-----MD4-----

Network Working Group R. Rivest

Request for Comments: 1186 MIT Laboratory for Computer Science

October 1990

The MD4 Message Digest Algorithm

Status of this Memo

This RFC is the specification of the MD4 Digest Algorithm. If you are going to implement MD4, it is suggested you do it this way. This memo is for informational use and does not constitute a standard. Distribution of this memo is unlimited.

Table of Contents

1. Abstract

This note describes the MD4 message digest algorithm. The algorithm takes as input an input message of arbitrary length and produces as output a 128-bit "fingerprint" or "message digest" of the input. It is conjectured that it is computationally infeasible to produce two messages having the same message digest, or to produce any message having a given prespecified target message digest. The MD4 algorithm is thus ideal for digital signature applications, where a large file must be "compressed" in a secure manner before being signed with the RSA public-key cryptosystem.

The MD4 algorithm is designed to be quite fast on 32-bit machines. On a SUN Sparc station, MD4 runs at 1,450,000 bytes/second. On a DEC MicroVax II, MD4 runs at approximately 70,000 bytes/second. On a 20MHz 80286, MD4 runs at approximately 32,000 bytes/second. In addition, the MD4 algorithm does not require any large substitution tables; the algorithm can be coded quite compactly.

The MD4 algorithm is being placed in the public domain for review and possible adoption as a standard.

(Note: The document supersedes an earlier draft. The algorithm described here is a slight modification of the one described in the draft.)

2. Terminology and Notation

In this note a "word" is a 32-bit quantity and a byte is an 8-bit quantity. A sequence of bits can be interpreted in a natural manner as a sequence of bytes, where each consecutive group of 8 bits is interpreted as a byte with the high-order (most significant) bit of each byte listed first. Similarly, a sequence of bytes can be interpreted as a sequence of 32-bit words, where each consecutive group of 4 bytes is interpreted as a word with the low-order (least significant) byte given first.

Let x_i denote "x sub i". If the subscript is an expression, we surround it in braces, as in x_{i+1}. Similarly, we use ^ for superscripts (exponentiation), so that x^i denotes x to the i-th power.

Let the symbol "+" denote addition of words (i.e., modulo- 2^32 addition). Let X <<< s denote the 32-bit value obtained by circularly shifting (rotating) X left by s bit positions. Let not(X) denote the bit-wise complement of X, and let X v Y denote the bit-wise OR of X and Y. Let X xor Y denote the bit-wise XOR of X and Y, and let XY denote the bit-wise AND of X and Y.

3. MD4 Algorithm Description

We begin by supposing that we have a b-bit message as input, and that we wish to find its message digest. Here b is an arbitrary nonnegative integer; b may be zero, it need not be a multiple of 8, and it may be arbitrarily large. We imagine the bits of the message written down as follows:

m_0 m_1 ... m_{b-1} .

The following five steps are performed to compute the message digest of the message.

Step 1. Append padding bits

The message is "padded" (extended) so that its length (in bits) is congruent to 448, modulo 512. That is, the message is extended so that it is just 64 bits shy of being a multiple of 512 bits long. Padding is always performed, even if the length of the message is already congruent to 448, modulo 512 (in which case 512 bits of padding are added).

Padding is performed as follows: a single "1" bit is appended to the message, and then enough zero bits are appended so that the length in bits of the padded message becomes congruent to 448, modulo 512.

Step 2. Append length

A 64-bit representation of b (the length of the message before the padding bits were added) is appended to the result of the previous step. In the unlikely event that b is greater than 2^64, then only the low-order 64 bits of b are used. (These bits are appended as two 32-bit words and appended low-order word first in accordance with the previous conventions.)

At this point the resulting message (after padding with bits and with b) has a length that is an exact multiple of 512 bits. Equivalently, this message has a length that is an exact multiple of 16 (32-bit) words. Let M[0 ... N-1] denote the words of the resulting message, where N is a multiple of 16.

Step 3. Initialize MD buffer

A 4-word buffer (A,B,C,D) is used to compute the message digest. Here each of A,B,C,D are 32-bit registers. These registers are initialized to the following values in hexadecimal, low-order bytes first):

word A: 01 23 45 67

word B: 89 ab cd ef

word C: fe dc ba 98

word D: 76 54 32 10

Step 4. Process message in 16-word blocks

We first define three auxiliary functions that each take as input three 32-bit words and produce as output one 32-bit word.

f(X,Y,Z) = XY v not(X)Z

g(X,Y,Z) = XY v XZ v YZ

h(X,Y,Z) = X xor Y xor Z

In each bit position f acts as a conditional: if x then y else z. (The function f could have been defined using + instead of v since XY and not(X)Z will never have 1's in the same bit position.) In each bit position g acts as a majority function: if at least two of x, y, z are on, then g has a one in that bit position, else g has a zero. It is interesting to note that if

the bits of X, Y, and Z are independent and unbiased, the each bit of f(X,Y,Z) will be independent and unbiased, and similarly each bit of g(X,Y,Z) will be independent and unbiased. The function h is the bit-wise "xor" or "parity" function; it has properties similar to those of f and g.

Do the following:

For i = 0 to N/16-1 do /* process each 16-word block */

For j = 0 to 15 do: /* copy block i into X */

Set X[j] to M[i*16+j].

end /* of loop on j */

Save A as AA, B as BB, C as CC, and D as DD.

[Round 1]

Let [A B C D i s] denote the operation

A = (A + f(B,C,D) + X[i]) <<< s .

Do the following 16 operations:

[A B C D 0 3]

[D A B C 1 7]

[C D A B 2 11]

[B C D A 3 19]

[A B C D 4 3]

[D A B C 5 7]

[C D A B 6 11]

[B C D A 7 19]

[A B C D 8 3]

[D A B C 9 7]

[C D A B 10 11]

[B C D A 11 19]

[A B C D 12 3]

[D A B C 13 7]

[C D A B 14 11]

[B C D A 15 19]

[Round 2]

Let [A B C D i s] denote the operation

A = (A + g(B,C,D) + X[i] + 5A827999) <<< s .

(The value 5A..99 is a hexadecimal 32-bit constant, written with the high-order digit first. This constant represents the square root of 2. The octal value of this constant is 013240474631. See Knuth, The Art of

Programming, Volume 2 (Seminumerical
Algorithms), Second Edition (1981),
Addison-Wesley. Table 2, page 660.)
Do the following 16 operations:
[A B C D 0 3]

[D A B C 4 5]
[C D A B 8 9]
[B C D A 12 13]
[A B C D 1 3]
[D A B C 5 5]
[C D A B 9 9]
[B C D A 13 13]
[A B C D 2 3]
[D A B C 6 5]
[C D A B 10 9]
[B C D A 14 13]
[A B C D 3 3]
[D A B C 7 5]
[C D A B 11 9]
[B C D A 15 13]
[Round 3]
Let [A B C D i s] denote the operation
A = (A + h(B,C,D) + X[i] + 6ED9EBA1) <<< s .
(The value 6E..A1 is a hexadecimal 32-bit
constant, written with the high-order digit
first. This constant represents the square
root of 3. The octal value of this constant
is 015666365641. See Knuth, The Art of
Programming, Volume 2 (Seminumerical
Algorithms), Second Edition (1981),
Addison-Wesley. Table 2, page 660.)
Do the following 16 operations:
[A B C D 0 3]
[D A B C 8 9]
[C D A B 4 11]
[B C D A 12 15]
[A B C D 2 3]
[D A B C 10 9]

[C D A B 6 11]

[B C D A 14 15]

[A B C D 1 3]

[D A B C 9 9]

[C D A B 5 11]

[B C D A 13 15]

[A B C D 3 3]

[D A B C 11 9]

[C D A B 7 11]

[B C D A 15 15]

Then perform the following additions:

A = A + AA

B = B + BB

C = C + CC

D = D + DD

(That is, each of the four registers is incremented by
the value it had before this block was started.)

end /* of loop on i */

Step 5. Output

The message digest produced as output is A,B,C,D. That is, we
begin with the low-order byte of A, and end with the high-order
byte of D.

This completes the description of MD4. A reference
implementation in C is given in the Appendix.

4. Extensions

If more than 128 bits of output are required, then the following
procedure is recommended to obtain a 256-bit output. (There is no
provision made for obtaining more than 256 bits.)

Two copies of MD4 are run in parallel over the input. The first copy
is standard as described above. The second copy is modified as
follows.

The initial state of the second copy is:

word A: 00 11 22 33

word B: 44 55 66 77

word C: 88 99 aa bb

word D: cc dd ee ff

The magic constants in rounds 2 and 3 for the second copy of MD4 are
changed from sqrt(2) and sqrt(3) to cuberoot(2) and cuberoot(3):

Octal Hex

Round 2 constant 012050505746 50a28be6

Round 3 constant 013423350444 5c4dd124

Finally, after every 16-word block is processed (including the last block), the values of the A registers in the two copies are exchanged.

The final message digest is obtaining by appending the result of the second copy of MD4 to the end of the result of the first copy of MD4.

5. Summary

The MD4 message digest algorithm is simple to implement, and provides a "fingerprint" or message digest of a message of arbitrary length. It is conjectured that the difficulty of coming up with two messages having the same message digest is on the order of 2^64 operations, and that the difficulty of coming up with any message having a given message digest is on the order of 2^128 operations. The MD4 algorithm has been carefully scrutinized for weaknesses. It is, however, a relatively new algorithm and further security analysis is of course justified, as is the case with any new proposal of this sort. The level of security provided by MD4 should be sufficient for implementing very high security hybrid digital signature schemes based on MD4 and the RSA public-key cryptosystem.

6. Acknowledgements

I'd like to thank Don Coppersmith, Burt Kaliski, Ralph Merkle, and Noam Nisan for numerous helpful comments and suggestions.

APPENDIX - Reference Implementation

This appendix contains the following files:

md4.h -- header file for using MD4 implementation

md4.c -- the source code for MD4 routines

md4driver.c -- a sample "user" routine

session -- sample results of running md4driver

```
/*
** ********************************************************************
** md4.h -- Header file for implementation of **
** MD4 Message Digest Algorithm **
** Updated: 2/13/90 by Ronald L. Rivest **
** (C) 1990 RSA Data Security, Inc. **
** ********************************************************************
*/
```

```
/* MDstruct is the data structure for a message digest computation.
*/
typedef struct {
unsigned int buffer[4]; /* Holds 4-word result of MD computation */
unsigned char count[8]; /* Number of bits processed so far */
unsigned int done; /* Nonzero means MD computation finished */
} MDstruct, *MDptr;
/* MDbegin(MD)
```

```
** Input: MD -- an MDptr
** Initialize the MDstruct prepatory to doing a message digest
** computation.
*/
extern void MDbegin();
/* MDupdate(MD,X,count)
** Input: MD -- an MDptr
** X -- a pointer to an array of unsigned characters.
** count -- the number of bits of X to use (an unsigned int).
** Updates MD using the first "count" bits of X.
** The array pointed to by X is not modified.
** If count is not a multiple of 8, MDupdate uses high bits of
** last byte.
** This is the basic input routine for a user.
** The routine terminates the MD computation when count < 512, so
** every MD computation should end with one call to MDupdate with a
** count less than 512. Zero is OK for a count.
*/
extern void MDupdate();
/* MDprint(MD)
** Input: MD -- an MDptr
** Prints message digest buffer MD as 32 hexadecimal digits.
** Order is from low-order byte of buffer[0] to high-order byte
** of buffer[3].
** Each byte is printed with high-order hexadecimal digit first.
*/
extern void MDprint();
/*
** End of md4.h
**********(cut)**********/
```

```
/*
** ********************************************************************
** md4.c -- Implementation of MD4 Message Digest Algorithm **
** Updated: 2/16/90 by Ronald L. Rivest **
** (C) 1990 RSA Data Security, Inc. **
** ********************************************************************
*/
/*
** To use MD4:
** -- Include md4.h in your program
** -- Declare an MDstruct MD to hold the state of the digest
** computation.
** -- Initialize MD using MDbegin(&MD)
```

```
** -- For each full block (64 bytes) X you wish to process, call
** MDupdate(&MD,X,512)
** (512 is the number of bits in a full block.)
** -- For the last block (less than 64 bytes) you wish to process,
** MDupdate(&MD,X,n)
** where n is the number of bits in the partial block. A partial
** block terminates the computation, so every MD computation
** should terminate by processing a partial block, even if it
** has n = 0.
** -- The message digest is available in MD.buffer[0] ...
** MD.buffer[3]. (Least-significant byte of each word
** should be output first.)
** -- You can print out the digest using MDprint(&MD)
*/
/* Implementation notes:
** This implementation assumes that ints are 32-bit quantities.
** If the machine stores the least-significant byte of an int in the
** least-addressed byte (e.g., VAX and 8086), then LOWBYTEFIRST
** should be set to TRUE. Otherwise (e.g., SUNS), LOWBYTEFIRST
** should be set to FALSE. Note that on machines with LOWBYTEFIRST
** FALSE the routine MDupdate modifies has a side-effect on its input
** array (the order of bytes in each word are reversed). If this is
** undesired a call to MDreverse(X) can reverse the bytes of X back
** into order after each call to MDupdate.
*/
```

```
#define TRUE 1
#define FALSE 0
#define LOWBYTEFIRST FALSE
/* Compile-time includes
*/
#include <stdio.h>
#include "md4.h"
/* Compile-time declarations of MD4 "magic constants".
*/
#define I0 0x67452301 /* Initial values for MD buffer */
#define I1 0xefcdab89
#define I2 0x98badcfe
#define I3 0x10325476
#define C2 013240474631 /* round 2 constant = sqrt(2) in octal */
#define C3 015666365641 /* round 3 constant = sqrt(3) in octal */
/* C2 and C3 are from Knuth, The Art of Programming, Volume 2
** (Seminumerical Algorithms), Second Edition (1981), Addison-Wesley.
** Table 2, page 660.
*/
```



```
#define fs1 3 /* round 1 shift amounts */
#define fs2 7
#define fs3 11
#define fs4 19
#define gs1 3 /* round 2 shift amounts */
#define gs2 5
#define gs3 9
#define gs4 13
#define hs1 3 /* round 3 shift amounts */
#define hs2 9
#define hs3 11
#define hs4 15
/* Compile-time macro declarations for MD4.
** Note: The "rot" operator uses the variable "tmp".
** It assumes tmp is declared as unsigned int, so that the >>
** operator will shift in zeros rather than extending the sign bit.
*/
#define f(X,Y,Z) ((X&Y) | ((~X)&Z))
#define g(X,Y,Z) ((X&Y) | (X&Z) | (Y&Z))
```

```
#define h(X,Y,Z) (X^Y^Z)
#define rot(X,S) (tmp=X,(tmp<<S) | (tmp>>(32-S)))
#define ff(A,B,C,D,i,s) A = rot((A + f(B,C,D) + X[i]),s)
#define gg(A,B,C,D,i,s) A = rot((A + g(B,C,D) + X[i] + C2),s)
#define hh(A,B,C,D,i,s) A = rot((A + h(B,C,D) + X[i] + C3),s)
/* MDprint(MDp)
** Print message digest buffer MDp as 32 hexadecimal digits.
** Order is from low-order byte of buffer[0] to high-order byte of
** buffer[3].
** Each byte is printed with high-order hexadecimal digit first.
** This is a user-callable routine.
*/
void
MDprint(MDp)
MDptr MDp;
{ int i,j;
for (i=0;i<4;i++)
for (j=0;j<32;j=j+8)
printf("%02x",(MDp->buffer[i]>>j) & 0xFF);
}
/* MDbegin(MDp)
** Initialize message digest buffer MDp.
** This is a user-callable routine.
*/
void
MDbegin(MDp)
```



```
MDptr MDp;
{ int i;
MDp->buffer[0] = I0;
MDp->buffer[1] = I1;
MDp->buffer[2] = I2;
MDp->buffer[3] = I3;
for (i=0;i<8;i++) MDp->count[i] = 0;
MDp->done = 0;
}
/* MDreverse(X)
** Reverse the byte-ordering of every int in X.
** Assumes X is an array of 16 ints.
```

```
** The macro revx reverses the byte-ordering of the next word of X.
*/
#define revx { t = (*X << 16) | (*X >> 16); ¥
*X++ = ((t & 0xFF00FF00) >> 8) | ((t & 0x00FF00FF) << 8); }
MDreverse(X)
unsigned int *X;
{ register unsigned int t;
revx; revx; revx; revx; revx; revx; revx; revx;
revx; revx; revx; revx; revx; revx; revx; revx;
}
/* MDblock(MDp,X)
** Update message digest buffer MDp->buffer using 16-word data block X.
** Assumes all 16 words of X are full of data.
** Does not update MDp->count.
** This routine is not user-callable.
*/
static void
MDblock(MDp,X)
MDptr MDp;
unsigned int *X;
{
register unsigned int tmp, A, B, C, D;
#if LOWBYTEFIRST == FALSE
MDreverse(X);
#endif
A = MDp->buffer[0];
B = MDp->buffer[1];
C = MDp->buffer[2];
D = MDp->buffer[3];
/* Update the message digest buffer */
ff(A , B , C , D , 0 , fs1); /* Round 1 */
ff(D , A , B , C , 1 , fs2);
ff(C , D , A , B , 2 , fs3);
ff(B , C , D , A , 3 , fs4);
```



```
ff(A , B , C , D , 4 , fs1);
ff(D , A , B , C , 5 , fs2);
ff(C , D , A , B , 6 , fs3);
ff(B , C , D , A , 7 , fs4);
```

```
ff(A , B , C , D , 8 , fs1);
ff(D , A , B , C , 9 , fs2);
ff(C , D , A , B , 10 , fs3);
ff(B , C , D , A , 11 , fs4);
ff(A , B , C , D , 12 , fs1);
ff(D , A , B , C , 13 , fs2);
ff(C , D , A , B , 14 , fs3);
ff(B , C , D , A , 15 , fs4);
gg(A , B , C , D , 0 , gs1); /* Round 2 */
gg(D , A , B , C , 4 , gs2);
gg(C , D , A , B , 8 , gs3);
gg(B , C , D , A , 12 , gs4);
gg(A , B , C , D , 1 , gs1);
gg(D , A , B , C , 5 , gs2);
gg(C , D , A , B , 9 , gs3);
gg(B , C , D , A , 13 , gs4);
gg(A , B , C , D , 2 , gs1);
gg(D , A , B , C , 6 , gs2);
gg(C , D , A , B , 10 , gs3);
gg(B , C , D , A , 14 , gs4);
gg(A , B , C , D , 3 , gs1);
gg(D , A , B , C , 7 , gs2);
gg(C , D , A , B , 11 , gs3);
gg(B , C , D , A , 15 , gs4);
hh(A , B , C , D , 0 , hs1); /* Round 3 */
hh(D , A , B , C , 8 , hs2);
hh(C , D , A , B , 4 , hs3);
hh(B , C , D , A , 12 , hs4);
hh(A , B , C , D , 2 , hs1);
hh(D , A , B , C , 10 , hs2);
hh(C , D , A , B , 6 , hs3);
hh(B , C , D , A , 14 , hs4);
hh(A , B , C , D , 1 , hs1);
hh(D , A , B , C , 9 , hs2);
hh(C , D , A , B , 5 , hs3);
hh(B , C , D , A , 13 , hs4);
hh(A , B , C , D , 3 , hs1);
hh(D , A , B , C , 11 , hs2);
hh(C , D , A , B , 7 , hs3);
hh(B , C , D , A , 15 , hs4);
```

```
MDp->buffer[0] += A;
MDp->buffer[1] += B;
MDp->buffer[2] += C;
MDp->buffer[3] += D;
```

```
}
/* MDupdate(MDp,X,count)
** Input: MDp -- an MDptr
** X -- a pointer to an array of unsigned characters.
** count -- the number of bits of X to use.
** (if not a multiple of 8, uses high bits of last byte.)
** Update MDp using the number of bits of X given by count.
** This is the basic input routine for an MD4 user.
** The routine completes the MD computation when count < 512, so
** every MD computation should end with one call to MDupdate with a
** count less than 512. A call with count 0 will be ignored if the
** MD has already been terminated (done != 0), so an extra call with
** count 0 can be given as a "courtesy close" to force termination
** if desired.
*/
void
MDupdate(MDp,X,count)
MDptr MDp;
unsigned char *X;
unsigned int count;
{ unsigned int i, tmp, bit, byte, mask;
unsigned char XX[64];
unsigned char *p;
/* return with no error if this is a courtesy close with count
** zero and MDp->done is true.
*/
if (count == 0 && MDp->done) return;
/* check to see if MD is already done and report error */
if (MDp->done)
{ printf("¥nError: MDupdate MD already done."); return; }
/* Add count to MDp->count */
tmp = count;
p = MDp->count;
while (tmp)
```

```
{ tmp += *p;
*p++ = tmp;
tmp = tmp >> 8;
}
/* Process data */
if (count == 512)
{ /* Full block of data to handle */
MDblock(MDp,(unsigned int *)X);
}
else if (count > 512) /* Check for count too large */
{ printf("¥nError: MDupdate called with illegal count value %d."
,count);
return;
```


```
}
else /* partial block -- must be last block so finish up */
{ /* Find out how many bytes and residual bits there are */
byte = count >> 3;
bit = count & 7;
/* Copy X into XX since we need to modify it */
for (i=0;i<=byte;i++) XX[i] = X[i];
for (i=byte+1;i<64;i++) XX[i] = 0;
/* Add padding '1' bit and low-order zeros in last byte */
mask = 1 << (7 - bit);
XX[byte] = (XX[byte] | mask) & ~( mask - 1);
/* If room for bit count, finish up with this block */
if (byte <= 55)
{ for (i=0;i<8;i++) XX[56+i] = MDp->count[i];
MDblock(MDp,(unsigned int *)XX);
}
else /* need to do two blocks to finish up */
{ MDblock(MDp,(unsigned int *)XX);
for (i=0;i<56;i++) XX[i] = 0;
for (i=0;i<8;i++) XX[56+i] = MDp->count[i];
MDblock(MDp,(unsigned int *)XX);
}
/* Set flag saying we're done with MD computation */
MDp->done = 1;
}
```

```
}
/*
** End of md4.c
*****************************(cut)***********************************/
/*
** ********************************************************************
** md4driver.c -- sample routines to test **
** MD4 message digest algorithm. **
** Updated: 2/16/90 by Ronald L. Rivest **
** (C) 1990 RSA Data Security, Inc. **
** ********************************************************************
*/
#include <stdio.h>
#include "md4.h"
/* MDtimetrial()
** A time trial routine, to measure the speed of MD4.
** Measures speed for 1M blocks = 64M bytes.
*/
MDtimetrial()
```



```
{ unsigned int X[16];
MDstruct MD;
int i;
double t;
for (i=0;i<16;i++) X[i] = 0x01234567 + i;
printf
("MD4 time trial. Processing 1 million 64-character blocks...¥n");
clock();
MDbegin(&MD);
for (i=0;i<1000000;i++) MDupdate(&MD,X,512);
MDupdate(&MD,X,0);
t = (double) clock(); /* in microseconds */
MDprint(&MD); printf(" is digest of 64M byte test input.¥n");
printf("Seconds to process test input: %g¥n,t/1e6);
printf("Characters processed per second: %ld.¥n,(int)(64e12/t));
}
/* MDstring(s)
** Computes the message digest for string s.
** Prints out message digest, a space, the string (in quotes) and a
```

```
** carriage return.
*/
MDstring(s)
unsigned char *s;
{ unsigned int i, len = strlen(s);
MDstruct MD;
MDbegin(&MD);
for (i=0;i+64<=len;i=i+64) MDupdate(&MD,s+i,512);
MDupdate(&MD,s+i,(len-i)*8);
MDprint(&MD);
printf(" ¥"%s¥"¥n",s);
}
/* MDfile(filename)
** Computes the message digest for a specified file.
** Prints out message digest, a space, the file name, and a
** carriage return.
*/
MDfile(filename)
char *filename;
{ FILE *f = fopen(filename,"rb");
unsigned char X[64];
MDstruct MD;
int b;
if (f == NULL)
{ printf("%s can't be opened.¥n",filename); return; }
MDbegin(&MD);
while ((b=fread(X,1,64,f))!=0) MDupdate(&MD,X,b*8);
```


```
MDupdate(&MD,X,0);
MDprint(&MD);
printf(" %s¥n",filename);
fclose(f);
}
/* MDfilter()
** Writes the message digest of the data from stdin onto stdout,
** followed by a carriage return.
*/
MDfilter()
{ unsigned char X[64];
```

```
MDstruct MD;
int b;
MDbegin(&MD);
while ((b=fread(X,1,64,stdin))!=0) MDupdate(&MD,X,b*8);
MDupdate(&MD,X,0);
MDprint(&MD);
printf("¥n");
}
/* MDtestsuite()
** Run a standard suite of test data.
*/
MDtestsuite()
{
printf("MD4 test suite results:¥n");
MDstring("");
MDstring("a");
MDstring("abc");
MDstring("message digest");
MDstring("abcdefghijklmnopqrstuvwxyz");
MDstring
("ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789");
MDfile("foo"); /* Contents of file foo are "abc" */
}
main(argc,argv)
int argc;
char *argv[];
{ int i;
/* For each command line argument in turn:
** filename -- prints message digest and name of file
** -sstring -- prints message digest and contents of string
** -t -- prints time trial statistics for 64M bytes
** -x -- execute a standard suite of test data
** (no args) -- writes messages digest of stdin onto stdout
*/
```


```
if (argc==1) MDfilter();
else
for (i=1;i<argc;i++)
if (argv[i][0]=='-' && argv[i][1]=='s') MDstring(argv[i]+2);
```

```
else if (strcmp(argv[i],"-t")==0) MDtimetrial();
else if (strcmp(argv[i],"-x")==0) MDtestsuite();
else MDfile(argv[i]);
}
/*
** end of md4driver.c
*****************************(cut)***********************************/
-----------------------------------------------------------------
--- Sample session. Compiling and using MD4 on SUN Sparcstation ---
-----------------------------------------------------------------
>ls
total 66
-rw-rw-r-- 1 rivest 3 Feb 14 17:40 abcfile
-rwxrwxr-x 1 rivest 24576 Feb 17 12:28 md4
-rw-rw-r-- 1 rivest 9347 Feb 17 00:37 md4.c
-rw-rw-r-- 1 rivest 25150 Feb 17 12:25 md4.doc
-rw-rw-r-- 1 rivest 1844 Feb 16 21:21 md4.h
-rw-rw-r-- 1 rivest 3497 Feb 17 12:27 md4driver.c
>
>cc -o md4 -O4 md4.c md4driver.c
md4.c:
md4driver.c:
Linking:
>
>md4 -x
MD4 test suite results:
31d6cfe0d16ae931b73c59d7e0c089c0 ""
bde52cb31de33e46245e05fbdbd6fb24 "a"
a448017aaf21d8525fc10ae87aa6729d "abc"
d9130a8164549fe818874806e1c7014b "message digest"
d79e1c308aa5bbcdeea8ed63df412da9 "abcdefghijklmnopqrstuvwxyz"
043f8582f241db351ce627e153e7f0e4
"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789"
a448017aaf21d8525fc10ae87aa6729d abcfile
>
>md4 -sabc -shi
a448017aaf21d8525fc10ae87aa6729d "abc"
cfaee2512bd25eb033236f0cd054e308 "hi"
>
>md4 *
```

a448017aaf21d8525fc10ae87aa6729d abcfile

Rivest [Page 17]

RFC 1186 MD4 Message Digest Algorithm October 1990

d316f994da0e951cf9502928a1f73300 md4

379adb39eada0dfdbbdfdcd0d9def8c4 md4.c

9a3f73327c65954198b1f45a3aa12665 md4.doc

37fe165ac177b461ff78b86d10e4ff33 md4.h

7dcba2e2dc4d8f1408d08beb17dabb2a md4.o

08790161bfddc6f5788b4353875cb1c3 md4driver.c

1f84a7f690b0545d2d0480d5d3c26eea md4driver.o

>

>cat abcfile | md4

a448017aaf21d8525fc10ae87aa6729d

>

>md4 -t

MD4 time trial. Processing 1 million 64-character blocks...

6325bf77e5891c7c0d8104b64cc6e9ef is digest of 64M byte test input.

Seconds to process test input: 44.0982

Characters processed per second: 1451305.

>

>

---------- end of sample session ----------

Note: A version of this document including the C source code is

available for FTP from THEORY.LSC.MIT.EDU in the file "md4.doc".

Security Considerations

The level of security discussed in this memo by MD4 is considered to

be sufficient for implementing very high security hybrid digital

signature schemes based on MD4 and the RSA public-key cryptosystem.

Author's Address

Ronald L. Rivest

Massachusetts Institute of Technology

Laboratory for Computer Science

NE43-324

545 Technology Square

Cambridge, MA 02139-1986

Phone: (617) 253-5880

EMail: rivest@theory.lcs.mit.edu

Rivest [Page 18]

-----MD5-----

/* Copyright (C) 1991-2, RSA Data Security, Inc. Created 1991. All rights reserved.
 *
 * License to copy and use this software is granted provided that it
 * is identified as the "RSA Data Security, Inc. MD5 Message-Digest
 * Algorithm" in all material mentioning or referencing this software or this function.
 *
 * License is also granted to make and use derivative works provided that such works
 * are identified as "derived from the RSA Data Security, Inc. MD5 Message-Digest Algorithm" in all material
 * mentioning or referencing the derived work.
 *
 * RSA Data Security, Inc. makes no representations concerning either the merchantability of this software
 * or the suitability of this software for any particular purpose.
 * It is provided "as is"without express or implied warranty of any kind.
 *
 * These notices must be retained in any copies of any part of this
 * documentation and/or software.
 */

-----libcurl-----

COPYRIGHT AND PERMISSION NOTICE

Copyright (c) 1996 - 2006, Daniel Stenberg, <daniel@haxx.se>.

All rights reserved.

Permission to use, copy, modify, and distribute this software for any purpose
with or without fee is hereby granted, provided that the above copyright
notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR
IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY,
FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT OF THIRD PARTY RIGHTS. IN
NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM,
DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR
OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE
OR OTHER DEALINGS IN THE SOFTWARE.

Except as contained in this notice, the name of a copyright holder shall not

be used in advertising or otherwise to promote the sale, use or other dealings
in this Software without prior written authorization of the copyright holder.

-----FreeBSD-----

/*
 * Copyright (c) 1998 Todd C. Miller <Todd.Miller@courtesan.com>
 * All rights reserved.
 *
 * Redistribution and use in source and binary forms, with or without
 * modification, are permitted provided that the following conditions
 * are met:
 * 1. Redistributions of source code must retain the above copyright
 *   notice, this list of conditions and the following disclaimer.
 * 2. Redistributions in binary form must reproduce the above copyright
 *   notice, this list of conditions and the following disclaimer in the
 *   documentation and/or other materials provided with the distribution.
 * 3. The name of the author may not be used to endorse or promote products
 *   derived from this software without specific prior written permission.
 *
 * THIS SOFTWARE IS PROVIDED ``AS IS'' AND ANY EXPRESS OR IMPLIED WARRANTIES,
 * INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY
 * AND FITNESS FOR A PARTICULAR PURPOSE ARE DISCLAIMED. IN NO EVENT SHALL
 * THE AUTHOR BE LIABLE FOR ANY DIRECT, INDIRECT, INCIDENTAL, SPECIAL,
 * EXEMPLARY, OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO,
 * PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS;
 * OR BUSINESS INTERRUPTION) HOWEVER CAUSED AND ON ANY THEORY OF LIABILITY,
 * WHETHER IN CONTRACT, STRICT LIABILITY, OR TORT (INCLUDING NEGLIGENCE OR
 * OTHERWISE) ARISING IN ANY WAY OUT OF THE USE OF THIS SOFTWARE, EVEN IF
 * ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.
 */

-----DES-----

/* Copyright (C) 1995 Eric Young (eay@mincom.oz.au) All rights reserved.
 *
 * This file is part of an SSL implementation written by Eric Young (eay@mincom.oz.au).
 * The implementation was written so as to conform with Netscapes SSL specification.
 * This library and applications are FREE FOR COMMERCIAL AND NON-COMMERCIAL USE
 * as long as the following conditions are aheared to.

-----zlib 1.1.3-----

claim that you wrote the original software. If you use this software
in a product, an acknowledgment in the product documentation would be
appreciated but is not required.
2. Altered source versions must be plainly marked as such, and must not be
misrepresented as being the original software.
3. This notice may not be removed or altered from any source distribution.
Jean-loup Gailly Mark Adler
jloup@gzip.org madler@alumni.caltech.edu

-----JPEG Library-----

Source: http://www.ijg.org/files/README
* Copyright (C) 1991-1998, Thomas G. Lane.
* This file is part of the Independent JPEG Group's software.
* For conditions of distribution and use, see the accompanying README file.
Readme File
The Independent JPEG Group's JPEG software ==========
README for release 6b of 27-Mar-1998
==========
This distribution contains the sixth public release of the Independent JPEG Group's free JPEG software. You are welcome to redistribute this software and to use it for any purpose, subject to the conditions under LEGAL ISSUES, below.
Serious users of this software (particularly those incorporating it into larger programs) should contact IJG at jpeg-info@uunet.uu.net to be added to our electronic mailing list. Mailing list members are notified of updates and have a chance to participate in technical discussions, etc.
This software is the work of Tom Lane, Philip Gladstone, Jim Boucher, Lee Crocker, Julian Minguillon, Luis Ortiz, George Phillips, Davide Rossi, Guido Vollbeding, Ge' Weijers, and other members of the Independent JPEG Group.
IJG is not affiliated with the official ISO JPEG standards committee.
DOCUMENTATION ROADMAP
==========
This file contains the following sections:
OVERVIEW
General description of JPEG and the IJG software.
LEGAL ISSUES
Copyright, lack of warranty, terms of distribution.
REFERENCES
Where to learn more about JPEG.
ARCHIVE LOCATIONS
Where to find newer versions of this software.
RELATED SOFTWARE
Other stuff you should get.

FILE FORMAT WARS

Software *not* to get.

CSR CHANGES

Changes made by CSR

(formerly Xionics, Oak, Zoran)

TO DO

Plans for future IJG releases.

Other documentation files in the distribution are:

install.doc

How to configure and install the IJG software.

usage.doc

Usage instructions for cjpeg, djpeg, jpegtran, rdjpegcom, and wrjpegcom.Unix-style man

pages for programs (same info as usage.doc)

wizard.doc

Advanced usage instructions for JPEG wizards only.

change.log

Version-to-version change highlights.

libjpeg.doc

How to use the JPEG library in your own programs.

example.c

Sample code for calling the JPEG library.

structure.doc

Overview of the JPEG library's internal structure.

filelist.doc

Road map of IJG files.

coderules.doc

Coding style rules --- please read if you contribute code.

Please read at least the files install.doc and usage.doc. Useful information can also be found in the JPEG FAQ (Frequently Asked Questions) article. See ARCHIVE LOCATIONS below to find out where to obtain the FAQ article.

If you want to understand how the JPEG code works, we suggest reading one or more of the REFERENCES, then looking at the documentation files (in roughly the order listed) before diving into the code.

OVERVIEW

This package contains C software to implement JPEG image compression and decompression. JPEG (pronounced "jay-peg") is a standardized compression method for full-color and gray-scale images. JPEG is intended for compressing "real-world" scenes; line drawings, cartoons and other non-realistic images are not its strong suit. JPEG is lossy, mean

ing that the output image is not exactly identical to the input image. Hence you must not use JPEG if you have to have identical output bits. However, on typical photographic images, very good compression levels can be obtained with no visible change, and remarkably high compression levels are possible if you can tolerate a low-quality image.

For more details, see the references, or just experiment with various compression settings.

This software implements JPEG baseline, extended-sequential, and progressive compression processes. Provision is made for

supporting all variants of these processes, although some uncommon parameter settings aren't implemented yet. For legal reasons, we are not distributing code for the arithmetic-coding variants of JPEG; see LEGAL ISSUES. We have made no provision for supporting the hierarchical or lossless processes defined in the standard.

We provide a set of library routines for reading and writing JPEG image files, plus two sample applications "cjpeg" and "djpeg", which use the library to perform conversion between JPEG and some other popular image file formats. The library is intended to be reused in other applications.

In order to support file conversion and viewing software, we have included considerable functionality beyond the bare JPEG coding/decoding capability; for example, the color quantization modules are not strictly part of JPEG decoding, but they are essential for output to colormapped file formats or colormapped displays. These extra functions can be compiled out of the library if not required for a particular application. We have also included "jpegtran", a utility for lossless transcoding between different JPEG processes, and "rdjpgcom" and "wrjpgcom", two simple applications for inserting and extracting textual comments in JFIF files.

The emphasis in designing this software has been on achieving portability and flexibility, while also making it fast enough to be useful. In particular, the software is not intended to be read as a tutorial on JPEG. (See the REFERENCES section for introductory material.) Rather, it is intended to be reliable, portable, industrial-strength code. We do not claim to have achieved that goal in every aspect of the software, but we strive for it.

We welcome the use of this software as a component of commercial products. No royalty is required, but we do ask for an acknowledgement in product documentation, as described under LEGAL ISSUES.

LEGAL ISSUES

In plain English:

1. We don't promise that this software works. (But if you find any bugs, please let us know!)
2. You can use this software for whatever you want. You don't have to pay us.
3. You may not pretend that you wrote this software. If you use it in a program, you must acknowledge somewhere in your documentation that you've used the IJG code.

In legalese:

The authors make NO WARRANTY or representation, either express or implied, with respect to this software, its quality, accuracy, merchantability, or fitness for a particular purpose. This software is provided "AS IS", and you, its user, assume the entire risk as to its quality and accuracy.

Permission is hereby granted to use, copy, modify, and distribute this software (or portions thereof) for any purpose, without fee, subject to these conditions:

(1) If any part of the source code for this software is distributed, then this README file must be included, with this copyright and no-warranty notice unaltered; and any additions, deletions, or changes to the original files must be clearly indicated in accompanying documentation.

(2) If only executable code is distributed, then the accompanying documentation must state that "this software is based in part on the work of the Independent JPEG Group".

(3) Permission for use of this software is granted only if the user accepts full responsibility for any undesirable consequences; the authors accept NO LIABILITY for damages of any kind.

These conditions apply to any software derived from or based on the IJG code, not just to the unmodified library. If you use our

work, you ought to acknowledge us.

Permission is NOT granted for the use of any IJG author's name or company name in advertising or publicity relating to this software or products derived from it. This software may be referred to only as "the Independent JPEG Group's software".

We specifically permit and encourage the use of this software as the basis of commercial products, provided that all warranty or liability claims are assumed by the product vendor.

ansi2knr.c is included in this distribution by permission of L. Peter Deutsch, sole proprietor of its copyright holder, Aladdin Enterprises of Menlo Park, CA. ansi2knr.c is NOT covered by the above copyright and conditions, but instead by the usual distribution terms of the Free Software Foundation; principally, that you must include source code if you redistribute it. (See the file ansi2knr.c for full details.) However, since ansi2knr.c is not needed as part of any program generated from the IJG code, this does not limit you more than the foregoing paragraphs do.

The Unix configuration script "configure" was produced with GNU Autoconf. It is copyright by the Free Software Foundation but is freely distributable. The same holds for its supporting scripts (config.guess, config.sub, ltconfig, ltmain.sh). Another support script, install-sh, is copyright by M.I.T. but is also freely distributable.

It appears that the arithmetic coding option of the JPEG spec is covered by patents owned by IBM, AT&T, and Mitsubishi. Hence arithmetic coding cannot legally be used without obtaining one or more licenses.

For this reason, support for arithmetic coding has been removed from the free JPEG software.

(Since arithmetic coding provides only a marginal gain over the unpatented Huffman mode, it is unlikely that very many implementations will support it.) So far as we are aware, there are no patent restrictions on the remaining code.

The IJG distribution formerly included code to read and write GIF files. To avoid entanglement with the Unisys LZW patent, GIF reading support has been removed altogether, and the GIF writer has been simplified to produce "uncompressed GIFs". This technique does not use the LZW algorithm; the resulting GIF files are larger than usual, but are readable by all standard GIF decoders.

We are required to state that "The Graphics Interchange Format(c) is the Copyright property of CompuServe Incorporat ed. GIF(sm) is a Service Mark property of CompuServe Incorporated."

REFERENCES

We highly recommend reading one or more of these references before trying to understand the innards of the JPEG software.

The best short technical introduction to the JPEG compression algorithm is Wallace, Gregory K. "The JPEG Still Picture Compression Standard", Communications of the ACM, April 1991 (vol. 34 no. 4), pp. 30-44.

(Adjacent articles in that issue discuss MPEG motion picture compression, applications of JPEG, and related topics.) If you don't have the CACM issue handy, a PostScript file containing a revised version of Wallace's article is available at ftp://ftp.uu.net/graphics/jpeg/wallace.ps.gz. The file (actually a preprint for an article that appeared in IEEE Trans. Consumer Electronics) omits the sample images that appeared in CACM, but it includes corrections and some added material. Note: the Wallace article is copyright ACM and IEEE, and it may not be used for commercial purposes.

A somewhat less technical, more leisurely introduction to JPEG can be found in "The Data Compression Book" by Mark Nelson and Jean-loup Gailly, published by M&T Books (New York), 2nd ed. 1996, ISBN 1-55851-434-1. This book provides good explanations and example C code for a multitude of compression methods including JPEG. It is an excellent source if you are comfortable reading C code but don't know much about data compression in general. The book's JPEG sample code is far from industrial-strength, but when you are ready to look at a full implementation, you've got one here...

The best full description of JPEG is the textbook "JPEG Still Image Data Compression Standard" by William B. Pennebaker and Joan L. Mitchell, published by Van Nostrand Rein hold, 1993, ISBN 0-442-01272-1. Price US$59.95, 638 pp. The book includes

the complete text of the ISO JPEG standards (DIS 10918-1 and draft DIS 10918-2). This is by far th
e most complete exposition of JPEG in existence, and we highly recommend it.
The JPEG standard itself is not available electronically; you must order a paper copy through ISO or ITU.
In the USA, copies of the standard may be ordered from ANSI Sales at (212) 642-4900, or from Global Engineering Documents at (800) 854-7179. (ANSI doesn't take credit card orders, but Global does.) It's not cheap: as of 1992, ANSI was charging $95 for Part 1 and $47 for Part 2, plus 7% shipping/handling. The standard is divided into two parts
, Part 1 being the actual specification, while Part 2 covers compliance testing methods. Part 1 is titled "Digital Compression and Coding of Continuous-tone Still Images, Part 1: Requirements and guidelines" and has document numbers ISO/IEC IS 10918-1, ITU-T T.81. Part 2 is titled "Digital Compression and Coding of Continuous-to
ne Still Images, Part 2: Compliance testing" and has document numbers ISO/IEC IS 10918-2, ITU-T T.83.
Some extensions to the original JPEG standard are defined in JPEG Part 3, a newer ISO standard numbered ISO/IEC IS 10918-3 and ITU-T T.84. IJG currently does not support any Part 3 extensions.
The JPEG standard does not specify all details of an interchangeable file format. For the omitted details we follow the "JFIF" conventions, revision 1.02. A copy of the JFIF spec is available from:
Literature Department
C-Cube Microsystems, Inc.
1778 McCarthy Blvd.
Milpitas, CA 95035
phone (408) 944-6300, fax (408) 944-6314
A PostScript version of this document is available by FTP at ftp://ftp.uu.net/graphics/jpeg/jfif.ps.gz. There is also a plain text version at ftp://ftp.uu.net/graphics/jpeg/jfif.txt.gz, but it is missing thefigures.
The TIFF 6.0 file format specification can be obtained by FTP from ftp://ftp.sgi.com/graphics/tiff/TIFF6.ps.gz. The JPEG incorporation scheme found in the TIFF 6.0 spec of 3-June-92 has a number of serious problems. IJG does not recommend use of the TIFF 6.0 design (TIFF Compression tag 6). Instead, we recommend the JPEG design proposed by TI
FF Technical Note #2 (Compression tag 7). Copies of this Note can be obtained from ftp.sgi.com or from ftp://ftp.uu.net/graphics/jpeg/. It is expected that the next revision of the TIFF spec will replace the 6.0 JPEG design with the Note's design. Although IJG's own code does not support TIFF/JPEG, the free libtiff library uses our library to implement TIFF/JPEG per the Note. libtiff is available from ftp://ftp.sgi.com/graphics/tiff/.
ARCHIVE LOCATIONS
==========
The "official" archive site for this software is ftp.uu.net (Internet address 192.48.96.9). The most recent released version can always be found there in directory graphics/jpeg. This particular version will be archived as ftp://ftp.uu.net/graphics/jpeg/jpeg src.v6b.tar.gz. If you don't have direct Internet access, UUNET's archives are also available via UUCP; contact help@uunet.uu.net for information on retrieving files that way.
Numerous Internet sites maintain copies of the UUNET files. However, only ftp.uu.net is guaranteed to have the latest official version.
You can also obtain this software in DOS-compatible "zip" archive format from the SimTel archives(ftp://ftp.simtel.net/pub/simtelnet/msdos/graphics/), or on CompuServe in the Graphics Support forum (GO CIS:GRAPHSUP), library 12 "JPEG Tools". Again, these versions may sometimes lag behind the ftp.uu.net release.
The JPEG FAQ (Frequently Asked Questions) article is a useful source of general information about JPEG. It is updated

constantly and therefore is not included in this distribution. The FAQ is posted every two weeks to Usenet newsgroups comp.graphics.misc, news.answers, and other groups. It is available on the World Wide Web at http://www.faqs .org/faqs/jpeg-faq/ and other news.answers archive sites, including the official news.
answers archive at rtfm.mit.edu: ftp://rtfm.mit.edu/pub/usenet/news.answers/jpeg-faq/.
If you don't have Web or FTP access, send e-mail to mail-server@rtfm.mit.edu with body from a wide range of other formats, thus making cjpeg/djpeg considerably more useful.
The latest version is distributed by the NetPBM group, and is available from numerous sites, notably ftp://wuarchive.wustl.edu/graphics/graphics/packages/NetPBM/.
Unfortunately PBMPLUS/NETPBM is not nearly as portable as the IJG software is; you are likely to have difficulty making it work on any non-Unix machine.
A different free JPEG implementation, written by the PVRG group at Stanford, is available from ftp://havefun.stanford.edu/pub/jpeg/. This program is designed for research and experimentation rather than production use; it is slower, harder to use, and less portable than the IJG code, but it is easier to read and modify. Also, the PVRG code supports lossless JPEG, which we do not. (On the other hand, it doesn't do progressive JPEG.)
FILE FORMAT WARS
Some JPEG programs produce files that are not compatible with our library. The root of the problem is that the ISO JPEG committee failed to specify a concrete file format.
Some vendors "filled in the blanks" on their own, creating proprietary formats that no one else could read. (For example, none of the early commercial JPEG implementations for the Macintosh were able to exchange compressed files.)
The file format we have adopted is called JFIF (see REFERENCES). This format has been agreed to by a number of major commercial JPEG vendors, and it has become the de facto standard. JFIF is a minimal or "low end" representation. We recommend the use of TIFF/JPEG (TIFF revision 6.0 as modified by TIFF Technical Note #2) for "high end"
send usenet/news.answers/jpeg-faq/part1
send usenet/news.answers/jpeg-faq/part2
RELATED SOFTWARE
Numerous viewing and image manipulation programs now support JPEG. (Quite a few of them use this library to do so.) The JPEG FAQ described above lists some of the more popular free and shareware viewers, and tells where to obtain them on Internet.
If you are on a Unix machine, we highly recommend Jef Poskanzer's free PBMPLUS software, which provides many useful operations on PPM-format image files. In particular, it can convert PPM images to and applications that need to record a lot of additional data about an image. TIFF/JPEG is fairly new and not yet widely supported, unfortunately
.
The upcoming JPEG Part 3 standard defines a file format called SPIFF. SPIFF is interoperable with JFIF, in the sense that most JFIF decoders should be able to read the most common variant of SPIFF. SPIFF has some technical advantages over JFIF, but its major claim to fame is simply that it is an official standard rather than an informal one.
At this point it is unclear whether SPIFF will supersede JFIF or whether JFIF will remain the de-facto standard. IJG intends to support SPIFF once the standard is frozen, but we have not decided whether it should become our default output format or not. (In any case, our decoder will remain capable of reading JFIF indefinitely.)
Various proprietary file formats incorporating JPEG compression also exist. We have little or no sympathy for the existence of these formats. Indeed, one of the original reasons for developing this free software was to help force convergence on common,

open format standards for JPEG files. Don't use a proprietary file format!

TO DO

The major thrust for v7 will probably be improvement of visual quality. The current method for scaling the quantization tables is known not to be very good at low Q values.

We also intend to investigate block boundary smoothing, "poor man's variable quantization", and other means of improving quality-vs-file-size performance without sacrificing compatibility.

In future versions, we are considering supporting some of the upcoming JPEG Part 3 extensions --- principally, variable quantization and the SPIFF file format.

Please send bug reports, offers of help, etc. to jpeg-info@uunet.uu.net.

-----ICC Profile Header-----