CMU-ITC-91-059

TRADMYBD - Technical Reference

Technical Reference Manual and Programmers' Guide

Version 1.00

January 1991

Paul G. Crumley
Information Technology Center
Carnegie-Mellon University
Pittsburgh, PA 15213
412/268-6720
ARPA: pgc@andrew.cmu.andrew

PREFACE

Acknowledgements

This adapter would not have been produced with out the encouragement and guidance of many people. I especially wish to thank the following:

Bryan Striemer for his technical guidance and motivation to strive for excellence.

Mahadev Satyanarayanan "Satya" for his help in designing the software interfaces.

To all members of the Information Technology Center at Carnegie-Mellon University for making this department such a pleasant and "creative" place.

Trademarks

The following trademarks appear in this document:

IBM, IBM PC, IBM PC/XT, IBM PC/AT, IBM RT/PC, ACIS 4.2, and AIX are trademarks of International Business Machines.

Microsoft and MS-DOS are trademarks of Microsoft Corporation.

AMD is a trademark of Advanced Micro Devices.

UNIX is a trademark of AT&T Bell Laboratories.

Export Restrictions

WARNING: The shipment of the hardware and some of the software described in this document to a country outside the United States requires a U. S. Government license.

Nomenclature

All numbers in this document will be represented in base 10 unless they are followed by an "H" for hexadecimal or a "B" for binary representations. If needed, a "D" may be used to show base 10. An example: 10 = 0AH = 1010B = 10D.

"X" in a bit position will represent a "don't care" value. These bits should be set to "0" to ensure compatibility with future versions of the TRADMYBD hardware. These bits will return "undefined" values when read.

Bits will be numbered from right to left with the right-most bit being 0. This is NOT the nomenclature used by IBM in many of its documents but this numbering scheme is used by a large segment of the computer industry. For example, bits 0,2,4 and 6 are "1" and bits 1,3,5 and 7 are "0" in the number 01010101B.

large segment of the computer industry. For example, bits 0,2,4 and 6 are "1" and bits 1,3,5 and 7 are "0" in the number 01010101B.

CONTENTS

INTRODUCTION

TRADMYBD is designed to provide a high performance hardware implementation of the Data Encryption Standard (DES). This incarnation of TRADMYBD is not designed to be a production level adapter. This adapter is to be used to test the feasibility of using DES in a networked workstation environment. Depending on the experience gained with TRADMYBD, a new adapter may be designed for use in the CMU workstation environment.

DES OVERVIEW

TRADMYBD will transform a block of data (clear text) into a modified block of data (cipher text) of the same size using a key (key) to alter the exact form of modification. The adapter can also reverse this procedure by using the same key to return the cipher text to its clear text form. This type of algorithm is often described as the pair of functions E(d,k) and D(d,k) where E(d,k) is called encryption of data "d" using key "k". Similarly, D(d,k) is called decryption of data "d" using key "k". It is worth mentioning that d = D(E(d,k),k) = E(D(d,k),k).

The particular function used to encrypt and decrypt the data is described in the document "Federal Information Processing Standards Publication 46 DATA ENCRYPTION STANDARD." These functions are usually simply called DES in most discussions of data encryption.

DES has the following properties:

- A 64 bit block of clear text is transformed into a 64 bit block of cipher text.
- A key of 56 bits is used to determine this transformation.
- A "brute-force" attack is too computationally expensive to be practical.
- Even if matching pieces of clear and cipher text are obtained, determining the key is too computationally expensive to be practical.
- Hardware implementations of DES can be very fast while software implementations on conventional computer architectures are usually slow.
- The DES algorithm is VERY standardized and transferring data between systems provides no problems.
- A special mode of operation is available that masks patterns in the data. (This mode is called Cipher Block Chaining (CBC).)

ADAPTER DESCRIPTION

TRADMYBD is designed to be very simple to understand conceptually. It should also be easy to use. The bulk of encryption and decryption function is done by a single chip. This chip is made by Advanced Micro Devices (AMD) and is called the Am9568 Data Ciphering Processor (DCP). The rest of the adapter components provide a simple interface between the CPU and the DCP. TRADMYBD uses 16 bytes of the CPU's I/O address space. This space is used to provide an 8 bit Control Register, an 8 bit Report Register and an 8 bit Data Port. These registers and port are addressed as follows:

Address (Binary)	Write	Read
1111 0000 0000 0000 YYYY 1010 ZZZZ 0000	Control	Report
1111 0000 0000 0000 YYYY 1010 ZZZZ 0001	Undefined	Undefined
1111 0000 0000 0000 YYYY 1010 ZZZZ 0010	Undefined	Undefined
1111 0000 0000 0000 YYYY 1010 ZZZZ 0011	Undefined	Undefined
1111 0000 0000 0000 YYYY 1010 ZZZZ 0100	Undefined	Undefined
1111 0000 0000 0000 YYYY 1010 ZZZZ 0101	Undefined	Undefined
1111 0000 0000 0000 YYYY 1010 ZZZZ 0110	Undefined	Undefined
1111 0000 0000 0000 YYYY 1010 ZZZZ 0111	Undefined	Undefined
1111 0000 0000 0000 YYYY 1010 ZZZZ 1XXX	Data	Data

Note: The values of YYYY and ZZZZ are set by switch SW1 on the adapter. It is recommended that YYYY be set to 0001B and ZZZZ be set to 1001B thus causing the adapter to use 16 bytes starting at the RT/PC CPU address F0001A90H. On PC class machine only the 16 LSBs are used giving an I/O address of 1A90H.

Notice that TRADMYBD can access the Data Port at any of eight different locations. This is done so that the programmer can treat the Data Port as an array of eight bytes. Such an organization might allow programs to transfer data to the adapter more efficiently than if the data must be moved one byte at a time.

The TRADMYBD Control and Report Registers are defined as follows:

Register	Access Type	Bit 7	Bit 6	Bit 5 - 3	Bit 2	Bit 1	Bit 0
Control	Write	XXXX	XXXX	XXXX	Reset	Addr1	Addr0
Report	Read	OReq	IReq	XXXX	Reset	Addr1	Addr0

AddrX These bits select the DCP register that is accessed through the Data Port.
Reset "1" will cause the card to be reset. This bit will normally be "0".
OReq "1" indicates the output buffer is full and data should be read from the adapter.
IReq "1" indicates the input buffer is empty and data should be sent to the adapter.

Data Ciphering Processor (DCP) Description

TRADMYBD is operated by sending the DCP a command via the Command Register and then using the status bits in Status Register to control the flow of data through the DCP. The Am9568 DCP is fully described in AMD publication "Data Ciphering Processors Am9518, Am9568, AmZ8068."

The DCP is accessed through 5 registers. These are the Input, Output, Command, Status, and Mode Registers. These registers are accessed by setting the TRADMYBD Control register bits as follows:

Bits			Register	
Reset	Addr1	Addr0	Write	Read
0	X	0	Input	Output
0	0	1	Command	Status
0	1	1	Mode	Undefined

These registers have the following functions:

The Input Register is used to send data and keys into the DCP. This register is simply treated as an 8 bit quantity

The Output Register is used to remove data from the DCP. This register is simply treated as an 8 bit quantity.

The Command Register is used to send commands to the DCP. The most common commands include:

Command	Code
Load Clear Encryption Key	11H
Load Clear Decryption Key	12H
Load Clear IVE Register	85H
Load Clear IVD Register	84H
Read Clear IVE Register	8DH
Read Clear IVD Register	8CH
Start Encryption	41H
Start Description	40H
Stop	E0H
Reset DCP	00H

The Status Register is used to determine the state of the DCP. The bits in the Status Register are defined as follows:

Bit	Description
7	Start-Stop 0 = Is set by the Stop command

	1 = Is set by a Start command
6	Command Pending 0 = No command is waiting to complete 1 = Command needs data to complete
5	DCP Busy 0 = The encryption processor is idle 1 = The encryption processor is busy processing data
4	Latched Parity Error 0 = All key bytes had odd parity 1 = One or more key bytes had even parity
3	Parity Error 0 = Last key byte written had odd parity (OK) 1 = Last key byte written had even parity (not OK)
2	Auxiliary Port Flag 0 = Auxiliary key port is not ready for data 1 = Auxiliary key port is ready for data
1	Slave Port Flag 0 = Output buffer is full, read 8 bytes of data 1 = Output buffer is not full, don't read any data
0	Master Port Flag 0 = Input buffer is full, don't send any data 1 = Input buffer is not full, send more data

The Mode Register is used to configure the DCP for a particular type of operation. The bits in the Mode Register are defined as follows:

Bit	Description
7, 6, 5	Reserved
4	Encrypt-Decrypt 0 = Generic Start command will decrypt 1 = Generic Start command will encrypt
3, 2	Port Configuration 00 = Dual Port, Master Encrypted 01 = Dual Port, Slave Encrypted 10 = Single Port, Master Only 11 = Reserved
1, 0	Cipher Type 00 = Electronic Code Book 01 = Cipher Feedback 10 = Cipher Block Chain 11 = Reserved

Note: This register is usually set only once, at initialization time. The value written to this register MUST set the Port Configuration (bits 3 & 2) to 10B placing the DCP in Single Port mode.

Data Ciphering Processor (DCP) Access

The TRADMYBD DES Adapter allows the CPU to:

- o RESET the DCP.
- o Set the DCP register address.
- o read selected status from the DCP without having to read the DCP status register.
- o read and write the DCP's various registers (Input, Output, Command, Status, and Mode).

TRADMYBD provides access to the DCP by allowing the address of the desired DCP register to be written to bits Addr1 and Addr0 in the Control Register. After the DCP register address is set, the DCP register is read/written through the Data Port. The address does NOT have to be set before each DCP register access.

The Report Register on the DES Adapter allows the values written to the Control Register to be retrieved and also allows two bits, the Master Port Flag (IReq) and Slave Port Flag (OReq), in the DCP Status Register to be read without requiring the DCP address to be altered. This allows greater data throughput when encrypting/decrypting data by saving the time needed to change the DCP address to read the input and output status of the DCP.

TRADMYBD can be configured to use various I/O addresses by setting switch SW1 on the adapter. TRADMYBD uses 16 bytes starting at CPU address F000YAZ0H where the values of Y and Z are set by switches 1, 2, 3, 4 and 5, 6, 7, 8 respectively. In the IBM PC/XT/AT, only the 16 Least Significant Bits of the address are used.

Switch Number	Address Bit	Normal Setting	Normal Address
1	15	ON	0
2	14	ON	0
3	13	ON	0
4	12	OFF	1
5	7	OFF	1
6	6	ON	0
7	5	ON	0
8	4	OFF	1

Note: When a switch is ON the corresponding address bit will be matched as a "0". With the normal switch settings the adapter will respond to 16 bytes starting at RT/PC CPU address F0001A90H. On PC class machine only the 16 LSBs are used giving an I/O address of 1A90H.

PROGRAMMING NOTES

TRADMYBD is designed to be easy to use. This example shows all the steps required to encrypt a block of data using Electronic Code Book (ECB) mode.

1. Initialize the card by:

    a. Write 03H (address DCP mode register) to the Control Register.

    b. Write 08H (Single Port and Electronic Code Book) to the Data Port.

2. Set the key by:

    a. Write 01H (address DCP command register) to the Control Register.

    b. Write 11H, Set the Encryption key command, to the Data Port.

    c. Poll the Data Port waiting for the DCP status register to set bit 6, the Command Pending bit, to "1".

    d. Write the eight byte key to the Data Port one byte at a time.

    e. Read the Data Port and check the DCP status register to be sure bit 4, the Latched Parity Error bit is "0". If this bit is "1" one of the key bytes did not have odd parity. (Each byte MUST have odd parity.)

3. Start Encrypting data by:

    a. Write 41H, Start Encrypting data command, to the Data Port.

    b. Write 00H (address DCP I/O register) to the Control Register.

    c. Poll the Report Register waiting for the IReq bit, bit 6, to be set to "1".

    d. Write 8 bytes of data to the Data Port.

    e. Poll the Report Register waiting for the OReq bit, bit 7, to be set to "1".

    f. Read 8 bytes of data from the adapter Data Port.

    g. If you have more data to encrypt go to step c).

    h. Write 01H (address DCP command register) to the Control Register.

    i. Write C0H, the Stop command, to the Data Port.

There are ways to pipeline the data transfer and increase the throughput of the card by up to 200%. These techniques are discussed in AMD publication "Data Ciphering Processors Am9518, Am9568, AmZ8068."

PERFORMANCE

The TRADMYBD adapter can support a variety of programming models. The performance measurements given here were obtained using the device driver code listed below running on an IBM RT/PC. The RT/PC was equipped with the original "ROMP" processor and 4 MB of RAM. IBM's Academic Operating System (ACIS 4.2) was used. Better times would be obtained by using the newer, APC processors.

The sample device driver code moves the data directly from user space to the adapter and back eliminating intermediate copy buffers and also allows the TRADMYBD adapter to be used by many processes simultaneously. This version of the code has the following performance:

A call to DES() will take the following amount of time:

$$T = N * K + C$$

where:

T is microseconds (10**-6)
N is the size of the data to be en/decrypted in bytes
K is a constant 4.13 microseconds/byte
C is a constant 470 microseconds

This equation predicts T +/- 5% for any value of N up to about 200,000 bytes. After 200,000 bytes page faults start to occur and performance is degraded. This should not be a big problem as most programs will not need to en/decrypt more than 200,000 bytes with only one function call. If more than 200,000 bytes must be en/decrypted, simply break it up into smaller pieces.

This model shows that the NO-OP case takes 470 uSecs. This time is consumed by UNIX finding the right device driver, saving machine state, copying the parameters to kernel address space, having the device driver set up the TRADMYBD DES adapter and restoring state when returning to the application program.

The per-byte time of 4.13 uSec/byte is consumed by:

- bad compiler optimizations consume about 0.350 uSec/byte. Newer compilers should be able to do a better job but if not, hand-tuned assembler code will get this back.

- raw data I/O to card about 2.100 uSec/byte. A new, more expensive design is needed to improve this. The theoretical limit for the RT/PC I/O bus, if price is no object, is about 0.600 uSec/byte.

- the rest of the wasted time, about 1.7 uSec/byte, is caused by the processor keeping the MMU in translation mode rather than switching to real mode in the kernel. This looks like the biggest bottle-neck at this time and would require some "heavy-duty" software changes to the operating system to better these times. The newer APC processors can get back most of this time.

APPENDIX A. SCHEMATICS

Schematics for the TRADMYBD adapter are availble from the author.

APPENDIX B. TEST DATA

This appendix contains test data for the DES Adapter.

ECB Test Data

These test cases are used in ECB mode to test all the S-Boxes in the DCP. (Clear, Key and Cipher data is given in Hexadecimal notation.)

Case	Clear	Data	Key	Data	Cipher	Data
1	01A1D6D0	39776742	7CA11045	4A1A6E57	690F5B0D	9A26939B
2	5CD54CA8	3DEF57DA	0131D961	9DC1376E	7A389D10	354BD271
3	0248D438	06F67172	07A1133E	4A0B2686	868EBB51	CAB4599A
4	51454B58	2DDF440A	3849674C	2602319E	7178876E	01F19B2A
5	42FD4430	59577FA2	04B915BA	43FEB5B6	AF37FB42	1F8C4095
6	059B5E08	51CF143A	0113B970	FD34F2CE	86A560F1	0EC6D85B
7	0756D8E0	774761D2	0170F175	468FB5E6	0CD3DA02	0021DC09
8	762514B8	29BF486A	43297FAD	38E373FE	EA676B2C	B7DB2B7A
9	3BDD1190	49372802	07A71370	45DA2A16	DFD64A81	5CAF1A0F
10	26955F68	35AF609A	04689104	C2FD3B2F	5C513C9C	4886C088
11	164D5E40	4F275232	37D06BB5	16CB7546	0A2AEEAE	3FF4AB77
12	6B056E18	759F5CCA	1F08260D	1AC2465E	EF1BF03E	5DFA575A
13	004BD6EF	09176062	58402364	1ABA6176	88BF0DB6	D70DEE56
14	480D3900	6EE762F2	02581616	4629B007	A1F99155	41020B56
15	437540C8	698F3CFA	49793EBC	79B3258F	6FBF1CAF	CFFD0556
16	072D43A0	77075292	4FB05E15	15AB73A7	2F22E49B	AB7CA1AC
17	02FE5577	8117F12A	49E95D6D	4CA229BF	5A6B612C	C26CCE4A
18	1D9D5C50	18F728C2	018310DC	409B26D6	5F4C038E	D12B2E41
19	30553228	6D6F295A	1C587F1C	13924FEF	63FAC0D0	34D9F793

CBC Test Data

These test cases are used in CBC mode to test the chaining data paths and IV registers in the DCP. (Clear, Key, Cipher and IV data is given in Hexadecimal notation.)

E-Key = D-Key = 0123456789ABCDEF; IVE = IVD = 0123456789ABCDEF

Encryption Test Data

Case	Clear	Data	Cipher	Data
1	4E6F7720	69732074	E5C7CDDE	872BF27C
2	68652074	696D6520	43E93400	8C389C0F
3	666F7220	616C6C20	68378849	9A7C05F6

Decryption Test Data

Case	Clear	Data	Cipher	Data
1	E5C7CDDE	872BF27C	4E6F7720	69732074
2	43E93400	8C389C0F	68652074	696D6520
3	68378849	9A7C05F6	666F7220	616C6C20

APPENDIX C. UNIX SOFTWARE INTERFACES

This appendix contains the information needed to use the TRADMYBD adapter from the UNIX program environment. This software requires the ACIS 4.2 version of Berkeley Unix and only works on the IBM RT/PC. This software requires that the DES device driver is installed in the operating system kernel and the special device "/dev/des" must exist in "/dev".

Other languages can be used if they provide a mechanism for issuing IOCTLs. For more information see the device driver documentation.

All of this software is available in machine readable form from the author.

Device Driver Code

DESIOCTL.H

This include file defines the IOCTLs that are supported by TRADMYBD. All of TRADMYBD's functions are accessed via IOCTLs.

```
/*
start of desioctl.h
*/

/*
    Written by Paul G. Crumley
    Carnegie-Mellon University
    Information Technology Center
    Pittsburgh, PA  15213

    February 1987

    Device driver for Paul's TRADMYBD DES card
*/

#define DES_IOCTL(op) ((unsigned int)_IOW(D, op, struct DES_Struct))

/*
end of desioctl.h
*/
```

DESREG.H

This include file defines what the TRADMYBD hardware looks like to software.

```
/*
    Written by Paul G. Crumley
    Carnegie-Mellon University
    Information Technology Center
    Pittsburgh, PA  15213
    pgc@andrew.itc.cmu.edu
    1986, 1987
*/


/* this is what TRADMYBD's registers look like to a C program. */

struct desdevice    /* layout of device registers */
    {
    unsigned char csr;
    unsigned char junk[7];
    union
      {
        unsigned char b[8];
      unsigned int  w[2];
        } data;
    };

/* values to stuff into the above locations. */


/* these are written to the csr to: */
/* reset or let TRADMYBD run */

#define     DESCntlReset          (0x04)
#define DESCntlRun          (0x00)

/* access various DCP registers */

#define DESCntlIO     (0x00)
#define DESCntlCommand      (0x01)
#define DESCntlStatus       (0x01)
```

```
#define DESCntlMode          (0x03)


/* these are masks to apply to values read from the csr */
/* when asking TRADMYBD to report its status. */

#define DESRptNeedsDataM     (0x80)
#define DESRptHasDataM       (0x40)
#define DESRptDCPAddrM       (0x03)
#define DESRptResetM         (0x04)


/* written to b location when csr points to DESCntlCommand */
/* to send the following commands to the DCP. */

#define DESCmdLdClrEKey      (0x11)
#define DESCmdLdClrDKey      (0x12)
#define DESCmdStartE         (0x41)
#define DESCmdStartD         (0x40)
#define DESCmdLdClEIV        (0x85)
#define DESCmdLdClDIV        (0x84)
#define DESCmdRdClEIV        (0x8D)
#define DESCmdRdClDIV        (0x8C)
#define DESCmdStop           (0xe0)
#define DESCmdReset          (0x00)


/* masks to use when reading b location when csr points to */
/* DESCntlStatus. */

#define DESStatMPortActM     (0x01)
#define DESStatSPortActM     (0x02)
#define DESStatAPortActM     (0x04)
#define DESStatParM          (0x08)
#define DESStatLParM         (0x10)
#define DESStatBusyM         (0x20)
#define DESStatCmdPendM      (0x40)
#define DESStatStartM        (0x80)


/* written to b location when csr points to DESCntlMode */
/* to set the operatine mode of the DCP. */

#define DESModeDualMESC      (0x00)
```

```
#define DESModeDualMCSE      (0x04)
#define DESModeSingle        (0x08)
#define DESModeECB           (0x00)
#define DESModeCF      (0x01)
#define DESModeCBC           (0x02)
```

DES.C

This C code is the actual device driver for TRADMYBD.

```
/*
    Written by Paul G. Crumley
    Carnegie-Mellon University
    Information Technology Center
    Pittsburgh, PA  15213
    pgc@andrew.itc.cmu.edu
    1986, 1987

    Device driver for Paul's TRADMYBD DES card.

    This device driver provides an IOCTL interface allowing
    user programs to en/decrypt data using the NBS DES
    algorithm.

    The current TRADMYBD adapter relies on the processor
    to move all the data through the card.  (no DMA...yuck!)
    Because of this limitation, it doesn't make sense to have
    more than one of these cards in an RT at any time.  This
    device driver only supports one TRADMYBD card.  If you
    stuff more than one in the system (at different addresses
    of course) the driver will probably complain and will use
   only the first card it finds.

    My understanding of device drivers indicates that this device
    driver code will NOT be re-entered.  This device driver is NOT
    re-entrant.  If this assumption is wrong in future releases of
    BSD Unix(tm) please let me know!
*/


/*
    Two levels of extra message reporting are available:

    DESVERBOSE is to be used when debugging
        user code and causes the device driver
        to tell the user things that the program
        can control.
```

```
    DESDEBUG is to be used when altering the
        device driver or when you are having
        trouble getting the driver to run with
        your system configuration.

    The driver is shipped with both of these reporting levels turned on.
*/

#define DESVERBOSE
#define DESDEBUG

#ifdef DESVERBOSE
#define DESEITHER
#endif
#ifdef DESDEBUG
#define DESEITHER
#endif

/* don't bother if there are no DES devices to be configured into the
system */
#include "des.h"
#if NDES > 0

#include "../h/param.h"
#include "../h/buf.h"
#include "../machineio/ioccvar.h"
#include "../machineio/desreg.h"
#include "../machine/mmu.h"
#include "../h/desioctl.h"
#include "../h/des.h"
#include "../h/errno.h"
#include "../h/ioctl.h"

struct desdevice *desbase = NULL;    /* base address of TRADMYBD */

caddr_t     desaddr[] = {(caddr_t) 0xf0001a90, (caddr_t) 0x00000000};
/* standard places to look */


/* tell C that these are functions */

int desprobe(), desattach(), desintr();
```

```
/* Device Driver structures */

struct iocc_device *(desinfo[NDES]);

struct iocc_driver desdriver =
    {
/*  probe,  slave,      attach,            dgo,     addr,          name,
iocc_dev,   */
    desprobe,     0,    desattach,  0,      desaddr,  "des",  desinfo,
/*  mname,  minfo,        intr,     csr   */
    0,          0,          desintr,      0
    };

/*-------------------------------------------------------------------------
-----*/


/*-------------------------------------------------------------------------
-----*/
/*  useful macros and constants
*/
/*-------------------------------------------------------------------------
-----*/

#define TimeOutCount        (100)
#define     LOCAL           static


/*-------------------------------------------------------------------------
-----*/
/*              globals                                          */
/*-------------------------------------------------------------------------
-----*/

LOCAL int DoingSelfTest = 0;  /* unfortunately needed */


/*-------------------------------------------------------------------------
-----*/
/*                              local functions
*/
/*-------------------------------------------------------------------------
-----*/
```

```
/*
The following are used only by testdev and are declared LOCAL so that
they will not interfere with other, external, uses of these identifiers.

The values are carefully chosen so that they will exercise all the S-
Boxes
of the DES algorithm.
*/

#define testsize    19

LOCAL unsigned char source[testsize][DES_DataGroupSize]  = {
          {0x01, 0xa1, 0xd6, 0xd0,   0x39, 0x77, 0x67, 0x42},
          {0x5c, 0xd5, 0x4c, 0xa8,   0x3d, 0xef, 0x57, 0xda},
          {0x02, 0x48, 0xd4, 0x38,   0x06, 0xf6, 0x71, 0x72},
          {0x51, 0x45, 0x4b, 0x58,   0x2d, 0xdf, 0x44, 0x0a},
          {0x42, 0xfd, 0x44, 0x30,   0x59, 0x57, 0x7f, 0xa2},
          {0x05, 0x9b, 0x5e, 0x08,   0x51, 0xcf, 0x14, 0x3a},
          {0x07, 0x56, 0xd8, 0xe0,   0x77, 0x47, 0x61, 0xd2},
          {0x76, 0x25, 0x14, 0xb8,   0x29, 0xbf, 0x48, 0x6a},
          {0x3b, 0xdd, 0x11, 0x90,   0x49, 0x37, 0x28, 0x02},
          {0x26, 0x95, 0x5f, 0x68,   0x35, 0xaf, 0x60, 0x9a},
          {0x16, 0x4d, 0x5e, 0x40,   0x4f, 0x27, 0x52, 0x32},
          {0x6b, 0x05, 0x6e, 0x18,   0x75, 0x9f, 0x5c, 0xca},
          {0x00, 0x4b, 0xd6, 0xef,   0x09, 0x17, 0x60, 0x62},
          {0x48, 0x0d, 0x39, 0x00,   0x6e, 0xe7, 0x62, 0xf2},
          {0x43, 0x75, 0x40, 0xc8,   0x69, 0x8f, 0x3c, 0xfa},
          {0x07, 0x2d, 0x43, 0xa0,   0x77, 0x07, 0x52, 0x92},
          {0x02, 0xfe, 0x55, 0x77,   0x81, 0x17, 0xf1, 0x2a},
          {0x1d, 0x9d, 0x5c, 0x50,   0x18, 0xf7, 0x28, 0xc2},
          {0x30, 0x55, 0x32, 0x28,   0x6d, 0x6f, 0x29, 0x5a}
                                        };
LOCAL unsigned char results[testsize][DES_DataGroupSize] = {
          {0x69, 0x0f, 0x5b, 0x0d,   0x9a, 0x26, 0x93, 0x9b},
          {0x7a, 0x38, 0x9d, 0x10,   0x35, 0x4b, 0xd2, 0x71},
          {0x86, 0x8e, 0xbb, 0x51,   0xca, 0xb4, 0x59, 0x9a},
          {0x71, 0x78, 0x87, 0x6e,   0x01, 0xf1, 0x9b, 0x2a},
          {0xaf, 0x37, 0xfb, 0x42,   0x1f, 0x8c, 0x40, 0x95},
          {0x86, 0xa5, 0x60, 0xf1,   0x0e, 0xc6, 0xd8, 0x5b},
          {0x0c, 0xd3, 0xda, 0x02,   0x00, 0x21, 0xdc, 0x09},
          {0xea, 0x67, 0x6b, 0x2c,   0xb7, 0xdb, 0x2b, 0x7a},
          {0xdf, 0xd6, 0x4a, 0x81,   0x5c, 0xaf, 0x1a, 0x0f},
          {0x5c, 0x51, 0x3c, 0x9c,   0x48, 0x86, 0xc0, 0x88},
```

```
        {0x0a, 0x2a, 0xee, 0xae,   0x3f, 0xf4, 0xab, 0x77},
        {0xef, 0x1b, 0xf0, 0x3e,   0x5d, 0xfa, 0x57, 0x5a},
        {0x88, 0xbf, 0x0d, 0xb6,   0xd7, 0x0d, 0xee, 0x56},
        {0xa1, 0xf9, 0x91, 0x55,   0x41, 0x02, 0x0b, 0x56},
        {0x6f, 0xbf, 0x1c, 0xaf,   0xcf, 0xfd, 0x05, 0x56},
        {0x2f, 0x22, 0xe4, 0x9b,   0xab, 0x7c, 0xa1, 0xac},
        {0x5a, 0x6b, 0x61, 0x2c,   0xc2, 0x6c, 0xce, 0x4a},
        {0x5f, 0x4c, 0x03, 0x8e,   0xd1, 0x2b, 0x2e, 0x41},
        {0x63, 0xfa, 0xc0, 0xd0,   0x34, 0xd9, 0xf7, 0x93}
                                        };
LOCAL unsigned char key[testsize][DES_KeySize]   = {
        {0x7c, 0xa1, 0x10, 0x45,   0x4a, 0x1a, 0x6e, 0x57},
        {0x01, 0x31, 0xd9, 0x61,   0x9d, 0xc1, 0x37, 0x6e},
        {0x07, 0xa1, 0x13, 0x3e,   0x4a, 0x0b, 0x26, 0x86},
        {0x38, 0x49, 0x67, 0x4c,   0x26, 0x02, 0x31, 0x9e},
        {0x04, 0xb9, 0x15, 0xba,   0x43, 0xfe, 0xb5, 0xb6},
        {0x01, 0x13, 0xb9, 0x70,   0xfd, 0x34, 0xf2, 0xce},
        {0x01, 0x70, 0xf1, 0x75,   0x46, 0x8f, 0xb5, 0xe6},
        {0x43, 0x29, 0x7f, 0xad,   0x38, 0xe3, 0x73, 0xfe},
        {0x07, 0xa7, 0x13, 0x70,   0x45, 0xda, 0x2a, 0x16},
        {0x04, 0x68, 0x91, 0x04,   0xc2, 0xfd, 0x3b, 0x2f},
        {0x37, 0xd0, 0x6b, 0xb5,   0x16, 0xcb, 0x75, 0x46},
        {0x1f, 0x08, 0x26, 0x0d,   0x1a, 0xc2, 0x46, 0x5e},
        {0x58, 0x40, 0x23, 0x64,   0x1a, 0xba, 0x61, 0x76},
        {0x02, 0x58, 0x16, 0x16,   0x46, 0x29, 0xb0, 0x07},
        {0x49, 0x79, 0x3e, 0xbc,   0x79, 0xb3, 0x25, 0x8f},
        {0x4f, 0xb0, 0x5e, 0x15,   0x15, 0xab, 0x73, 0xa7},
        {0x49, 0xe9, 0x5d, 0x6d,   0x4c, 0xa2, 0x29, 0xbf},
        {0x01, 0x83, 0x10, 0xdc,   0x40, 0x9b, 0x26, 0xd6},
        {0x1c, 0x58, 0x7f, 0x1c,   0x13, 0x92, 0x4f, 0xef}
                                        };


LOCAL unsigned char destination[testsize][DES_DataGroupSize];



LOCAL testdev()

/*
This function will look try to test a TRADMYBD des adapter that is
located at
address desbase.  If the adapter is fully operational, testdev will
return
```

```
DES_ERROR_NO_ERROR.
Errors detected include:

    DES_ERROR_NO_ERROR
              Everything seems OK.
    DES_ERROR_BAD_ADDR
              The value of desbase is obvious wrong.
    DES_ERROR_NO_CARD
              The card does not exist in the system or if it is there
      it is too damaged to respond at all.
    DES_ERROR_BAD_CARD
      The card seems to be in the system but some part of the card
      is not functioning properly.  This can also be caused
      if the value of desbase points to some adapter other than
      a TRADMYBD card.
*/

    {
    /* first test for obviously wrong values of addr */

    if ((((unsigned int) desbase) & 0xf000000f) != 0xf0000000)
      {
#ifdef DESDEBUG
      printf("DES:  testdev:  address & 0xf000000f != 0xf0000000.\n");
        printf("DES:  testdev:  address is: 0x%x.\n",(unsigned int)
desbase);
      printf("DES:  testdev:  DES_ERROR_BAD_ADDR returned.\n");
#endif
        return(DES_ERROR_BAD_ADDR);
      }


    /* next look for any board at all */

    desbase->csr = DESCntlReset;     /* try to reset the card */
    /* make sure that only the reset bit is set */
    if (((desbase->csr) & (DESRptNeedsDataM | DESRptHasDataM |
DESRptDCPAddrM | DESRptResetM)) != DESCntlReset)
      {                      /* probably not a TRADMYBD */
#ifdef DESDEBUG
      printf("DES:  testdev:  couldn't find an adapter at address
0x%x.\n",(unsigned int) desbase);
      printf("DES:  testdev:  DES_ERROR_NO_CARD returned.\n");
#endif
```

```
        return(DES_ERROR_NO_CARD);
      }



    /* lastly, run a bunch of carefully selected data through the
adapter to be sure it works. */
    {
    int     i, j, rc;   /* need some storage */
    struct DES_Struct DESp;

        rc = 0;       /* start with result code 0 and add (OR) to it as
we go */

      for (i=0;i<testsize;i++)
          {
          DESp.DES_Src = &source[i][0];       /* set up for desioctl */
          DESp.DES_Dest = &destination[i][0];
          DESp.DES_Length = DES_DataGroupSize;
          DESp.DES_Key[0] = key[i][0];
          DESp.DES_Key[1] = key[i][1];
          DESp.DES_Key[2] = key[i][2];
          DESp.DES_Key[3] = key[i][3];
          DESp.DES_Key[4] = key[i][4];
          DESp.DES_Key[5] = key[i][5];
          DESp.DES_Key[6] = key[i][6];
          DESp.DES_Key[7] = key[i][7];
          DESp.DES_IV = &source[i][0];        /* place holder */

          DoingSelfTest = 1;  /* turn off address space checks */
          rc |= desioctl(0, DES_IOCTL(DES_ENCRYPT_ECB), &DESp, 0); /*
encrypt data */
          DoingSelfTest = 0;  /* turn checks back on */

#ifdef DESDEBUG
          printf("DES:  testdev: rc = 0x%x.\n", rc);
#endif

          for     (j=0;j<DES_DataGroupSize;j++)       /* check results
*/
                rc |= (destination[i][j] != results[i][j]);
          }

      if (rc)
```

```
        {
#ifdef DESDEBUG
        printf("DES:  testdev:  adapter is not functioning
properly\n");
        printf("DES:  testdev:  or it is not really a TRADMYBD
adapter.\n");
        printf("DES:  testdev:  DES_ERROR_BAD_CARD returned.\n");
#endif
        return(DES_ERROR_BAD_CARD);
        }
    }       /* done with i, j, rc and DESp */


    /* got this far, everything must be OK */

#ifdef DESDEBUG
    printf("DES:  testdev:  card at 0x%x seems OK.\n",(unsigned int)
desbase);
    printf("DES:  testdev:  DES_ERROR_NO_ERROR returned.\n");
#endif

    return(DES_ERROR_NO_ERROR);
    }



/*-------------------------------------------------------------------
-----*/
/*                              external functions
*/
/*-------------------------------------------------------------------
-----*/


desprobe(addr)
    register struct desdevice *addr;
    {
    struct desdevice *save_desbase;
    struct DES_Struct DESp;
    int rc;

    save_desbase = desbase;    /* save old value */
    desbase = addr;            /* this is the new one to check out */
```

```
    DESp.DES_Src = &source[0][0];       /* set up for desioctl */
    DESp.DES_Dest = &destination[0][0];
    DESp.DES_Length = 0;
    DESp.DES_Key[0] = 0;
    DESp.DES_Key[1] = 0;
    DESp.DES_Key[2] = 0;
    DESp.DES_Key[3] = 0;
    DESp.DES_Key[4] = 0;
    DESp.DES_Key[5] = 0;
    DESp.DES_Key[6] = 0;
    DESp.DES_Key[7] = 0;
    DESp.DES_IV = &source[0][0];        /* place holder */

    if ((rc = desioctl(0, DES_IOCTL(DES_TEST), &DESp, 0)) ==
DES_ERROR_NO_ERROR)
      {                 /* found a good one! */
      if (save_desbase) /* did we already have a good one? */
          {             /* yes */
          desbase = save_desbase;   /* put back starting value */
          printf("DES:  More than one adapter is in this system.
Only\n");
          printf("DES:  the adapter at 0x%x will be used.\n", (unsigned
int) desbase);
          printf("DES:  The adapter at 0x%x will be ignored.\n",
(unsigned int) addr);
          return (PROBE_BAD);
          }
        else
          {             /* no, use this one */
#ifdef DESDEBUG
          printf("DES:  Operational DES card found at address 0x%x.\n",
(unsigned int) desbase);
#endif
          return(PROBE_NOINT);
          }
        }
      else
      {
      desbase     = save_desbase;       /* put back starting value */
      switch (rc)
          {
          case DES_ERROR_BAD_ADDR:
            printf("DES:  The address of 0x%x is thoroughly
unreasonable...\n",(unsigned int) addr);
```

```
            printf("DES:  This system probably has been configured
incorrectly.\n");
            break;
          case DES_ERROR_NO_CARD:
#ifdef DESEITHER
            printf("DES:  No adapter was found at address 0x%x.\n",
(unsigned int) addr);
#endif
            break;
          case DES_ERROR_BAD_CARD:
            printf("DES:  Either the adapter at address 0x%x is
broken\n",(unsigned int) addr);
            printf("DES:  or some different card is located at this
address.\n");
            printf("DES:  Have the adapter serviced or correct the
system\n");
            printf("DES:  configuration for this address.\n");
            break;
#ifdef DESDEBUG
          default:
            printf("DES:  This should never be executed in
desprobe!\n");
#endif
          }
        }
    return(PROBE_BAD);        /* if we got this far the adapter is bad
*/
    }

/*-------------------------------------------------------------------
-----*/

desattach()
    {
#ifdef DESDEBUG
    printf("DES:  in desattach\n");
#endif
    DoingSelfTest = 0;      /* just to be sure */
    return(0);
    }

/*-------------------------------------------------------------------
-----*/
```

```
desintr()
    {
#ifdef DESDEBUG
    printf("DES:  desintr invoked\n");
#endif
    return(1);
    }

/*-------------------------------------------------------------------------
-----*/

desopen(dev, flag)
    dev_t dev;
    int flag;
    {
#ifdef DESEITHER
    printf("DES:  in desopen\n");
#endif
    if (desbase)
        return(0);
      else
      return(ENXIO);
    }

/*-------------------------------------------------------------------------
-----*/

desclose(dev, flag)
    dev_t dev;
    int flag;

    {
#ifdef DESEITHER
    printf("DES:  in desclose\n");
#endif
    return(0);
    }

/*-------------------------------------------------------------------------
-----*/

desread()
    {
#ifdef DESEITHER
```

```
    printf("DES:  desread() function was called!\n");
#endif
    return(ENODEV);
    }

/*-------------------------------------------------------------------------
-----*/

deswrite()
    {
#ifdef DESEITHER
    printf("DES:  deswrite() function was called!\n");
#endif
    return(ENODEV);
    }

/*-------------------------------------------------------------------------
-----*/

desioctl(dev, cmd, DESp, flag)
    dev_t dev;    /* ignored */
    int cmd;
    register struct DES_Struct *DESp;
    int flag;     /* ignored */
    {
      register int i, j;
        register int ThisCount;

/*

This routine will do the following:

 1:  check cmd to make sure it is valid
 2:  check the length of the data to make sure it is valid
 3:  set the mode of TRADMBYD (ECB, CBC, CF)
 4:  if needed, set the IV
 5:  load the key and check for odd parity
 6:  if length of the data is > 0 continue
 7:  check the addresses for the input and output buffers
 8:  move the data through TRADMYBD (pipeline if appropriate)
 9:  if needed, retrieve the IV

This allows the function to be called with a length of 0 to test
a key to see if it is valid.
```

```
NOTE:
I realize this function is just SCREAMING to be broken up
into a number of smaller functions.  Unfortunately, to get
the best performance possible, I have decided that it should
really be one giant piece of code.  I have tried to break the
code into a number of steps that would lend themselves to
being sub-functions and have tried to lable what is happening
where.  Sorry if this makes you pull your hair out but some
performance tests I ran show you really CAN see the difference.
In short, if you think it is broken and can't stand to touch the
code, ask me to fix it.
*/


#ifdef DESEITHER
    printf("DES:  Entering IOCTL...\n");
    printf("DES:  \n");
    printf("DES:  cmd: .......0x%x\n", cmd);
    printf("DES:  DESp: ......0x%x\n", (unsigned int) DESp);
    printf("DES:    Src: .......0x%x\n", (unsigned int) (DESp-
>DES_Src));
    printf("DES:      Value: .....0x%lx\n", * ((unsigned long *) (DESp-
>DES_Src)));
    printf("DES:    Dest: ......0x%x\n", (unsigned int) (DESp-
>DES_Dest));
    printf("DES:      Value: .....0x%lx\n", * ((unsigned long *) (DESp-
>DES_Dest)));
    printf("DES:    Length: ....0x%x\n", (DESp->DES_Length));
    printf("DES:    Key: .......0x%lx\n", * ((unsigned long *) (DESp-
>DES_Key)));
    printf("DES:    IV: ........0x%x\n", (unsigned int) (DESp->DES_IV));
    printf("DES:      Value: .....0x%lx\n", * ((unsigned long *) (DESp-
>DES_IV)));
    printf("DES:  \n");
    printf("DES:  End of parameter list...\n");
    printf("DES:  \n");
#endif



/* 1. check cmd for reasonable values */

    switch (cmd)
```

```
      {
      case DES_IOCTL(DES_TEST):
          return(testdev());            /* that's all we have to do! */

      case DES_IOCTL(DES_ENCRYPT_ECB):    /* 2. check length while we're
here */
      case DES_IOCTL(DES_DECRYPT_ECB):
      case DES_IOCTL(DES_ENCRYPT_CBC):
      case DES_IOCTL(DES_DECRYPT_CBC):
          if ((DESp->DES_Length) % DES_DataGroupSize)
            return(DES_ERROR_SIZE);
          break;

      case DES_IOCTL(DES_ENCRYPT_CF):         /* 2. no restrictions on
length */
      case DES_IOCTL(DES_DECRYPT_CF):
          break;

      default: return(DES_ERROR_COMMAND); /* this is not a valid
command! */
      }


/* 3. set the mode register */

    desbase->csr = DESCntlMode;
    switch (cmd)
      {
      case DES_IOCTL(DES_ENCRYPT_ECB):
      case DES_IOCTL(DES_DECRYPT_ECB):
          desbase->data.b[0] = (DESModeSingle | DESModeECB);
          break;
      case DES_IOCTL(DES_ENCRYPT_CBC):
      case DES_IOCTL(DES_DECRYPT_CBC):
          desbase->data.b[0] = (DESModeSingle | DESModeCBC);
          break;
      case DES_IOCTL(DES_ENCRYPT_CF):
      case DES_IOCTL(DES_DECRYPT_CF):
          desbase->data.b[0] = (DESModeSingle | DESModeCF);
          break;
#ifdef DESDEBUG
      default: goto error;          /* something TERRIBLE happend!!! */
#endif
      }
```

```
/* 4. if needed, load the IV register */

    switch (cmd)
      {
      case DES_IOCTL(DES_ENCRYPT_ECB):
      case DES_IOCTL(DES_DECRYPT_ECB):
          break;
      case DES_IOCTL(DES_ENCRYPT_CBC):
      case DES_IOCTL(DES_ENCRYPT_CF):
          desbase->csr = DESCntlCommand;
          desbase->data.b[0] = DESCmdLdClEIV;
#ifdef DESDEBUG
          printf("DES:  after DESCmdLdClEIV status reg =
0x%x.\n",desbase->data.b[0]);
#endif
          desbase->csr = DESCntlIO;
          for (i=0;i<DES_IVSize;i++)
            {
#ifdef DESDEBUG
            printf("DES:  EIV out: 0x%x\n",DESp->DES_IV[i]);
#endif
            desbase->data.b[0] = DESp->DES_IV[i];
            }
          break;
      case DES_IOCTL(DES_DECRYPT_CBC):
      case DES_IOCTL(DES_DECRYPT_CF):
          desbase->csr = DESCntlCommand;
          desbase->data.b[0] = DESCmdLdClDIV;
#ifdef DESDEBUG
          printf("DES:  after DESCmdLdClDIV status reg =
0x%x.\n",desbase->data.b[0]);
#endif
          desbase->csr = DESCntlIO;
          for (i=0;i<DES_IVSize;i++)
            {
#ifdef DESDEBUG
            printf("DES:  DIV out: 0x%x\n",DESp->DES_IV[i]);
#endif
            desbase->data.b[0] = DESp->DES_IV[i];
            }
          break;
#ifdef DESDEBUG
```

```
      default: goto error;        /* something TERRIBLE happend!!! */
#endif
      }


/* 5. set the key and check for all odd parity */

    desbase->csr = DESCntlCommand;
    switch (cmd)
      {
      case DES_IOCTL(DES_ENCRYPT_ECB):
      case DES_IOCTL(DES_ENCRYPT_CBC):
      case DES_IOCTL(DES_ENCRYPT_CF):
          desbase->data.b[0] = DESCmdLdClrEKey;
          break;
      case DES_IOCTL(DES_DECRYPT_ECB):
      case DES_IOCTL(DES_DECRYPT_CBC):
      case DES_IOCTL(DES_DECRYPT_CF):
          desbase->data.b[0] = DESCmdLdClrDKey;
          break;
#ifdef DESDEBUG
      default: goto error;        /* something TERRIBLE happend!!! */
#endif
      }

    desbase->csr = DESCntlIO;
    for (i = 0; i < DES_KeySize; desbase->data.b[0] = DESp-
>DES_Key[i++]) ;
    desbase->csr = DESCntlStatus;
    if ((desbase->data.b[0]) & DESStatLParM)
      {
#ifdef DESEITHER
      printf("DES:  key had bad parity.\n");
#endif
      return(DES_ERROR_KEY);
      }


/* 6. if the size of data is 0 then only the key is being tested so
return */

    if (DESp->DES_Length == 0)
      return(DES_ERROR_NO_ERROR);
```

```
/* 7. check buffer addresses and addressability */

    if (DoingSelfTest == 0)
      {
        if (isitok(DESp->DES_Src, DESp->DES_Length, UREAD_OK) == 0)
          {
          return(DES_ERROR_BUFFER);
          }
        if (isitok(DESp->DES_Dest, DESp->DES_Length, UWRITE_OK) == 0)
          {
          return(DES_ERROR_BUFFER);
          }
        }

/* 8. move data through TRADMYBD */

    ThisCount = DESp->DES_Length / DES_DataGroupSize;

    desbase->csr = DESCntlCommand;
    switch (cmd)
      {
      case DES_IOCTL(DES_ENCRYPT_ECB):
      case DES_IOCTL(DES_ENCRYPT_CBC):
      case DES_IOCTL(DES_ENCRYPT_CF):
          desbase->data.b[0] = DESCmdStartE;
          break;
      case DES_IOCTL(DES_DECRYPT_ECB):
      case DES_IOCTL(DES_DECRYPT_CBC):
      case DES_IOCTL(DES_DECRYPT_CF):
          desbase->data.b[0] = DESCmdStartD;
          break;
#ifdef DESDEBUG
      default: goto error;          /* something TERRIBLE happend!!! */
#endif
      }

    desbase->csr = DESCntlIO;

/*
big switch to determine if data must be moved through TRADMYBD in
block or byte form.  the block form is further subdivided into the
cases of 1 block or more than 1 block.
*/
```

```
    switch (cmd)
      {
      case DES_IOCTL(DES_ENCRYPT_ECB):    /* these modes do data in
blocks */
      case DES_IOCTL(DES_DECRYPT_ECB):
      case DES_IOCTL(DES_ENCRYPT_CBC):
      case DES_IOCTL(DES_DECRYPT_CBC):
          if (ThisCount == 1)           /* only 1 block */
            {               /* special case, no pipelining */
            register unsigned int *cpi, *cpo;

            cpi = (unsigned int *) DESp->DES_Src;
            cpo = (unsigned int *) DESp->DES_Dest;

#ifdef DESDEBUG
            printf("DES:  out:  %x\n",*(cpi));
            printf("DES:  out:  %x\n",*(cpi+1));
#endif
            i = TimeOutCount;   /* time out value */
            while ((((desbase->csr) & DESRptNeedsDataM) == 0) && (i--))
;
            if (i==0)
                {
                printf("DES:  Adapter timed out!!  System may be
unusable.\n");
                return(DES_ERROR_HARDWARE);
                }
            desbase->data.w[0] = *(cpi++);
            desbase->data.w[0] = *(cpi++);
            i = TimeOutCount;   /* time out value */
            while ((((desbase->csr) & DESRptHasDataM) == 0) && (i--)) ;
            if (i==0)
                {
                printf("DES:  Adapter timed out!!  System may be
unusable.\n");
                return(DES_ERROR_HARDWARE);
                }
            *(cpo++) = desbase->data.w[0];
            *(cpo++) = desbase->data.w[0];
#ifdef DESDEBUG
              printf("DES:   in:  %x\n",*(cpo-2));
              printf("DES:   in:  %x\n",*(cpo-1));
```

```
#endif
            }
            else            /* more than 1 block */
            {               /* pipeline data through TRADMYBD */
            register unsigned int *cpi, *cpo;
            register unsigned int *DESport;

            cpi = (unsigned int *) DESp->DES_Src;
            cpo = (unsigned int *) DESp->DES_Dest;
            DESport = &(desbase->data.w[0]);

#ifdef DESDEBUG
            printf("DES:  out:  %x\n",*(cpi));
            printf("DES:  out:  %x\n",*(cpi+1));
#endif

            *DESport = *(cpi++);
            *DESport = *(cpi++);

            i = TimeOutCount;   /* time out value */
            while ((((desbase->csr) & DESRptNeedsDataM) == 0) && (i--))
;
            if (i==0)
                {
                printf("DES:  Adapter timed out!!  System may be
unusable.\n");
                return(DES_ERROR_HARDWARE);
                }

            for (i=0;i<(ThisCount-1);i++)
                {
#ifdef DESDEBUG
            printf("DES:  out:  %x\n",*(cpi));
            printf("DES:  out:  %x\n",*(cpi+1));
#endif
                *DESport = *(cpi++);
                *DESport = *(cpi++);

                *(cpo++) = *DESport;
                *(cpo++) = *DESport;
#ifdef DESDEBUG
            printf("DES:  in:   %x\n",*(cpo-2));
            printf("DES:  in:   %x\n",*(cpo-1));
#endif
```

```
                }

            i = TimeOutCount;   /* time out value */
            while ((((desbase->csr) & DESRptHasDataM) == 0) && (i--)) ;
            if (i==0)
                {
                printf("DES:  Adapter timed out!!  System may be
unusable.\n");
                return(DES_ERROR_HARDWARE);
                }

            *(cpo++) = *DESport;
            *(cpo++) = *DESport;

#ifdef DESDEBUG
            printf("DES:  in:   %x\n",*(cpo-2));
            printf("DES:  in:   %x\n",*(cpo-1));
#endif
            }
          break;

/*  more of switch (cmd) */
      case DES_IOCTL(DES_ENCRYPT_CF):         /* this mode does data by
the byte */
      case DES_IOCTL(DES_DECRYPT_CF):
          {
          register unsigned char *cpi, *cpo;

          cpi = (unsigned char *) DESp->DES_Src;
          cpo = (unsigned char *) DESp->DES_Dest;

          for (i=0;i<DESp->DES_Length;i++)
            {
#ifdef DESDEBUG
                  printf("DES:  out:  %x\n",*(cpi));
#endif
                  j = TimeOutCount;   /* time out value */
                  while ((((desbase->csr) & DESRptNeedsDataM) == 0) &&
(j--)) ;
                  if (j==0)
                      {
                      printf("DES:  Adapter timed out!!  System may be
unusable.\n");
                      return(DES_ERROR_HARDWARE);
```

```
                    }
                desbase->data.b[0] = *(cpi++);
                j = TimeOutCount;   /* time out value */
                while ((((desbase->csr) & DESRptHasDataM) == 0) && (j-
-)) ;
                if (j==0)
                    {
                    printf("DES:  Adapter timed out!!  System may be
unusable.\n");
                    return(DES_ERROR_HARDWARE);
                    }
                *(cpo++) = desbase->data.b[0];
#ifdef DESDEBUG
                  printf("DES:   in:  %x\n",*(cpo-1));
#endif
            }
          }
          break;

/*  end of switch (cmd) */
#ifdef DESDEBUG
      default: goto error;        /* something TERRIBLE happend!!! */
#endif
      }


/* shut down TRADMYBD */

    desbase->csr = DESCntlCommand;
    desbase->data.b[0] = DESCmdStop;


/* 9. if needed, unload the IV register */

    switch (cmd)
      {
      case DES_IOCTL(DES_ENCRYPT_ECB):
      case DES_IOCTL(DES_DECRYPT_ECB):
          break;
      case DES_IOCTL(DES_ENCRYPT_CBC):
      case DES_IOCTL(DES_ENCRYPT_CF):
          desbase->csr = DESCntlCommand;
          desbase->data.b[0] = DESCmdRdClEIV;
          desbase->csr = DESCntlIO;
```

```
          for (i=0;i<DES_IVSize;i++)
            {
            DESp->DES_IV[i] = desbase->data.b[0];
#ifdef DESDEBUG
            printf("DES:  EIV in: 0x%x\n",DESp->DES_IV[i]);
#endif
            }
          break;
      case DES_IOCTL(DES_DECRYPT_CBC):
      case DES_IOCTL(DES_DECRYPT_CF):
          desbase->csr = DESCntlCommand;
          desbase->data.b[0] = DESCmdRdClDIV;
          desbase->csr = DESCntlIO;
          for (i=0;i<DES_IVSize;i++)
            {
            DESp->DES_IV[i] = desbase->data.b[0];
#ifdef DESDEBUG
            printf("DES:  DIV in: 0x%x\n",DESp->DES_IV[i]);
#endif
            }
          break;
#ifdef DESDEBUG
      default: goto error;        /* something TERRIBLE happend!!! */
#endif
      }


/* all done! */

    return(DES_ERROR_NO_ERROR);

#ifdef DESDEBUG
error:    /* we only get here if something TERRIBLE happened */
    printf("DES:  Internal device driver error!  Notify system
administrators.\n");
    return(DES_ERROR_SOFTWARE);
#endif
    }

/* #endif NDES > 1 */
#endif
```

DES.H

This include file describes the interface exported to application programs.

```
/*
start of des.h
*/

/*
    Written by Paul G. Crumley
    Carnegie-Mellon University
    Information Technology Center
    Pittsburgh, PA  15213

    February 1987

    Device driver for Paul's TRADMYBD DES card
*/


/* useful constants */

#define DES_DataGroupSize   8
#define DES_KeySize         8
#define DES_IVSize          8


/* op codes for DES */

#define DES_TEST  0
#define DES_ENCRYPT_ECB 1
#define DES_DECRYPT_ECB 2
#define DES_ENCRYPT_CBC 3
#define DES_DECRYPT_CBC 4
#define DES_ENCRYPT_CF  5
#define DES_DECRYPT_CF  6


/* values for errno returned by DES */

/* these are caused by somthing the program did/didn't do */
```

```
#define DES_ERROR_COMMAND     101
#define DES_ERROR_SIZE        102
#define DES_ERROR_KEY         103
#define DES_ERROR_BUFFER      104

/* these may be transients, try again before giving up but */
/*  do give up after one or two more tries                 */

#define DES_ERROR_GENERAL     150

/* these are problems you can't do anything about... */

#define DES_ERROR_SOFTWARE    200
#define DES_ERROR_HARDWARE    210


/* useful types and structures */

typedef unsigned char   DES_Src_t;
typedef unsigned char   DES_Dest_t;
typedef int       DES_Length_t;
typedef unsigned char   DES_Key_t;
typedef unsigned char   DES_IV_t;


struct DES_Struct /* passed in DES_IOCTL */
    {
    DES_Src_t         *DES_Src;
    DES_Dest_t        *DES_Dest;
    DES_Length_t    DES_Length;
    DES_Key_t         DES_Key[DES_KeySize];
    DES_IV_t          *DES_IV;
    };


typedef int DES_Descriptor;


/* useful macros */

#ifndef DES_IOCTL
#include <desioctl.h>
#endif DES_IOCTL
```

```
#define DES_Open()                  (open("/dev/des0",O_RDONLY,0))
#define DES(dev,op,parms)           (ioctl(dev,DES_IOCTL(op),parms))
#define DES_Close(dev)              (close(dev))

/*
end of des.h
*/
```

DESTEST.C

This program will test the TRADMYBD adapter and software. It also shows how the DES() interface is used.

```
/*
start of destest.c
*/

/*
    Written by Paul G. Crumley
    Carnegie-Mellon University
    Information Technology Center
    Pittsburgh, PA  15213

    February 1987

*/

#include <sys/types.h>
#include <sys/ioctl.h>
#include <sys/file.h>
#include "desioctl.h"
#include "des.h"




/*
These are used to test the DES special device.
*/

#define testsize    19

static unsigned char source[testsize][DES_DataGroupSize]  = {
          {0x01, 0xa1, 0xd6, 0xd0,   0x39, 0x77, 0x67, 0x42},
          {0x5c, 0xd5, 0x4c, 0xa8,   0x3d, 0xef, 0x57, 0xda},
          {0x02, 0x48, 0xd4, 0x38,   0x06, 0xf6, 0x71, 0x72},
          {0x51, 0x45, 0x4b, 0x58,   0x2d, 0xdf, 0x44, 0x0a},
          {0x42, 0xfd, 0x44, 0x30,   0x59, 0x57, 0x7f, 0xa2},
          {0x05, 0x9b, 0x5e, 0x08,   0x51, 0xcf, 0x14, 0x3a},
```

```
        {0x07, 0x56, 0xd8, 0xe0,   0x77, 0x47, 0x61, 0xd2},
        {0x76, 0x25, 0x14, 0xb8,   0x29, 0xbf, 0x48, 0x6a},
        {0x3b, 0xdd, 0x11, 0x90,   0x49, 0x37, 0x28, 0x02},
        {0x26, 0x95, 0x5f, 0x68,   0x35, 0xaf, 0x60, 0x9a},
        {0x16, 0x4d, 0x5e, 0x40,   0x4f, 0x27, 0x52, 0x32},
        {0x6b, 0x05, 0x6e, 0x18,   0x75, 0x9f, 0x5c, 0xca},
        {0x00, 0x4b, 0xd6, 0xef,   0x09, 0x17, 0x60, 0x62},
        {0x48, 0x0d, 0x39, 0x00,   0x6e, 0xe7, 0x62, 0xf2},
        {0x43, 0x75, 0x40, 0xc8,   0x69, 0x8f, 0x3c, 0xfa},
        {0x07, 0x2d, 0x43, 0xa0,   0x77, 0x07, 0x52, 0x92},
        {0x02, 0xfe, 0x55, 0x77,   0x81, 0x17, 0xf1, 0x2a},
        {0x1d, 0x9d, 0x5c, 0x50,   0x18, 0xf7, 0x28, 0xc2},
        {0x30, 0x55, 0x32, 0x28,   0x6d, 0x6f, 0x29, 0x5a}
                                        };
static unsigned char results[testsize][DES_DataGroupSize] = {
        {0x69, 0x0f, 0x5b, 0x0d,   0x9a, 0x26, 0x93, 0x9b},
        {0x7a, 0x38, 0x9d, 0x10,   0x35, 0x4b, 0xd2, 0x71},
        {0x86, 0x8e, 0xbb, 0x51,   0xca, 0xb4, 0x59, 0x9a},
        {0x71, 0x78, 0x87, 0x6e,   0x01, 0xf1, 0x9b, 0x2a},
        {0xaf, 0x37, 0xfb, 0x42,   0x1f, 0x8c, 0x40, 0x95},
        {0x86, 0xa5, 0x60, 0xf1,   0x0e, 0xc6, 0xd8, 0x5b},
        {0x0c, 0xd3, 0xda, 0x02,   0x00, 0x21, 0xdc, 0x09},
        {0xea, 0x67, 0x6b, 0x2c,   0xb7, 0xdb, 0x2b, 0x7a},
        {0xdf, 0xd6, 0x4a, 0x81,   0x5c, 0xaf, 0x1a, 0x0f},
        {0x5c, 0x51, 0x3c, 0x9c,   0x48, 0x86, 0xc0, 0x88},
        {0x0a, 0x2a, 0xee, 0xae,   0x3f, 0xf4, 0xab, 0x77},
        {0xef, 0x1b, 0xf0, 0x3e,   0x5d, 0xfa, 0x57, 0x5a},
        {0x88, 0xbf, 0x0d, 0xb6,   0xd7, 0x0d, 0xee, 0x56},
        {0xa1, 0xf9, 0x91, 0x55,   0x41, 0x02, 0x0b, 0x56},
        {0x6f, 0xbf, 0x1c, 0xaf,   0xcf, 0xfd, 0x05, 0x56},
        {0x2f, 0x22, 0xe4, 0x9b,   0xab, 0x7c, 0xa1, 0xac},
        {0x5a, 0x6b, 0x61, 0x2c,   0xc2, 0x6c, 0xce, 0x4a},
        {0x5f, 0x4c, 0x03, 0x8e,   0xd1, 0x2b, 0x2e, 0x41},
        {0x63, 0xfa, 0xc0, 0xd0,   0x34, 0xd9, 0xf7, 0x93}
                                        };
static unsigned char key[testsize][DES_DataGroupSize]  = {
        {0x7c, 0xa1, 0x10, 0x45,   0x4a, 0x1a, 0x6e, 0x57},
        {0x01, 0x31, 0xd9, 0x61,   0x9d, 0xc1, 0x37, 0x6e},
        {0x07, 0xa1, 0x13, 0x3e,   0x4a, 0x0b, 0x26, 0x86},
        {0x38, 0x49, 0x67, 0x4c,   0x26, 0x02, 0x31, 0x9e},
        {0x04, 0xb9, 0x15, 0xba,   0x43, 0xfe, 0xb5, 0xb6},
        {0x01, 0x13, 0xb9, 0x70,   0xfd, 0x34, 0xf2, 0xce},
        {0x01, 0x70, 0xf1, 0x75,   0x46, 0x8f, 0xb5, 0xe6},
        {0x43, 0x29, 0x7f, 0xad,   0x38, 0xe3, 0x73, 0xfe},
```

```
        {0x07, 0xa7, 0x13, 0x70,    0x45, 0xda, 0x2a, 0x16},
        {0x04, 0x68, 0x91, 0x04,    0xc2, 0xfd, 0x3b, 0x2f},
        {0x37, 0xd0, 0x6b, 0xb5,    0x16, 0xcb, 0x75, 0x46},
        {0x1f, 0x08, 0x26, 0x0d,    0x1a, 0xc2, 0x46, 0x5e},
        {0x58, 0x40, 0x23, 0x64,    0x1a, 0xba, 0x61, 0x76},
        {0x02, 0x58, 0x16, 0x16,    0x46, 0x29, 0xb0, 0x07},
        {0x49, 0x79, 0x3e, 0xbc,    0x79, 0xb3, 0x25, 0x8f},
        {0x4f, 0xb0, 0x5e, 0x15,    0x15, 0xab, 0x73, 0xa7},
        {0x49, 0xe9, 0x5d, 0x6d,    0x4c, 0xa2, 0x29, 0xbf},
        {0x01, 0x83, 0x10, 0xdc,    0x40, 0x9b, 0x26, 0xd6},
        {0x1c, 0x58, 0x7f, 0x1c,    0x13, 0x92, 0x4f, 0xef}
                                        };

static unsigned char destination[testsize][DES_DataGroupSize];



static unsigned char nowsource[3][DES_DataGroupSize]  = {
        {0x4e, 0x6f, 0x77, 0x20,     0x69, 0x73, 0x20, 0x74},
        {0x68, 0x65, 0x20, 0x74,     0x69, 0x6d, 0x65, 0x20},
        {0x66, 0x6f, 0x72, 0x20,     0x61, 0x6c, 0x6c, 0x20}
                                        };

static unsigned char nowresultsECB[3][DES_DataGroupSize]    = {
        {0x3f, 0xa4, 0x0e, 0x8a,     0x98, 0x4d, 0x48, 0x15},
        {0x6a, 0x27, 0x17, 0x87,     0xab, 0x88, 0x83, 0xf9},
        {0x89, 0x3d, 0x51, 0xec,     0x4b, 0x56, 0x3b, 0x53}
                                        };


static unsigned char nowresultsCBC[3][DES_DataGroupSize]    = {
        {0xe5, 0xc7, 0xcd, 0xde,     0x87, 0x2b, 0xf2, 0x7c},
        {0x43, 0xe9, 0x34, 0x00,     0x8c, 0x38, 0x9c, 0x0f},
        {0x68, 0x37, 0x88, 0x49,     0x9a, 0x7c, 0x05, 0xf6}
                                        };

static unsigned char zerothruf [DES_KeySize] =
        {0x01, 0x23, 0x45, 0x67,     0x89, 0xab, 0xcd, 0xef};

static unsigned char zeroes [DES_KeySize] =
        {0x00, 0x00, 0x00, 0x00,     0x00, 0x00, 0x00, 0x00};

static unsigned char IVarea [DES_IVSize];
```

```
extern errno;

main(argc, argv)
    int argc;
    char *argv[];
    {
    int desfd;
    struct DES_Struct desobj;
    int i, j, rc;

    if ((desfd = DES_Open()) < 0)
      {perror("destest:  DES_Open"); exit(-1);}

    printf("desfd = %d.\n",desfd);

/*
These tests will exercise all the S-Boxes of the DES chip.
*/

  printf("\n** TESTING ECB ENCRYPTION **\n\n");

  for (i=0;i<testsize;i++)
   {
    desobj.DES_Src = &source[i][0];
    desobj.DES_Dest = &destination[i][0];
    desobj.DES_Length = DES_DataGroupSize;
    for (j=0;j<DES_KeySize;j++)
        desobj.DES_Key[j] = key[i][j];

    printf("test #%d.\n",i);

    printf("SRC:  ");
    print_hex_array(&source[i][0]);

    printf("KEY:  ");
    print_hex_array(&key[i][0]);

    if ((rc = DES(desfd,DES_ENCRYPT_ECB,&desobj)) < 0)
        {
      printf("destest:  error testing S-boxes on itteration %d.\n",i);
      printf("rc = 0x%x = %d, errno = %d.\n", rc, rc, errno);
      perror("encrypt");
      }
```

```
  printf("DEST: ");
  print_hex_array(&destination[i][0]);

  printf("CHK:  ");
  print_hex_array(&results[i][0]);

  printf("\n");

  rc = 0;
  for (j=0;j<DES_DataGroupSize;j++)
    rc |= (destination[i][j] != results[i][j]);
  if (rc)
    {
    printf("destest:  error testing S-boxes on itteration %d.\n",i);
    printf("            data error.\n");
    }
 }



printf("\n** TESTING ECB DECRYPTION **\n\n");

for (i=0;i<testsize;i++)
 {
  desobj.DES_Src = &results[i][0];
  desobj.DES_Dest = &destination[i][0];
  desobj.DES_Length = DES_DataGroupSize;
  for (j=0;j<DES_KeySize;j++)
      desobj.DES_Key[j] = key[i][j];

  printf("test #%d.\n",i);

  printf("SRC:  ");
  print_hex_array(&results[i][0]);

  printf("KEY:  ");
  print_hex_array(&key[i][0]);

  if ((rc = DES(desfd,DES_DECRYPT_ECB,&desobj)) < 0)
      {
    printf("destest:  error testing S-boxes on itteration %d.\n",i);
    printf("rc = 0x%x = %d, errno = %d.\n", rc, rc, errno);
    perror("decrypt");
```

```
    }

  printf("DEST: ");
  print_hex_array(&destination[i][0]);

  printf("CHK:  ");
  print_hex_array(&source[i][0]);

  printf("\n");

  rc = 0;
  for (j=0;j<DES_DataGroupSize;j++)
    rc |= (destination[i][j] != source[i][j]);
  if (rc)
    {
    printf("destest:  error testing S-boxes on itteration %d.\n",i);
    printf("            data error.\n");
    }
}

  printf("S-box test complete.\n");


  printf("ECB test.\n");

  desobj.DES_Src = &nowsource[0][0];
  desobj.DES_Dest = &destination[0][0];
  desobj.DES_Length = 3 * DES_DataGroupSize;
  for (j=0;j<DES_KeySize;j++)
      desobj.DES_Key[j] = zerothruf[j];

  printf("SRC:  ");
  print_hex_array(&nowsource[0][0]);
  print_hex_array(&nowsource[1][0]);
  print_hex_array(&nowsource[2][0]);

  printf("KEY:  ");
  print_hex_array(&zerothruf[0]);

  if ((rc = DES(desfd,DES_ENCRYPT_ECB,&desobj)) < 0)
      {
    printf("destest:  error.\n");
    printf("rc = 0x%x = %d, errno = %d.\n", rc, rc, errno);
    }
```

```
    printf("DEST: ");
    print_hex_array(&destination[0][0]);
    print_hex_array(&destination[1][0]);
    print_hex_array(&destination[2][0]);

    printf("CHK:  ");
    print_hex_array(&nowresultsECB[0][0]);
    print_hex_array(&nowresultsECB[1][0]);
    print_hex_array(&nowresultsECB[2][0]);

    printf("\n");

    rc = 0;
    for (i=0;i<3;i++)
     for (j=0;j<DES_DataGroupSize;j++)
      rc |= (destination[i][j] != nowresultsECB[i][j]);
    if (rc)
      {
/*    printf("destest:  data error.\n");*/
      }




    desobj.DES_Src = &destination[0][0];
    desobj.DES_Dest = &destination[0][0];
    desobj.DES_Length = 3 * DES_DataGroupSize;
    for (j=0;j<DES_KeySize;j++)
        desobj.DES_Key[j] = zerothruf[j];

    printf("SRC:  ");
    print_hex_array(&destination[0][0]);
    print_hex_array(&destination[1][0]);
    print_hex_array(&destination[2][0]);

    printf("KEY:  ");
    print_hex_array(&zerothruf[0]);

    if ((rc = DES(desfd,DES_DECRYPT_ECB,&desobj)) < 0)
        {
      printf("destest:  error.\n");
      printf("rc = 0x%x = %d, errno = %d.\n", rc, rc, errno);
```

```
    }

  printf("DEST: ");
  print_hex_array(&destination[0][0]);
  print_hex_array(&destination[1][0]);
  print_hex_array(&destination[2][0]);

  printf("CHK:  ");
  print_hex_array(&nowsource[0][0]);
  print_hex_array(&nowsource[1][0]);
  print_hex_array(&nowsource[2][0]);

  printf("\n");

  rc = 0;
  for (i=0;i<3;i++)
   for (j=0;j<DES_DataGroupSize;j++)
    rc |= (destination[i][j] != nowsource[i][j]);
  if (rc)
    {
/*   printf("destest:  data error.\n");*/
    }



  printf("CBC test.\n");

  desobj.DES_Src = &nowsource[0][0];
  desobj.DES_Dest = &destination[0][0];
  desobj.DES_Length = 3 * DES_DataGroupSize;
  for (j=0;j<DES_KeySize;j++)
      desobj.DES_Key[j] = zerothruf[j];
  for (j=0;j<DES_IVSize;j++)
      IVarea[j] = zeroes[j];
  desobj.DES_IV = &IVarea[0];

  printf("SRC:  ");
  print_hex_array(&nowsource[0][0]);
  print_hex_array(&nowsource[1][0]);
  print_hex_array(&nowsource[2][0]);

  printf("KEY:  ");
  print_hex_array(&zerothruf[0]);
```

```
    printf("IV:    ");
    print_hex_array(&IVarea[0]);

    if ((rc = DES(desfd,DES_ENCRYPT_CBC,&desobj)) < 0)
        {
      printf("destest:  error.\n");
      printf("rc = 0x%x = %d, errno = %d.\n", rc, rc, errno);
      }

    printf("DEST: ");
    print_hex_array(&destination[0][0]);
    print_hex_array(&destination[1][0]);
    print_hex_array(&destination[2][0]);

    printf("CHK:  ");
    print_hex_array(&nowresultsCBC[0][0]);
    print_hex_array(&nowresultsCBC[1][0]);
    print_hex_array(&nowresultsCBC[2][0]);

    printf("IV:    ");
    print_hex_array(&IVarea[0]);

    printf("\n");

    rc = 0;
    for (i=0;i<3;i++)
     for (j=0;j<DES_DataGroupSize;j++)
      rc |= (destination[i][j] != nowresultsCBC[i][j]);
    if (rc)
      {
/*    printf("destest:  data error.\n");*/
      }




    desobj.DES_Src = &destination[0][0];
    desobj.DES_Dest = &destination[0][0];
    desobj.DES_Length = 3 * DES_DataGroupSize;
    for (j=0;j<DES_KeySize;j++)
        desobj.DES_Key[j] = zerothruf[j];
    for (j=0;j<DES_IVSize;j++)
```

```
        IVarea[j] = zeroes[j];
    desobj.DES_IV = &IVarea[0];

    printf("SRC:  ");
    print_hex_array(&destination[0][0]);
    print_hex_array(&destination[1][0]);
    print_hex_array(&destination[2][0]);

    printf("KEY:  ");
    print_hex_array(&zerothruf[0]);

    printf("IV:   ");
    print_hex_array(&IVarea[0]);

    if ((rc = DES(desfd,DES_DECRYPT_CBC,&desobj)) < 0)
        {
      printf("destest:  error.\n");
      printf("rc = 0x%x = %d, errno = %d.\n", rc, rc, errno);
      }

    printf("DEST: ");
    print_hex_array(&destination[0][0]);
    print_hex_array(&destination[1][0]);
    print_hex_array(&destination[2][0]);

    printf("CHK:  ");
    print_hex_array(&nowsource[0][0]);
    print_hex_array(&nowsource[1][0]);
    print_hex_array(&nowsource[2][0]);

    printf("IV:   ");
    print_hex_array(&IVarea[0]);

    printf("\n");

    rc = 0;
    for (i=0;i<3;i++)
     for (j=0;j<DES_DataGroupSize;j++)
      rc |= (destination[i][j] != nowsource[i][j]);
    if (rc)
      {
/*    printf("destest:  data error.\n"); */
      }
```

```
    printf("other tests complete.\n");

    DES_Close(desfd);
    }


/*-----------------------------------------------------------------------
*/

int print_hex_array(array)
/*
This function will print the hex representation of the 8 characters
passed
in the array array.
*/

unsigned char *array;

{
int i;

for ( (i = 0) ; (i < 8) ; i++ )
    printf("%X",*(array++));
printf("\n");
}

/*
end of destest.c
*/
```

APPENDIX X. INDEX