *
 * The difference pattern has a defining length of 6.
 * The pattern is:  11 12 13 14 15 16
 *                   1  1  1  0  0  1
 *
 * This program extends the results in sha06.c. Specifically a message is
 * constructed which results in a collision after one round of SHA.
 *
 * Date: 14/11/97
 * Author: P.R. Kasselman
 * Filename: sha07.c */

#include <stdlib.h>
#include <stdio.h>
#include <math.h>
#include <time.h>

unsigned int Rotate(unsigned int X, unsigned int s);
unsigned int RotateRight(unsigned int X, unsigned int s);
void PrintBin(unsigned int j);
unsigned int DefLen(unsigned int j);
unsigned int SHA_F1(unsigned int B, unsigned int C, unsigned int D);
void UpdateChain(unsigned int Temp, unsigned int *A,
 unsigned int *B, unsigned int *C,
 unsigned int *D, unsigned int *E, int i);
void ReverseUpdateChain(unsigned int Temp, unsigned int *A,
unsigned int *B, unsigned int *C,
unsigned int *D, unsigned int *E, int i);

#define K1 0x5A827999

int main()
{
   unsigned int i,j;
   unsigned int Temp1, Temp2, TempInt, Stop, Iteration;
   unsigned int M1[80], M2[80];
   unsigned int A[80], B[80], C[80], D[80], E[80];
   unsigned int At[80], Bt[80], Ct[80], Dt[80], Et[80];
   time_t TheTime;

   /* Seed Random number generator */
   TheTime = time(NULL);
   srandom(TheTime);

   /* Initialise Chaining varaibles */
   Temp1 = 0;
   Temp2 = 0;
```

```
   A[0] = 0x67452301;
   B[0] = 0xEFCDAB89;
   C[0] = 0x98BADCFE;
   D[0] = 0x10325476;
   E[0] = 0xC3D2E1F0;

   At[0] = 0x67452301;
   Bt[0] = 0xEFCDAB89;
   Ct[0] = 0x98BADCFE;
   Dt[0] = 0x10325476;
   Et[0] = 0xC3D2E1F0;

   for(i=0; i<4; i++)
     {
UpdateChain(Temp1, A, B, C, D, E, i);
UpdateChain(Temp2, At, Bt, Ct, Dt, Et, i);
     }

   /* Determine if collision holds */

   Stop = 1;
   Iteration = 0;

   while(Stop != 0)
     {
/* Specify required differences in messages */

for(i=11; i<14; i++)
  {
     M1[i] = (random()^Rotate(random(),1)) & 0xfffffffe;
     M2[i] = M1[i];
  }

M1[11] = M1[11] | 0x00000001;
M2[12] = M2[12] | 0x00000001;
M2[13] = M2[13] | 0x00000001;

M1[i] = (random()^Rotate(random(),1)) & 0xfffffffe;
M2[i] = M1[i];

printf("%8.8X %8.8X\n", M1[11], M2[11]);
printf("%8.8X %8.8X\n", M1[12], M2[12]);
printf("%8.8X %8.8X\n", M1[13], M2[13]);

/* Initialise the chaining variables */

printf("\nConfirmation of Collision\n");

/* Step 1 */

Temp1 = 0x00000000;
Temp2 = 0xffffffff;

A[11] = random() ^ Rotate(random(),1);
```

```
B[11] = A[11] + RotateRight(1,30);
C[11] = -1-Rotate(0,5)-SHA_F1(Temp1,Rotate(A[11],30),Rotate(B[11],30))-K1-
M1[13];
D[11] = 0-Rotate(Temp1,5)-SHA_F1(A[11],Rotate(B[11],30),C[11])-K1- M1[12];
E[11] = Temp1-Rotate(A[11],5)-SHA_F1(B[11],C[11],D[11])-K1-M1[11];

At[11] = A[11];
Bt[11] = B[11];
Ct[11] = -1-Rotate(0,5)-SHA_F1(Temp2,Rotate(At[11],30),Rotate(Bt[11],30))-
K1-M2[13];
Dt[11] = 0 - Rotate(Temp2,5) - SHA_F1(At[11],Rotate(Bt[11],30),Ct[11]) - K1 -
M2[12];
Et[11] = Temp2-Rotate(At[11],5)-SHA_F1(Bt[11],Ct[11],Dt[11])-K1-M2[11];

printf("Temp1 - Temp2  - (M1[11] - M2[11]) = %8.8X\n",
       Temp1-Temp2 - (M1[11] - M2[11]));

UpdateChain(Temp1, A, B, C, D, E, 11);
UpdateChain(Temp2, At, Bt, Ct, Dt, Et, 11);

/* Step 2 */

printf("A[12] - At[12] + (M1[12] - M2[12]) = %8.8X\n",
       A[12]-At[12]+(M1[12]-M2[12]));

Temp1 = 0x00000000;
Temp2 = 0x00000000;

UpdateChain(Temp1, A, B, C, D, E, 12);
UpdateChain(Temp2, At, Bt, Ct, Dt, Et, 12);

/* Step 3 */

TempInt = SHA_F1(B[13],C[13],D[13]) - SHA_F1(Bt[13],Ct[13],Dt[13]);

printf("F1(B[13],C[13],D[13])-F1(Bt[13],Ct[13],Dt[13])+(M1[13]-M2[13]) = %8.8X\n",
       TempInt + M1[13] - M2[13]);

Temp1 = 0xffffffff;
Temp2 = 0xffffffff;

UpdateChain(Temp1, A, B, C, D, E, 13);
UpdateChain(Temp2, At, Bt, Ct, Dt, Et, 13);

/* Step 4 */

TempInt = SHA_F1(B[14],C[14],D[14]) - SHA_F1(Bt[14],Ct[14],Dt[14]);

printf("F1(B[14],C[14],D[14])-F1(Bt[14],Ct[14],Dt[14])+(M1[14]-M2[14]) = %8.8X\n",
       TempInt + M1[14] - M2[14]);

Temp1 = random() ^ Rotate(random(),1);
Temp2 = Temp1;

UpdateChain(Temp1, A, B, C, D, E, 14);
```

```
UpdateChain(Temp2, At, Bt, Ct, Dt, Et, 14);

/* Step 5 */

TempInt = SHA_F1(B[15],C[15],D[15]) - SHA_F1(Bt[15],Ct[15],Dt[15]);

printf("F1(B[15],C[15],D[15])-F1(Bt[15],Ct[15],Dt[15])+(M1[15]-M2[15]) = %8.8X\n",
       TempInt + M1[15] - M2[15]);

Temp1 = random() ^ Rotate(random(),1);
Temp2 = Temp1;

UpdateChain(Temp1, A, B, C, D, E, 15);
UpdateChain(Temp2, At, Bt, Ct, Dt, Et, 15);

/* Step 6 */

TempInt = E[16] - Et[16];

printf("E[16]-Et[16]+(M1[16]-M2[16]) = %8.8X\n",
       TempInt + M1[16] - M2[16]);

/* Work forward */

for(i=16; i<20; i++)
  {
     Temp1 = random() ^ Rotate(random(),1);
     Temp2 = Temp1;

     UpdateChain(Temp1, A, B, C, D, E, i);
     UpdateChain(Temp2, At, Bt, Ct, Dt, Et, i);
  }

/* Work back to meet the initial values */

for(i=10; i>=5; i--)
  {
     Temp1 = random()^Rotate(random(),1);
     Temp2 = Temp1;

     ReverseUpdateChain(Temp1, A, B, C, D, E, i+1);
     ReverseUpdateChain(Temp2, At, Bt, Ct, Dt, Et, i+1);
  }

for(i=4; i>=1; i--)
  {
     ReverseUpdateChain(E[i], A, B, C, D, E, i+1);
     ReverseUpdateChain(Et[i], At, Bt, Ct, Dt, Et, i+1);
  }

/* Reconstruct message */
for(i=0; i<16; i++)
  {
     M1[i] = A[i+1]-Rotate(A[i],5)-SHA_F1(B[i],C[i],D[i])-E[i]-K1;
     M2[i] = At[i+1]-Rotate(At[i],5)-SHA_F1(Bt[i],Ct[i],Dt[i])-Et[i]-K1;
```

```
  }

/* Expand Message */
for(i=16; i<21; i++)
  {
     M1[i] = (M1[i-3] ^ M1[i-8] ^ M1[i-14] ^ M1[i-16]);
     M2[i] = (M2[i-3] ^ M2[i-8] ^ M2[i-14] ^ M2[i-16]);
  }

/* Determine the common hash value */
for(i=0; i<20; i++)
  {
     Temp1 = Rotate(A[i],5)+SHA_F1(B[i],C[i],D[i])+E[i]+M1[i]+K1;
     UpdateChain(Temp1, A, B, C, D, E, i);

     Temp2 = Rotate(At[i],5)+SHA_F1(Bt[i],Ct[i],Dt[i])+Et[i]+M2[i]+K1;
     UpdateChain(Temp2, At, Bt, Ct, Dt, Et, i);
  }

i--;
Stop = (A[i]^At[i])+(B[i]^Bt[i])+(C[i]^Ct[i])+(D[i]^Dt[i])+(E[i]^Et[i]);
Iteration++;
     }

   printf("\nChaining Varaibles\n");
   printf("\tA\tB\tC\tD\tE\n");
   for(i=0; i<20; i++)
     {
printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X\n",i,A[i],B[i],C[i],D[i],E[i]);
     }

   printf("\tAt\tBt\tCt\tDt\tEt\n");
   for(i=0; i<20; i++)
     {
printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X\n",i,At[i],Bt[i],Ct[i],Dt[i],Et[i]);
     }

   printf("\nDifference Between Messages\n");
   for(i=0; i<20; i++)
     {
printf("M1[%d] - M2[%d] = %8.8X\n", i, i, M1[i] - M2[i]);
     }

   printf("\nCollision Message\n");
   for(i=0; i<20; i++)
     {
printf("M1[%d] = %8.8X \t M2[%d] = %8.8X\n", i, M1[i], i, M2[i]);
     }

   i = 19;
   printf("\nCommon Hash Value for First Round\n");
   printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X\n",i,A[i],B[i],C[i],D[i],E[i]);
   printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X\n",i,At[i],Bt[i],Ct[i],Dt[i],Et[i]);

   printf("\nDifference in hash values\n");
```

```
   printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X\n",
  i,A[i]^At[i],B[i]^Bt[i],C[i]^Ct[i],D[i]^Dt[i],E[i]^Et[i]);

   printf("Number of Iterations : %d\n", Iteration);
   return(0);
}

   unsigned int SHA_F1(unsigned int B, unsigned int C, unsigned int D)
     {
   return((B&C) | (~B&D));
}

unsigned int Rotate(unsigned int X, unsigned int s)
{
   unsigned int temp;

   temp = X;
   X = (X << s) | (temp >> (32-s));
   return(X);
}

unsigned int DefLen(unsigned int j)
{
   unsigned int i, TempInt;

   for(i=0; i<32; i++)
     {
if(((j >> (31-i)) & 0x00000001) == 1)
  {
     TempInt = i;
     break;
  }
     }

   for(i=31; i>=0; i--)
     {
if(((j >> (31-i)) & 0x00000001) == 1)
  {
     TempInt = i-TempInt;
     break;
  }
     }

   return(TempInt);
}

void PrintBin(unsigned int j)
{
   unsigned int i;

   for(i=0; i<32; i++)
     {
printf("%d", (j >> (31-i)) & 0x00000001);
     }
```

```
}

void UpdateChain(unsigned int Temp, unsigned int *A,
 unsigned int *B, unsigned int *C,
 unsigned int *D, unsigned int *E, int i)
{
   E[i+1] = D[i];
   D[i+1] = C[i];
   C[i+1] = Rotate(B[i],30);
   B[i+1] = A[i];
   A[i+1] = Temp;
}

void ReverseUpdateChain(unsigned int Temp, unsigned int *A,
unsigned int *B, unsigned int *C,
unsigned int *D, unsigned int *E, int i)
{
   A[i-1] = B[i];
   B[i-1] = RotateRight(C[i],30);
   C[i-1] = D[i];
   D[i-1] = E[i];
   E[i-1] = Temp;
}

unsigned int RotateRight(unsigned int X, unsigned int s)
{
   unsigned int temp;

   temp = X;
   X = (X >> s) | (temp << (32-s));
   return(X);
}
```

APPENDIX G: SOURCE CODE: IMPLEMENTATION OF HAVAL ATTACK

This appendix contains an implementation of the attack on HAVAL as described in Chapter 10.

```
/*
 * The analysis for the last two rounds of HAVAL is concidered.
 *
 * This program verifies the existence of collisions for the last two
 * rounds of three round HAVAL.
 *
 * Author: P.R. Kasselman
 * Date: 24/04/1999
 * Filename: haval29.c */

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>

unsigned int Rotate(unsigned int X, unsigned int s);
unsigned int RotateRight(unsigned int X, unsigned int s);

unsigned int FF2(unsigned int B, unsigned int C, unsigned int D,
       unsigned int E, unsigned int F, unsigned int G,
       unsigned int H);

unsigned int FF3(unsigned int B, unsigned int C, unsigned int D,
       unsigned int E, unsigned int F, unsigned int G,
       unsigned int H);

unsigned int InverseStep(unsigned int H2, unsigned int A1, unsigned int B1,
 unsigned int C1, unsigned int D1, unsigned int E1,
 unsigned int F1, unsigned int G1, unsigned int H1,
 unsigned int K);

/*#define MAX 0x00100000*/
#define MAX 100000
#define MAX_LIMIT 1000

#define RFACT    0
#define RFACT7   7
#define RFACT11 11

#define AA    0
#define BB    1
#define CC    2
#define DD    3
#define EE    4
#define FF    5
#define GG    6
#define HH    7

int main()
{
   char TempChar;
```

```
    unsigned int i, j;
    unsigned int Test, Count1;
    unsigned int DeltaW19, W19, Wt19;
    unsigned int DeltaH56;
    unsigned int Ht56, H56;
    unsigned int Bt62, B62, Ct61, C61, Dt60, D60;
    unsigned int Et59, E59, Ft58, F58, Gt57, G57;
    unsigned int C62, D62, E62, F62, G62, H62;
    unsigned int B61, D61, E61, F61, G61, H61;
    unsigned int B60, C60, E60, F60, G60, H60;
    unsigned int B59, C59, D59, F59, G59, H59;
    unsigned int B58, C58, D58, E58, G58, H58;
    unsigned int B57, C57, D57, E57, F57, H57;
    unsigned int B56, C56, D56, E56, F56, G56;
    unsigned int DataH56[MAX];
    unsigned int Chain[8][96];
    unsigned int W[32], Wt[32];
    unsigned int K1[32] = {0x452821E6L, 0x38D01377L, 0xBE5466CFL, 0x34E90C6CL,
       0xC0AC29B7L, 0xC97C50DDL, 0x3F84D5B5L, 0xB5470917L, 0x9216D5D9L,
       0x8979FB1BL, 0xD1310BA6L, 0x98DFB5ACL, 0x2FFD72DBL, 0xD01ADFB7L,
       0xB8E1AFEDL, 0x6A267E96L, 0xBA7C9045L, 0xF12C7F99L, 0x24A19947L,
       0xB3916CF7L, 0x0801F2E2L, 0x858EFC16L, 0x636920D8L, 0x71574E69L,
       0xA458FEA3L, 0xF4933D7EL, 0x0D95748FL, 0x728EB658L, 0x718BCD58L,
       0x82154AEEL, 0x7B54A41DL, 0xC25A59B5L};
    unsigned int K2[32] = {0x9C30D539L, 0x2AF26013L, 0xC5D1B023L, 0x286085F0L,
       0xCA417918L, 0xB8DB38EFL, 0x8E79DCB0L, 0x603A180EL, 0x6C9E0E8BL,
       0xB01E8A3EL, 0xD71577C1L, 0xBD314B27L, 0x78AF2FDAL, 0x55605C60L,
       0xE65525F3L, 0xAA55AB94L, 0x57489862L, 0x63E81440L, 0x55CA396AL,
       0x2AAB10B6L, 0xB4CC5C34L, 0x1141E8CEL, 0xA15486AFL, 0x7C72E993L,
       0xB3EE1411L, 0x636FBC2AL, 0x2BA9C55DL, 0x741831F6L, 0xCE5C3E16L,
       0x9B87931EL, 0xAFD6BA33L, 0x6C24CF5CL};
    unsigned int Ord2[32] = {5, 14, 26, 18, 11, 28, 7, 16, 0, 23, 20, 22, 1, 10,
       4, 8, 30, 3, 21, 9, 17, 24, 29, 6, 19, 12, 15, 13, 2, 25, 31, 27};
    unsigned int Ord3[32] = {19, 9, 4, 20, 28, 17, 8, 22, 29, 14, 25, 12, 24,
       30, 16, 26, 31, 15, 7, 3, 1, 0, 18, 27, 13, 6, 21, 10, 23, 11, 5, 2};
    unsigned int A1, B1, C1, D1, E1, F1, G1, H1, TempInt;
    time_t TheTime;
    FILE *fp1, *fp2;

    TheTime = time(NULL);
    srandom(TheTime);

    DeltaW19  = 0xaaaaaaab;
    DeltaH56 = DeltaW19;

    W19 = random()^Rotate(random(),1);
    Wt19 = W19 - DeltaW19;

    Count1 = 0;

    for(i=0; i<MAX; i++)
      {
Test = 1;
while(Test != 0)
  {
```

```
      Count1++;
      H56 = random()^Rotate(random(),1);
      Ht56 = H56 - DeltaH56;

      Test = (RotateRight(H56,RFACT11) - RotateRight(Ht56,RFACT11)) +
        (DeltaW19);

  }
DataH56[i] = H56;
      }

   printf("Equation (9) = %8.8X\n", (RotateRight(H56,RFACT11) + W19) - (Ro-
tateRight(Ht56,RFACT11) + Wt19));
   printf("\nAverage number of Iterations: %lf\n\n", ((double)(Count1))/MAX);

   /* Determine if (8) holds */
   B62 = H56;
   Bt62 = H56 - DeltaH56;
   C62 = H56|Ht56;
   D62 = ~(H56|Ht56);
   E62 = H56|Ht56;
   F62 = H56|Ht56;
   G62 = ~(H56|Ht56);
   H62 = random() ^ Rotate(random(),1);

   Test = (RotateRight(FF2(B62,C62,D62,E62,F62,G62,H62),RFACT7) -
   RotateRight(FF2(Bt62,C62,D62,E62,F62,G62,H62),RFACT7));

   printf("Equation (8) = %8.8X\n", Test);


   /* Determine if (7) Holds */

   B61 = ~(H56|Ht56);
   C61 = B62;
   Ct61 = Bt62;
   D61 = C62;
   E61 = D62;
   F61 = E62;
   G61 = F62;
   H61 = G62;

   Test = (RotateRight(FF2(B61,C61,D61,E61,F61,G61,H61),RFACT7) -
   RotateRight(FF2(B61,Ct61,D61,E61,F61,G61,H61),RFACT7));

   printf("Equation (7) = %8.8X\n", Test);

   /* Determine if (6) Holds */

   B60 = ~(H56|Ht56);
   C60 = B61;
   D60 = C61;
   Dt60 = Ct61;
   E60 = D61;
   F60 = E61;
```

```
G60 = F61;
H60 = G61;

Test = (RotateRight(FF2(B60,C60,D60,E60,F60,G60,H60),RFACT7) -
RotateRight(FF2(B60,C60,Dt60,E60,F60,G60,H60),RFACT7));

printf("Equation (6) = %8.8X\n", Test);

/* Determine if (5) Holds */

B59 = (H56|Ht56);
C59 = B60;
D59 = C60;
E59 = D60;
Et59 = Dt60;
F59 = E60;
G59 = F60;
H59 = G60;

Test = (RotateRight(FF2(B59,C59,D59,E59,F59,G59,H59),RFACT7) -
RotateRight(FF2(B59,C59,D59,Et59,F59,G59,H59),RFACT7));

printf("Equation (5) = %8.8X\n", Test);

/* Determine if (4) Holds */

B58 = ~(H56|Ht56);
C58 = B59;
D58 = C59;
E58 = D59;
F58 = E59;
Ft58 = Et59;
G58 = F59;
H58 = G59;

Test = (RotateRight(FF2(B58,C58,D58,E58,F58,G58,H58),RFACT7) -
RotateRight(FF2(B58,C58,D58,E58,Ft58,G58,H58),RFACT7));

printf("Equation (4) = %8.8X\n", Test);

/* Determine if (3) Holds */

B57 = ~(H56|Ht56);
C57 = B58;
D57 = C58;
E57 = D58;
F57 = E58;
G57 = F58;
Gt57 = Ft58;
H57 = G58;

Test = (RotateRight(FF2(B57,C57,D57,E57,F57,G57,H57),RFACT7) -
RotateRight(FF2(B57,C57,D57,E57,F57,Gt57,H57),RFACT7));

printf("Equation (3) = %8.8X\n", Test);
```

```
   /* Determine if (2) Holds */

   B56 = ~(H56|Ht56);
   C56 = B57;
   D56 = C57;
   E56 = D57;
   F56 = E57;
   G56 = F57;
   H56 = G57;
   Ht56 = Gt57;

   Test = (RotateRight(FF2(B56,C56,D56,E56,F56,G56,H56),RFACT7) -
   RotateRight(FF2(B56,C56,D56,E56,F56,G56,Ht56),RFACT7));

   printf("Equation (2) = %8.8X\n", Test);

   printf("B56: %8.8X\n", B56);
   printf("C56: %8.8X\n", C56);
   printf("D56: %8.8X\n", D56);
   printf("E56: %8.8X\n", E56);
   printf("F56: %8.8X\n", F56);
   printf("G56: %8.8X\n", G56);
   printf("H56: %8.8X\n", H56);
   printf("Ht56: %8.8X\n", Ht56);
   printf("H57: %8.8X\n", H57);
   printf("H58: %8.8X\n", H58);
   printf("H59: %8.8X\n", H59);
   printf("H60: %8.8X\n", H60);
   printf("H61: %8.8X\n", H61);
   printf("H62: %8.8X\n", H62);

   /* Derive a message that results in a collision for the last two round-
s of
    * three round HAVAL */

   /* Start forward search */
   Chain[AA][56] = random()^Rotate(random(),1);
   Chain[BB][56] = B56;
   Chain[CC][56] = C56;
   Chain[DD][56] = D56;
   Chain[EE][56] = E56;
   Chain[FF][56] = F56;
   Chain[GG][56] = G56;
   Chain[HH][56] = H56;
   Chain[HH][57] = H57;

   W[12] = InverseStep(Chain[HH][57], Chain[AA][56], Chain[BB][56],
       Chain[CC][56], Chain[DD][56], Chain[EE][56],
       Chain[FF][56], Chain[GG][56], Chain[HH][56], K1[25]);

   Chain[AA][57] = Chain[BB][56];
   Chain[BB][57] = Chain[CC][56];
   Chain[CC][57] = Chain[DD][56];
   Chain[DD][57] = Chain[EE][56];
```

```
Chain[EE][57] = Chain[FF][56];
Chain[FF][57] = Chain[GG][56];
Chain[GG][57] = Chain[HH][56];

Chain[HH][58] = H58;

W[15] = InverseStep(Chain[HH][58], Chain[AA][57], Chain[BB][57],
    Chain[CC][57], Chain[DD][57], Chain[EE][57],
    Chain[FF][57], Chain[GG][57], Chain[HH][57], K1[26]);

Chain[AA][58] = Chain[BB][57];
Chain[BB][58] = Chain[CC][57];
Chain[CC][58] = Chain[DD][57];
Chain[DD][58] = Chain[EE][57];
Chain[EE][58] = Chain[FF][57];
Chain[FF][58] = Chain[GG][57];
Chain[GG][58] = Chain[HH][57];

Chain[HH][59] = H59;

W[13] = InverseStep(Chain[HH][59], Chain[AA][58], Chain[BB][58],
    Chain[CC][58], Chain[DD][58], Chain[EE][58],
    Chain[FF][58], Chain[GG][58], Chain[HH][58], K1[27]);

Chain[AA][59] = Chain[BB][58];
Chain[BB][59] = Chain[CC][58];
Chain[CC][59] = Chain[DD][58];
Chain[DD][59] = Chain[EE][58];
Chain[EE][59] = Chain[FF][58];
Chain[FF][59] = Chain[GG][58];
Chain[GG][59] = Chain[HH][58];

Chain[HH][60] = H60;

W[2] = InverseStep(Chain[HH][60], Chain[AA][59], Chain[BB][59],
   Chain[CC][59], Chain[DD][59], Chain[EE][59],
   Chain[FF][59], Chain[GG][59], Chain[HH][59], K1[28]);

Chain[AA][60] = Chain[BB][59];
Chain[BB][60] = Chain[CC][59];
Chain[CC][60] = Chain[DD][59];
Chain[DD][60] = Chain[EE][59];
Chain[EE][60] = Chain[FF][59];
Chain[FF][60] = Chain[GG][59];
Chain[GG][60] = Chain[HH][59];

Chain[HH][61] = H61;

W[25] = InverseStep(Chain[HH][61], Chain[AA][60], Chain[BB][60],
    Chain[CC][60], Chain[DD][60], Chain[EE][60],
    Chain[FF][60], Chain[GG][60], Chain[HH][60], K1[29]);

Chain[AA][61] = Chain[BB][60];
Chain[BB][61] = Chain[CC][60];
Chain[CC][61] = Chain[DD][60];
```

```
   Chain[DD][61] = Chain[EE][60];
   Chain[EE][61] = Chain[FF][60];
   Chain[FF][61] = Chain[GG][60];
   Chain[GG][61] = Chain[HH][60];

   Chain[HH][62] = H62;

   W[31] = InverseStep(Chain[HH][62], Chain[AA][61], Chain[BB][61],
       Chain[CC][61], Chain[DD][61], Chain[EE][61],
       Chain[FF][61], Chain[GG][61], Chain[HH][61], K1[30]);

   Chain[AA][62] = Chain[BB][61];
   Chain[BB][62] = Chain[CC][61];
   Chain[CC][62] = Chain[DD][61];
   Chain[DD][62] = Chain[EE][61];
   Chain[EE][62] = Chain[FF][61];
   Chain[FF][62] = Chain[GG][61];
   Chain[GG][62] = Chain[HH][61];

   Chain[HH][63] = random()^Rotate(random(),1);

   W[27] = InverseStep(Chain[HH][63], Chain[AA][62], Chain[BB][62],
       Chain[CC][62], Chain[DD][62], Chain[EE][62],
       Chain[FF][62], Chain[GG][62], Chain[HH][62], K1[31]);

   /* Find W[19]  and Wt[19] */
   Chain[AA][55] = random()^Rotate(random(),1);
   Chain[BB][55] = Chain[AA][56];
   Chain[CC][55] = Chain[BB][56];
   Chain[DD][55] = Chain[CC][56];
   Chain[EE][55] = Chain[DD][56];
   Chain[FF][55] = Chain[EE][56];
   Chain[GG][55] = Chain[FF][56];
   Chain[HH][55] = Chain[GG][56];

   W[19] = InverseStep(Chain[HH][56], Chain[AA][55], Chain[BB][55],
       Chain[CC][55], Chain[DD][55], Chain[EE][55],
       Chain[FF][55], Chain[GG][55], Chain[HH][55], K1[24]);

   Wt[19] = InverseStep(Ht56, Chain[AA][55], Chain[BB][55],
Chain[CC][55], Chain[DD][55], Chain[EE][55],
Chain[FF][55], Chain[GG][55], Chain[HH][55], K1[24]);

   for(i=0; i<32; i++) {
      if(i!=19) {
 Wt[i] = W[i];
      }
   }

   printf("Wt[19]: %8.8X\n", Wt[19]);

   printf("Equation (9) = %8.8X\n", (RotateRight(H56,RFACT11) + W[19]) - (Ro-
tateRight(Ht56,RFACT11) + Wt[19]));

   /* Test if message words derived allows a collision to be established */
```

```
   A1 = Chain[AA][55];
   B1 = Chain[BB][55];
   C1 = Chain[CC][55];
   D1 = Chain[DD][55];
   E1 = Chain[EE][55];
   F1 = Chain[FF][55];
   G1 = Chain[GG][55];
   H1 = Chain[HH][55];

   for(i=0; i<8; i++) {
      printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", 55+i, A1, B1,

      TempInt = RotateRight(FF2(B1,C1,D1,E1,F1,G1,H1), RFACT7) +
RotateRight(A1, RFACT11) + W[Ord2[i+24]] + K1[i+24];

      A1 = B1;
      B1 = C1;
      C1 = D1;
      D1 = E1;
      E1 = F1;
      F1 = G1;
      G1 = H1;
      H1 = TempInt;
   }

   for(i=0; i<1; i++) {
      printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", 62+i, A1, B1,

      TempInt = RotateRight(FF3(B1,C1,D1,E1,F1,G1,H1), RFACT7) +
RotateRight(A1, RFACT11) + W[Ord3[i]] + K2[i];

      A1 = B1;
      B1 = C1;
      C1 = D1;
      D1 = E1;
      E1 = F1;
      F1 = G1;
      G1 = H1;
      H1 = TempInt;
   }

   printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n\n", 62+i, A1, B1,

   /* Try for Wt19 */
   A1 = Chain[AA][55];
   B1 = Chain[BB][55];
   C1 = Chain[CC][55];
   D1 = Chain[DD][55];
   E1 = Chain[EE][55];
   F1 = Chain[FF][55];
   G1 = Chain[GG][55];
   H1 = Chain[HH][55];

   for(i=0; i<8; i++) {
```

```
      printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", 55+i, A1, B1,

      TempInt = RotateRight(FF2(B1,C1,D1,E1,F1,G1,H1), RFACT7) +
RotateRight(A1, RFACT11) + Wt[Ord2[i+24]] + K1[i+24];

      A1 = B1;
      B1 = C1;
      C1 = D1;
      D1 = E1;
      E1 = F1;
      F1 = G1;
      G1 = H1;
      H1 = TempInt;
   }

   for(i=0; i<1; i++) {
      printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", 62+i, A1, B1,

      TempInt = RotateRight(FF3(B1,C1,D1,E1,F1,G1,H1), RFACT7) +
RotateRight(A1, RFACT11) + Wt[Ord3[i]] + K2[i];

      A1 = B1;
      B1 = C1;
      C1 = D1;
      D1 = E1;
      E1 = F1;
      F1 = G1;
      G1 = H1;
      H1 = TempInt;
   }

   printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n\n", 62+i, A1, B1,

   /* Target IVs */

   Chain[HH][31] = 0x243F6A88L;
   Chain[GG][31] = 0x85A308D3L;
   Chain[FF][31] = 0x13198A2EL;
   Chain[EE][31] = 0x03707344L;
   Chain[DD][31] = 0xA4093822L;
   Chain[CC][31] = 0x299F31D0L;
   Chain[BB][31] = 0x082EFA98L;
   Chain[AA][31] = 0xEC4E6C89L;

   for(i=0; i<8; i++) {
      Chain[AA][32+i] = Chain[BB][31+i];
      Chain[BB][32+i] = Chain[CC][31+i];
      Chain[CC][32+i] = Chain[DD][31+i];
      Chain[DD][32+i] = Chain[EE][31+i];
      Chain[EE][32+i] = Chain[FF][31+i];
      Chain[FF][32+i] = Chain[GG][31+i];
      Chain[GG][32+i] = Chain[HH][31+i];

   }
```

```
   /* Derive all words except last 8 */

   for(i=0; i<16; i++) {
      Chain[AA][54-i] = random()^Rotate(random(),1);
      Chain[BB][54-i] = Chain[AA][55-i];
      Chain[CC][54-i] = Chain[BB][55-i];
      Chain[DD][54-i] = Chain[CC][55-i];
      Chain[EE][54-i] = Chain[DD][55-i];
      Chain[FF][54-i] = Chain[EE][55-i];
      Chain[GG][54-i] = Chain[FF][55-i];
      Chain[HH][54-i] = Chain[GG][55-i];

      W[Ord2[23-i]] = InverseStep(Chain[HH][55-i], Chain[AA][54-i],
  Chain[BB][54-i], Chain[CC][54-i],
  Chain[DD][54-i], Chain[EE][54-i],
  Chain[FF][54-i], Chain[GG][54-i],
  Chain[HH][54-i], K1[23-i]);
   }

   for(i=0; i<8; i++) {
      Chain[BB][38-i] = Chain[AA][39-i];
      Chain[CC][38-i] = Chain[BB][39-i];
      Chain[DD][38-i] = Chain[CC][39-i];
      Chain[EE][38-i] = Chain[DD][39-i];
      Chain[FF][38-i] = Chain[EE][39-i];
      Chain[GG][38-i] = Chain[FF][39-i];
      Chain[HH][38-i] = Chain[GG][39-i];

      W[Ord2[7-i]] = InverseStep(Chain[HH][39-i], Chain[AA][38-i],
  Chain[BB][38-i], Chain[CC][38-i],
  Chain[DD][38-i], Chain[EE][38-i],
  Chain[FF][38-i], Chain[GG][38-i],
  Chain[HH][38-i], K1[7-i]);
   }

   for(i=0; i<32; i++) {
      if(i!=19) {
 Wt[i] = W[i];
      }
      printf("%i) %8.8X %8.8X\n", i, W[i], Wt[i]);
   }

   /* With IV target reached and message word derived, proceed to verify
    * collision for last two rounds of three round HAVAL */

   A1 = Chain[AA][31];
   B1 = Chain[BB][31];
   C1 = Chain[CC][31];
   D1 = Chain[DD][31];
   E1 = Chain[EE][31];
   F1 = Chain[FF][31];
   G1 = Chain[GG][31];
   H1 = Chain[HH][31];

   for(i=0; i<32; i++) {
```

```
      printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", i+1, A1, B1,

      TempInt = RotateRight(FF2(B1,C1,D1,E1,F1,G1,H1), RFACT7) +
RotateRight(A1, RFACT11) + W[Ord2[i]] + K1[i];

      A1 = B1;
      B1 = C1;
      C1 = D1;
      D1 = E1;
      E1 = F1;
      F1 = G1;
      G1 = H1;
      H1 = TempInt;
   }

   for(i=0; i<32; i++) {
      printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", i+1, A1, B1,

      TempInt = RotateRight(FF3(B1,C1,D1,E1,F1,G1,H1), RFACT7) +
RotateRight(A1, RFACT11) + W[Ord3[i]] + K2[i];

      A1 = B1;
      B1 = C1;
      C1 = D1;
      D1 = E1;
      E1 = F1;
      F1 = G1;
      G1 = H1;
      H1 = TempInt;
   }

   printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n\n", i+1, A1, B1, C

   /* With IV target reached and message word derived, proceed to verify
    * collision for last two rounds of three round HAVAL */

   A1 = Chain[AA][31];
   B1 = Chain[BB][31];
   C1 = Chain[CC][31];
   D1 = Chain[DD][31];
   E1 = Chain[EE][31];
   F1 = Chain[FF][31];
   G1 = Chain[GG][31];
   H1 = Chain[HH][31];

   for(i=0; i<32; i++) {
      printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", i+1, A1, B1,

      TempInt = RotateRight(FF2(B1,C1,D1,E1,F1,G1,H1), RFACT7) +
RotateRight(A1, RFACT11) + Wt[Ord2[i]] + K1[i];

      A1 = B1;
      B1 = C1;
      C1 = D1;
      D1 = E1;
```

```
      E1 = F1;
      F1 = G1;
      G1 = H1;
      H1 = TempInt;
   }

   for(i=0; i<32; i++) {
      printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", i+1, A1, B1,

      TempInt = RotateRight(FF3(B1,C1,D1,E1,F1,G1,H1), RFACT7) +
RotateRight(A1, RFACT11) + Wt[Ord3[i]] + K2[i];

      A1 = B1;
      B1 = C1;
      C1 = D1;
      D1 = E1;
      E1 = F1;
      F1 = G1;
      G1 = H1;
      H1 = TempInt;
   }

   printf("%d) %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X %8.8X\n", i+1, A1, B1, C1,

   /* Write message words to file */

   fp1 = fopen("data1.dat", "w");
   fp2 = fopen("data2.dat", "w");

   if(fp1 == NULL || fp2 == NULL)
     {
printf("Error Opening File\n");
exit(1);
     }


   for(i=0; i<32; i++) {

      for(j=0; j<4; j++) {
 TempChar = (char) (W[i] >> ((j)*8) & 0x000000ff);
 fwrite(&TempChar, sizeof(char), 1, fp1);

 TempChar = (char) (Wt[i] >> ((j)*8) & 0x000000ff);
 fwrite(&TempChar, sizeof(char), 1, fp2);
      }
   }

   fclose(fp1);
   fclose(fp2);

   return(0);
}

unsigned int InverseStep(unsigned int H2, unsigned int A1, unsigned int B1,
 unsigned int C1, unsigned int D1, unsigned int E1,
```

```
 unsigned int F1, unsigned int G1, unsigned int H1,
 unsigned int K)
{
   unsigned int W;

   W = H2 - RotateRight(FF2(B1, C1, D1, E1, F1, G1, H1), RFACT7) -
     RotateRight(A1, RFACT11) - K;

   return(W);
}

unsigned int Rotate(unsigned int X, unsigned int s)
{
   unsigned int temp;

   temp = X;
   X = (X << s) | (temp >> (32-s));
   return(X);
}

unsigned int RotateRight(unsigned int X, unsigned int s)
{
   unsigned int temp;

   temp = X;
   X = (X >> s) | (temp << (32-s));
   return(X);
}

unsigned int FF2(unsigned int B, unsigned int C, unsigned int D,
       unsigned int E, unsigned int F, unsigned int G,
       unsigned int H)
{
   unsigned int x0, x1, x2, x3, x4, x5, x6;

   /* Permuteer die insette tot die funksies soos gespesifiseer */
   x0 = B;
   x1 = E;
   x2 = C;
   x3 = H;
   x4 = G;
   x5 = F;
   x6 = D;

   return((x1&x2&x3) ^ (x2&x4&x5) ^ (x1&x2) ^ (x1&x4) ^ (x2&x6) ^ (x3&x5) ^
  (x4&x5) ^ (x0&x2) ^ (x0));
}

unsigned int FF3(unsigned int B, unsigned int C, unsigned int D,
       unsigned int E, unsigned int F, unsigned int G,
       unsigned int H)
{
   unsigned int x0, x1, x2, x3, x4, x5, x6;

   /* Permuteer die insette tot die funksies soos gespesifiseer */
```