หอสมุดกลาง สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

# รายการอ้างอิง

## ภาษาไทย

ซีเอ็ดยูเคชั่น, คู่มือ ไอซี ทีทีแอล , กรุงเทพมหานคร :
บริษัท ซีเอ็ดยูเคชั่น จำกัด

ธานินทร์ ถาวรศาสนวงศ์, ทินกร ดุ๊ก, การอินเทอร์เฟส IBM PC.
กรุงเทพมหานคร : บริษัท ฟิสิกส์ เซ็นเตอร์ การพิมพ์

## ภาษาอังกฤษ

Carl H. Meyer, Stephen M Matyas, Cryptography : A New Dimension in Computer Data Security. John Wiley and Sons, 1982

David P. Anderson, P. Venkat Rangan, High - Performance Interface Architectures For Cryptographic Hardware. in Advances in Cryptology - Eurocrypt87, The Netherlands April 1987 Preceedings

Diffie W., Hellman M., New Directions In Cryptography. IEEE Transactions on Information Theory 22 (Nov 1976):64-65

Dorothy Elizabeth, Robling Denning, Cryptography And Data Security. Addison - Wesley Publishing Company, 1982

Jennifer Seberry, Josef Pieprzyk, Cryptography : An Introduction To Computer Security Prentice Hall of Australia Pty Ltd., 1989

William Caelli, Dennis Longley, Michael Shain, Information Security for Managers. Macmillan Publishers Ltd., 1989

William Caelli, Management and Technical issues in Data Security. Introduction to Crytography : A User's Viewpoint., Information Security Research Center, Queenland University of Technology,1991

# ภาคผนวก ก.

intel®

# 8294
# DATA ENCRYPTION UNIT

- Certified by National Bureau of Standards
- 80 Byte/Sec Data Conversion Rate
- 64-Bit Data Encryption Using 56-Bit Key
- DMA Interface
- 3 Interrupt Outputs to Aid in Loading and Unloading Data
- 7-Bit User Output Port
- Single 5V ± 10% Power Supply
- Peripheral to MCS-86™, MCS-85™, MCS-80™ and MCS-48™ Processors
- Implements Federal Information Processing Data Encryption Standard
- Encrypt and Decrypt Modes Available

The Intel® 8294 Data Encryption Unit (DEU) is a microprocessor peripheral device designed to encrypt and decrypt 64-bit blocks of data using the algorithm specified in the Federal Information Processing Data Encryption Standard. The DEU operates on 64-bit text words using a 56-bit user-specified key to produce 64-bit cipher words. The operation is reversible: if the cipher word is operated upon, the original text word is produced. The algorithm itself is permanently contained in the 8294; however, the 56-bit key is user-defined and may be changed at any time.

The 56-bit key and 64-bit message data are transferred to and from the 8294 in 8-bit bytes by way of the system data bus. A DMA interface and three interrupt outputs are available to minimize software overhead associated with data transfer. Also, by using the DMA interface two or more DEUs may be operated in parallel to achieve effective system conversion rates which are virtually any multiple of 80 bytes/second. The 8294 also has a 7-bit TTL compatible output port for user-specified functions.

Because the 8294 implements the NBS encryption algorithm it can be used in a variety of Electronic Funds Transfer applications as well as other electronic banking and data handling applications where data must be encrypted.

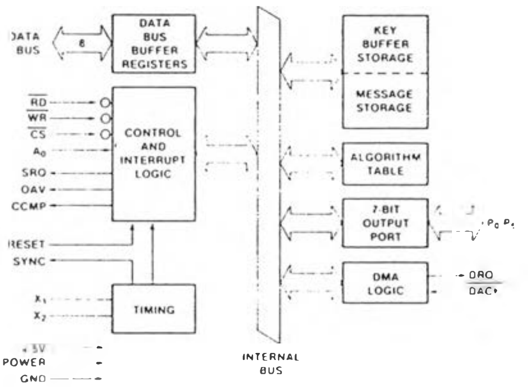


Figure 1. Block Diagram

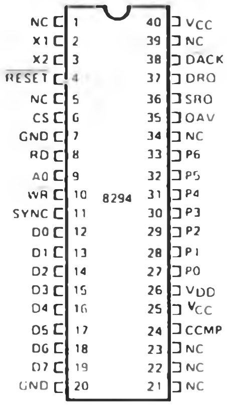


Figure 2. Pin Configuration

## Table 1. Pin Description

| Symbol | Pin No. | Type | Name and Function |
|---|---|---|---|
| NC | 1 | | **No Connection.** |
| X1<br>X2 | 2<br>3 | I | **Crystal:** Inputs for crystal, L-C or external timing signal to determine internal oscillator frequency. |
| $\overline{\text{RESET}}$ | 4 | I | **Reset:** A low signal to this pin resets the 8294. |
| NC | 5 | | **No Connection:** No connection or tied high. |
| $\overline{\text{CS}}$ | 6 | I | **Chip Select:** A low signal to this pin enables reading and writing to the 8294. |
| GND | 7 | | Ground: This pin must be tied to ground. |
| $\overline{\text{RD}}$ | 8 | I | **Read:** An active low read strobe at this pin enables the CPU to read data and status from the internal DEU registers. |
| $A_0$ | 9 | I | **Address:** Address input used by the CPU to select DEU registers during read and write operations. |
| $\overline{\text{WR}}$ | 10 | I | **Write:** An active low write strobe at this pin enables the CPU to send data and commands to the DEU. |
| SYNC | 11 | O | Sync: High frequency (Clock ÷ 15) output. Can be used as a strobe for external circuitry. |
| $D_0$<br>$D_1$<br>$D_2$<br>$D_3$<br>$D_4$<br>$D_5$<br>$D_6$<br>$D_7$ | 12<br>13<br>14<br>15<br>16<br>17<br>18<br>19 | I/O | **Data Bus:** Three-state, bi-directional data bus lines used to transfer data between the CPU and the 8294. |
| GND | 20 | | Ground: This pin must be tied to ground. |
| $V_{CC}$ | 40 | | Power: +5 volt power input: +5V ± 10%. |

| Symbol | Pin No. | Type | Name and Function |
|---|---|---|---|
| NC | 39 | | No Connection. |
| $\overline{\text{DACK}}$ | 38 | I | **DMA Acknowledge:** Input signal from the 8257 DMA Controller acknowledging that the requested DMA cycle has been granted. |
| DRQ | 37 | O | **DMA Request:** Output signal to the 8257 DMA Controller requesting a DMA cycle. |
| SRQ | 38 | O | **Service Request:** Interrupt to the CPU indicating that the 8294 is awaiting data or commands at the input buffer. SRQ=1 implies IBF=0. |
| OAV | 35 | O | **Output Available:** Interrupt to the CPU indicating that the 8294 has data or status available in its output buffer. OAV=1 implies OBF=1. |
| NC | 34 | | **No Connection.** |
| P6<br>P5<br>P4<br>P3<br>P2<br>P1<br>P0 | 33<br>32<br>31<br>30<br>29<br>28<br>27 | O | **Output Port:** User output port lines. Output lines available to the user via a CPU command which can assert selected port lines. These lines have nothing to do with the encryption function. At power-on, each line is in a 1 state. |
| $V_{DD}$ | 26 | | **Power:** +5V power input. (+5V ±10%) Low power standby pin. |
| $V_{CC}$ | 25 | | **Power:** Tied high. |
| CCMP | 24 | O | **Conversion Complete:** Interrupt to the CPU indicating that the encryption/decryption of an 8-byte block is complete. |
| NC | 23 | | **No Connection.** |
| NC | 22 | | **No Connection.** |
| NC | 21 | | **No Connection.** |

## FUNCTIONAL DESCRIPTION

### OPERATION

The data conversion sequence is as follows:

1. A Set Mode command is given, enabling the desired interrupt outputs.
2. An Enter New Key command is issued, followed by 8 data inputs which are retained by the DEU for encryption/decryption. Each byte must have odd parity.
3. An Encrypt Data or Decrypt Data command sets the DEU in the desired mode.

After this, data conversions are made by writing 8 data bytes and then reading back 8 converted data bytes. Any of the above commands may be issued between data conversions to change the basic operation of the DEU; e.g., a Decrypt Data command could be issued to change the DEU from encrypt mode to decrypt mode without changing either the key or the interrupt outputs enabled.

### INTERNAL DEU REGISTERS

Four internal registers are addressable by the master processor: 2 for input, and 2 for output. The following table describes how these registers are accessed.

| $\overline{RD}$ | $\overline{WR}$ | $\overline{CS}$ | $A_0$ | Register |
|---|---|---|---|---|
| 1 | 0 | 0 | 0 | Data input buffer |
| 0 | 1 | 0 | 0 | Data output buffer |
| 1 | 0 | 0 | 1 | Command input buffer |
| 0 | 1 | 0 | 1 | Status output buffer |
| X | X | 1 | X | Don't care |

The functions of each of these registers are described below.

**Data Input Buffer** — Data written to this register is interpreted in one of three ways, depending on the preceding command sequence.

1. Part of a key.
2. Data to be encrypted or decrypted.
3. A DMA block count.

**Data Output Buffer** — Data read from this register is the output of the encryption/decryption operation.

**Command Input Buffer** — Commands to the DEU are written into this register. (See command summary below.)

**Status Output Buffer** — DEU status is available in this register at all times. It is used by the processor for poll-driven command and data transfer operations.

| STATUS BIT: | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
|---|---|---|---|---|---|---|---|---|
| FUNCTION: | X | X | X | KPE | CF | DEC | IBF | OBF |

OBF Output Buffer Full; OBF = 1 indicates that output from the encryption/decryption function is available in the Data Output Buffer. It is reset when the data is read.

IBF Input Buffer Full; A write to the Data Input Buffer or to the Command Input Buffer sets IBF = 1. The DEU resets this flag when it has accepted the input byte. Nothing should be written when IBF = 1.

DEC Decrypt; Indicates whether the DEU is in an encrypt or a decrypt mode. DEC = 1 implies the decrypt mode. DEC = 0 implies the encrypt mode.

CF Completion Flag; This flag may be used to indicate any or all of three events in the data transfer protocol.

1. It may be used in lieu of a counter in the processor routine to flag the end of an 8-byte transfer.
2. It must be used to indicate the validity of the KPE flag.
3. It may be used in lieu of the CCMP interrupt to indicate the completion of a DMA operation.

KPE Key Parity Error; After a new key has been entered, the DEU uses this flag in conjunction with the CF flag to indicate correct or incorrect parity.

### COMMAND SUMMARY

**1 — Enter New Key**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| OP CODE: | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| | MSB | | | | | | | LSB |

This command is followed by 8 data byte inputs which are retained in the key buffer (RAM) to be used in encrypting and decrypting data. These data bytes must have odd parity represented by the LSB.

**2 — Encrypt Data**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| OP CODE: | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| | MSB | | | | | | | LSB |

This command puts the 8294 into the encrypt mode.

**3 — Decrypt Data**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| OP CODE: | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| | MSB | | | | | | | LSB |

This command puts the 8294 into the decrypt mode.

**4 — Set Mode**

| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| OP CODE: | 0 | 0 | 0 | 0 | A | B | C | D |
| | MSB | | | | | | | LSB |

where:

A is the OAV (Output Available) interrupt enable
B is the SRQ (Service Request) interrupt enable
C is the DMA (Direct Memory Access) transfer enable
D is the CCMP (Conversion Complete) interrupt enable

This command determines which interrupt outputs will be enabled. A "1" in bits A, B, or D will enable the OAV, SRQ, or CCMP interrupts respectively. A "1" in bit C will allow DMA transfers. When bit C is set the OAV and SRQ interrupts should also be enabled (bits A,B = 1). Following the command in which bit C, the DMA bit, is set, the 8294 will expect one data byte to specify the number of 8-byte blocks to be converted using DMA.

5 — **Write to Output Port**

| OP CODE: | 1 | $P_6$ | $P_5$ | $P_4$ | $P_3$ | $P_2$ | $P_1$ | $P_0$ |
|---|---|---|---|---|---|---|---|---|
| | MSB | | | | | | | LSB |

This command causes the 7 least significant bits of the command byte to be latched as output data on the 8294 output port. The initial output data is 1111111. Use of this port is independent of the encryption/decryption function.

## PROCESSOR/DEU INTERFACE PROTOCOL

### ENTERING A NEW KEY

The timing sequence for entering a new key is shown in Figure 3. A flowchart showing the CPU software to accommodate this sequence is given in Figure 4.

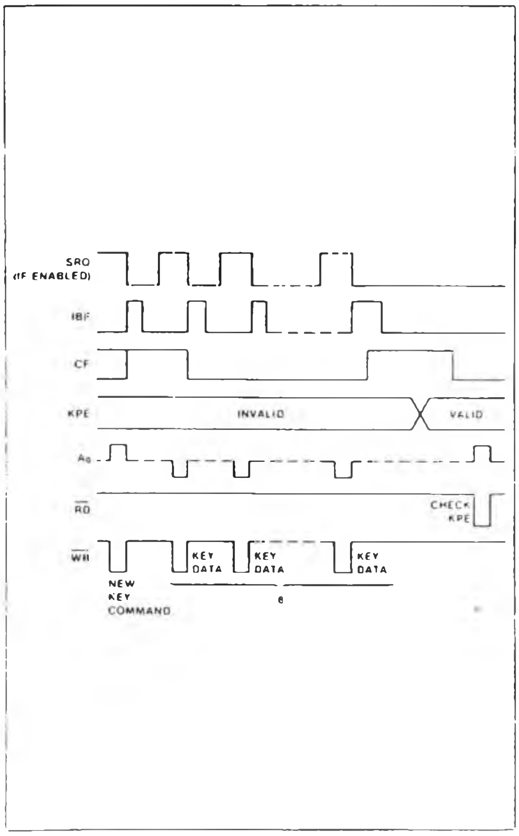


Figure 3. Entering a New Key

After the Enter New Key command is issued, 8 data bytes representing the new key are written to the data input buffer (most significant byte first). After the eighth byte is accepted by the DEU, CF goes true (CF = 1). The CF bit goes false again when KPE is valid. The CPU can then check the KPE flag. If KPE = 1, a parity error has been detected and the DEU has not accepted the key. Each byte is checked for odd parity, where the parity bit is the LSB of each byte.

Since the CF bit is used in this protocol to indicate the validity of the KPE flag, it may not be used to flag the end of the 8 byte key entry. CF = 1 only as long as KPE is invalid. Therefore, the CPU might not detect that CF = 1 and the key entry is complete before KPE becomes valid. Thus, a counter should be used, as in Figure 4, to flag the end of the new key entry. Then, CF is used to indicate a valid KPE flag.

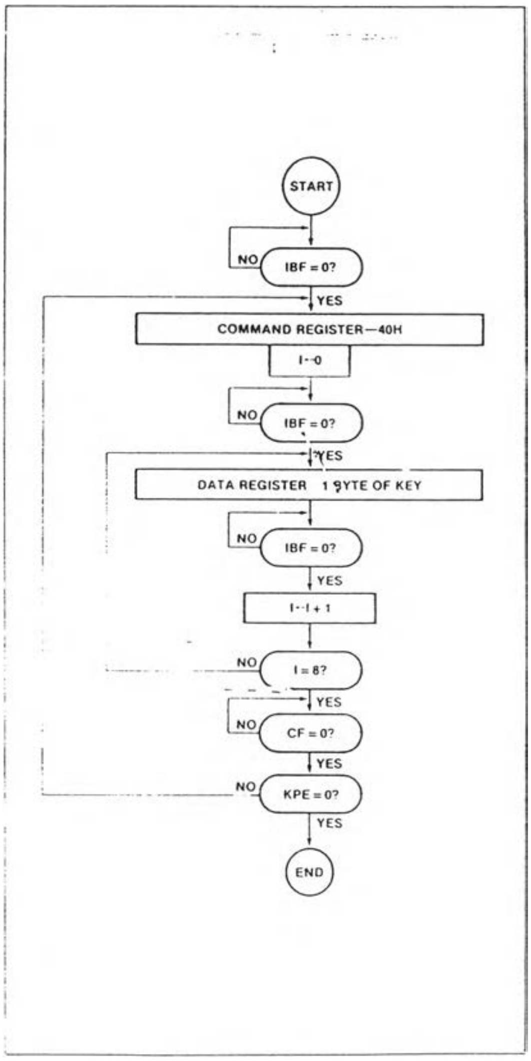


Figure 4. Flowchart for Entering a New Key

## ENCRYPTING OR DECRYPTING DATA

Figure 5 shows the timing sequence for encrypting or decrypting data. The CPU writes 8 data bytes to the DEU's data input buffer for encryption/decryption. CF then goes true (CF = 1) to indicate that the DEU has accepted the 8-byte block. Thus, the CPU may test for IBF = 0 and CF = 1 to terminate the input mode, or it may use a software counter. When the encryption/decryption is complete, the CCMP and OAV interrupts are asserted and the OBF flag is set true (OBF = 1). OAV and OBF are set false again after each of the converted data bytes is read back by the CPU. The CCMP interrupt is set false, and remains false, after the first read. After 8 bytes have been read back by the CPU, CF goes false (CF = 0). Thus, the CPU may test for CF = 0 to terminate the read mode. Also, the CCMP interrupt may be used to initiate a service routine which performs the next series of 8 data reads and 8 data writes.

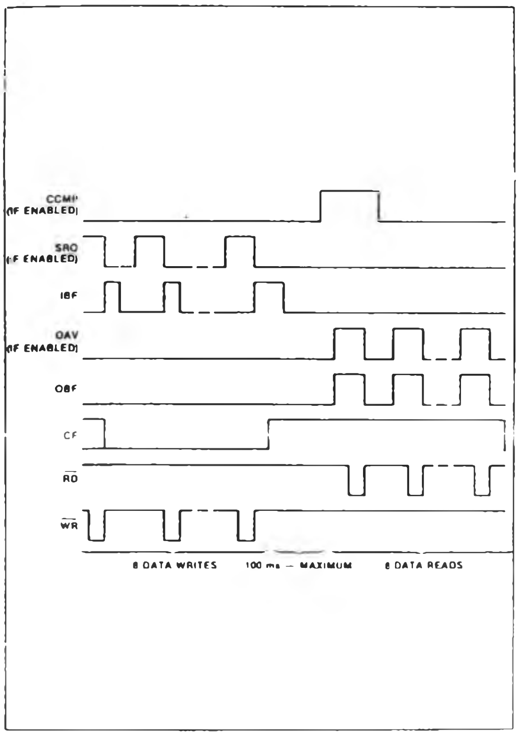


Figure 5. Encrypting/Decrypting Data

Figure 6 offers two flowcharts outlining the alternative means of implementing the data conversion protocol. Either the CF flag or a software counter may be used to end the read and write modes.

SRQ = 1 implies IBF = 0, OAV = 1 implies OBF = 1. This allows interrupt routines to do data transfers without checking status first. However, the OAV service routine must detect and flag the end of a data conversion.

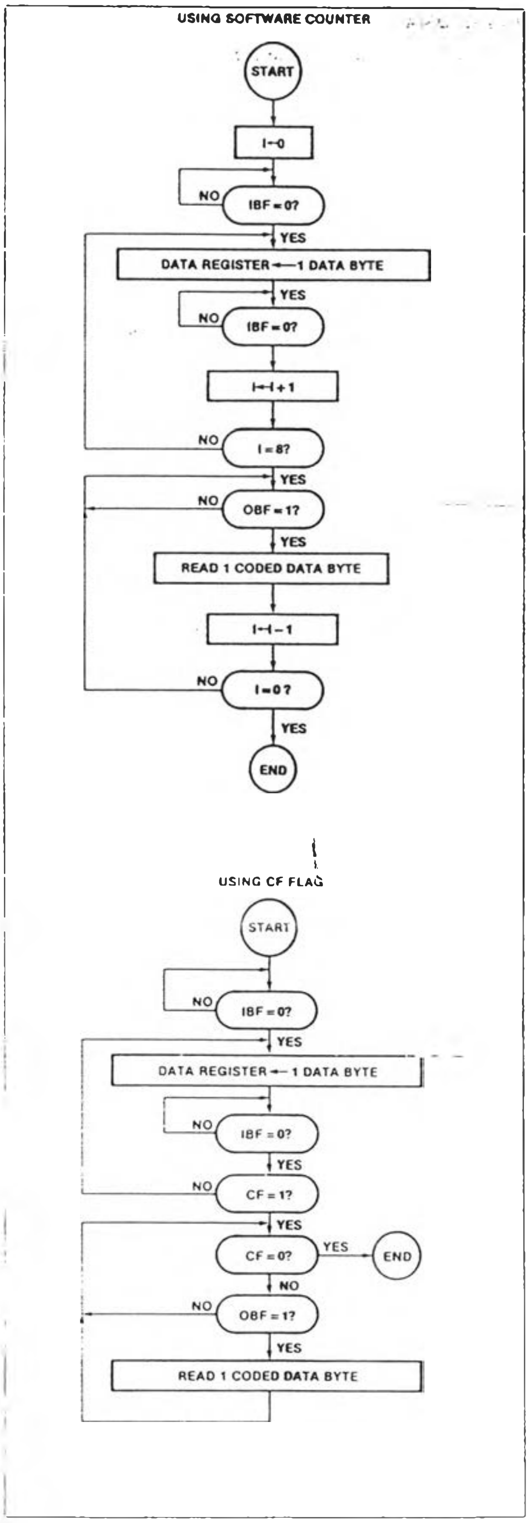


Figure 6. Data Conversion Flowcharts

## USING DMA

The timing sequence for data conversions using DMA is shown in Figure 7. This sequence can be better understood when considered in conjunction with the hardware DMA interface in Figure 8. Note that the use of the DMA feature requires 3 external AND gates and 2 DMA channels (one.for input, one for output). Since the DEU has only one DMA request pin, the SRQ and OAV outputs are used in conjunction with two of the AND gates to create separate DMA request outputs for the 2 DMA channels. The third AND gate combines the two active-low $\overline{DACK}$ inputs.

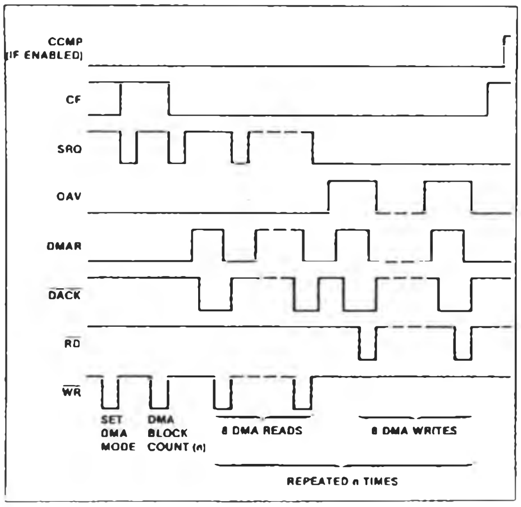


Figure 7. DMA Sequence

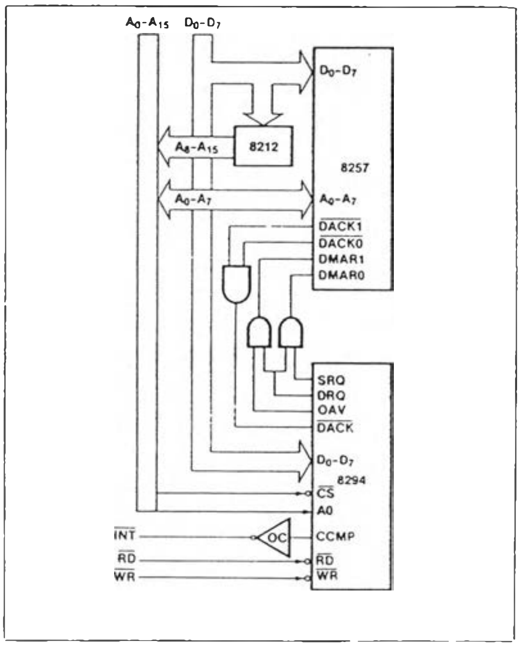


Figure 8. DMA Interface

To initiate a DMA transfer, the CPU must first initialize the two DMA channels as shown in the flowchart in Figure 9. It must then issue a Set Mode command to the DEU enabling the OAV, SRQ, and DMA outputs. The CCMP interrupt may be enabled or disabled, depending on whether that output is desired. Following the Set Mode command, there must be a data byte giving the number of 8-byte blocks of data (n<256) to be converted. The DEU then generates the required number of DMA requests to the 2 DMA channels with no further CPU intervention. When the requested number of blocks has been converted, the DEU will set CF and assert the CCMP interrupt (if enabled). CCMP then goes false again with the next write to the DEU (command or data). Upon completion of the conversion, the DMA mode is disabled and the DEU returns to the encrypt/decrypt mode. The enabled interrupt outputs, however, will remain enabled until another Set Mode command is issued.

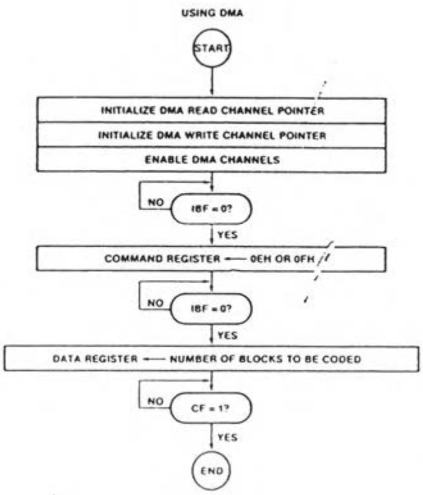

USING DMA
START
INITIALIZE DMA READ CHANNEL POINTER
INITIALIZE DMA WRITE CHANNEL POINTER
ENABLE DMA CHANNELS
NO
IBF = 0?
YES
COMMAND REGISTER ← 0EH OR 0FH
NO
IBF = 0?
YES
DATA REGISTER ← NUMBER OF BLOCKS TO BE CODED
NO
CF = 1?
YES
END

Figure 9. DMA Flowchart

## SINGLE BYTE COMMANDS

Figure 10 shows the timing and protocol for single byte commands. Note that any of the commands is effective as a pacify command in that they may be entered at any time, except during a DMA conversion. The DEU is thus set to a known state. However, if a command is issued out of sequence, an additional protocol is required (Figure 11). The CPU must wait until the command is accepted (IBF = 0). A data read must then be issued to clear anything the preceding command sequence may have left in the Data Output Buffer.

## CPU/DEU INTERFACES

Figures 12 through 15 illustrate four interface configurations used in the CPU/DEU data transfers. In all cases SRQ will be true (if enabled) and IBF will be false when the DEU is ready to accept data or commands.

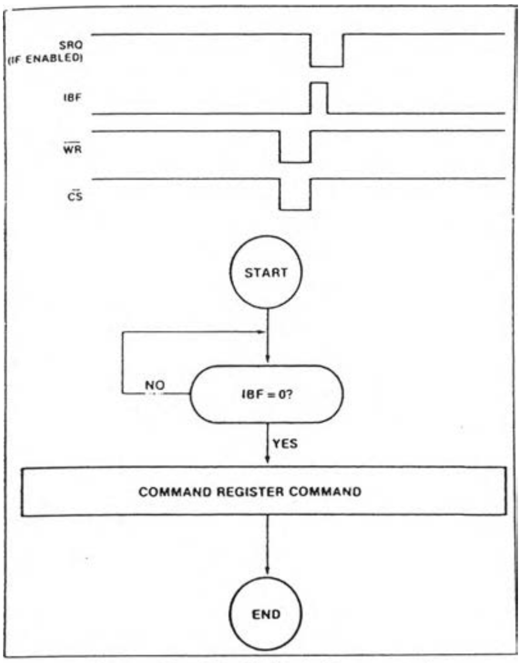


Figure 10. Single Byte Commands

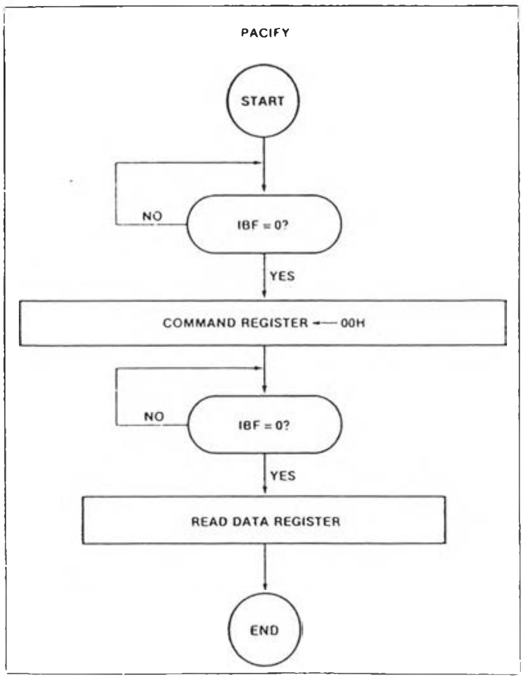


Figure 11. Pacify Protocol

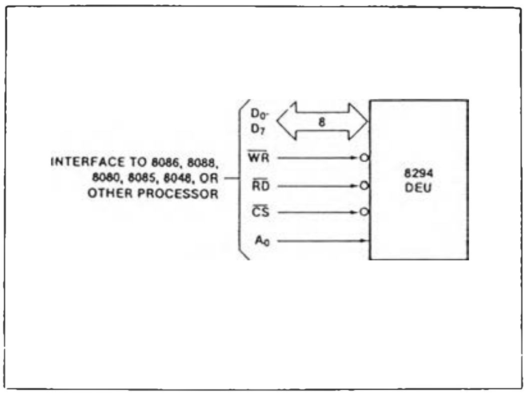


Figure 12. Polling Interface

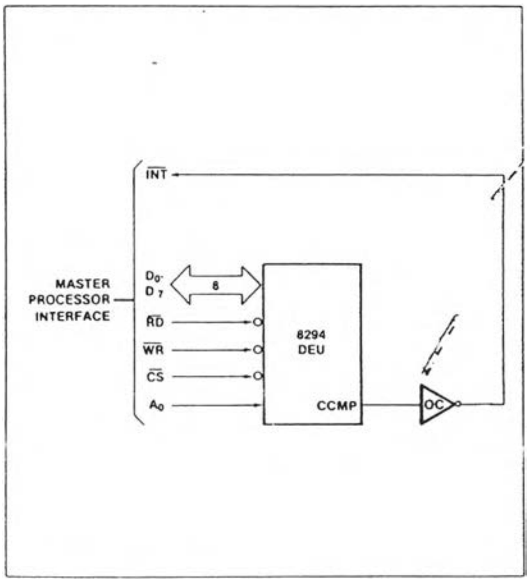


Figure 13. Single Interrupt Interface

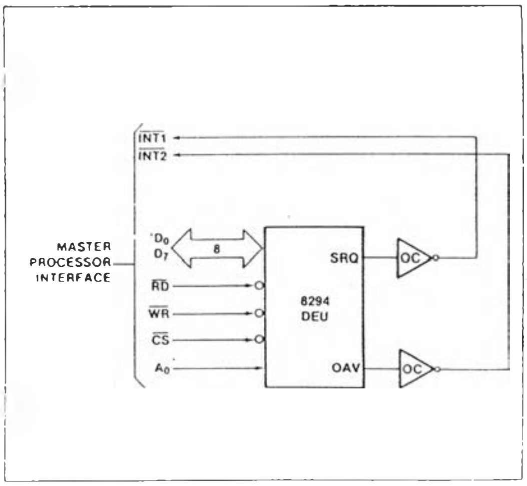


Figure 14. Dual Interrupt Interface

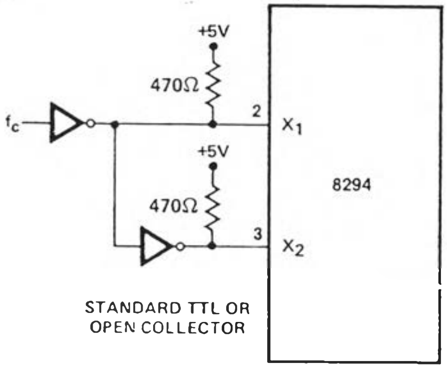


Figure 18. Recommended Connection for External Clock Signal

## ABSOLUTE MAXIMUM RATINGS*

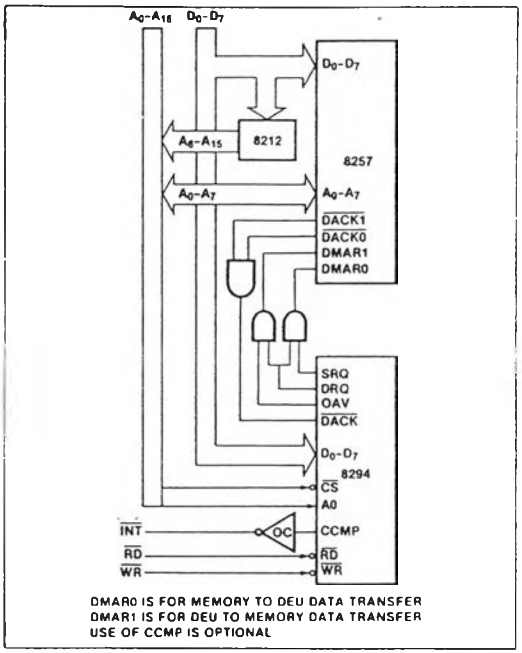
| | |
|---|---|
| Ambient Temperature Under Bias | 0°C to 70°C |
| Storage Temperature | −65°C to +150°C |
| Voltage on Any Pin With Respect to Ground | −0.5V to +7V |
| Power Dissipation | 1.5 Watt |

**NOTICE: Stresses above those listed under "Absolute Maximum Ratings" may cause permanent damage to the device. This is a stress rating only and functional operation of the device at these or any other conditions above those indicated in the operational sections of this specification is not implied. Exposure to absolute maximum rating conditions for extended periods may affect device reliability.*

## D.C. AND OPERATING CHARACTERISTICS ($T_A$ = 0°C to 70°C, $V_{CC}$ = +5V ± 10%, $V_{SS}$ = 0V)

| Symbol | Parameter | Limits | | | Unit | Test Conditions |
|---|---|---|---|---|---|---|
| | | Min. | Typ. | Max. | | |
| $V_{IL}$ | Input Low Voltage (All Except $X_1$, $X_2$, $\overline{RESET}$) | −0.5 | | 0.8 | V | |
| $V_{IL1}$ | Input Low Voltage ($X_1$, $X_2$, $\overline{RESET}$) | −0.5 | | 0.6 | V | |
| $V_{IH}$ | Input High Voltage (All Except $X_1$, $X_2$, $\overline{RESET}$) | 2.2 | | $V_{CC}$ | V | |
| $V_{IH1}$ | Input High Voltage ($X_1$, $X_2$, $\overline{RESET}$) | 3.8 | | $V_{CC}$ | V | |
| $V_{OL}$ | Output Low Voltage ($D_0$-$D_7$) | | | 0.45 | V | $I_{OL}$ = 2.0 mA |
| $V_{OL1}$ | Output Low Voltage (All Other Outputs) | | | 0.45 | V | $I_{OL}$ = 1.6 mA |
| $V_{OH}$ | Output High Voltage ($D_0$-$D_7$) | 2.4 | | | V | $I_{OH}$ = −400 µA |
| $V_{OH1}$ | Output High Voltage (All Other Outputs) | 2.4 | | | V | $I_{OH}$ = −50 µA |
| $I_{IL}$ | Input Leakage Current ($\overline{RD}$, $\overline{WR}$, $\overline{CS}$, $A_0$) | | | ±10 | µA | $V_{SS} \leq V_{IN} \leq V_{CC}$ |
| $I_{OZ}$ | Output Leakage Current ($D_0$-$D_7$, High Z State) | | | ±10 | µA | $V_{SS}$ +0.45 $\leq V_{OUT} \leq V_{CC}$ |
| $I_{DD}$ | $V_{DD}$ Supply Current | | 5 | 15 | mA | |
| $I_{DD}$ + $I_{CC}$ | Total Supply Current | | 60 | 125 | mA | |
| $I_{LI}$ | Low Input Load Current (Pins 24, 27-38) | | | 0.5 | mA | $V_{IL}$ = 0.8 V |
| $I_{LI1}$ | Low Input Load Current ($\overline{RESET}$) | | | 0.2 | mA | $V_{IL}$ = 0.8 V |
| $I_{IH}$ | Input High Leakage Current (Pins 24, 27-38) | | | 100 | µA | $V_{IN}$ = $V_{CC}$ |
| $C_{IN}$ | Input Capacitance | | | 10 | pF | |
| $C_{I/O}$ | I/O Capacitance | | | 20 | pF | |

หอสมุดกลาง สำนักงานวิทยทรัพยากร จุฬาลงกรณ์มหาวิทยาลัย



DMAR0 IS FOR MEMORY TO DEU DATA TRANSFER
DMAR1 IS FOR DEU TO MEMORY DATA TRANSFER
USE OF CCMP IS OPTIONAL

Figure 15. DMA Interface

## OSCILLATOR AND TIMING CIRCUITS

The 8294's internal timing generation is controlled by a self-contained oscillator and timing circuit. A choice of crystal, L-C or external clock can be used to derive the basic oscillator frequency.

The resident timing circuit consists of an oscillator, a state counter and a cycle counter as illustrated in Figure 16.

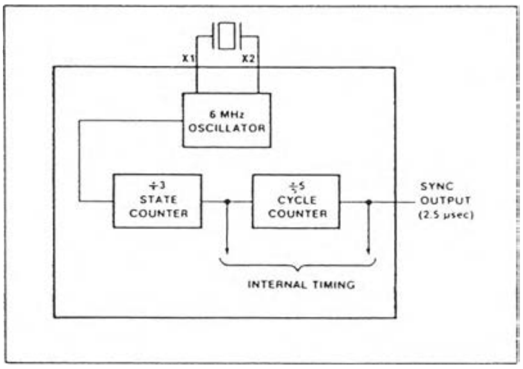


Figure 16. Oscillator Configuration

### OSCILLATOR

The on-board oscillator is a series resonant circuit with a frequency range of 1 to 6 MHz. Pins X1 and X2 are input and output (respectively) of a high gain amplifier stage. A crystal or inductor and capacitator connected between X1 and X2 provide the feedback and proper phase shift for oscillation. Recommended connections for crystal or L-C are shown in Figure 17

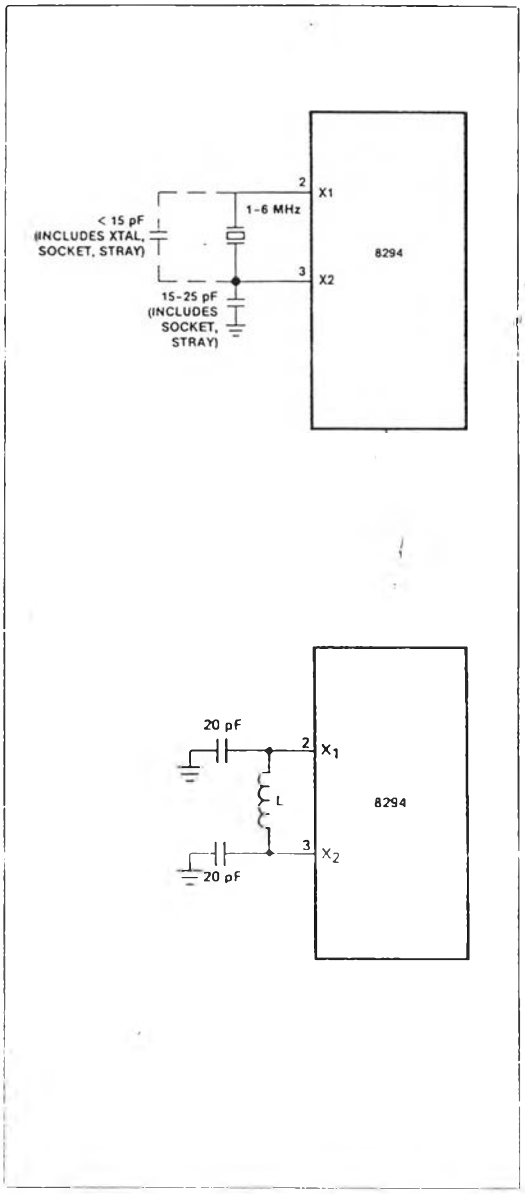


Figure 17. Recommended Crystal and L-C Connections

A recommended range of inductance and capacitance combinations is given below:

L = 120μH corresponds to 3MHz
L = 45μH corresponds to 5MHz

An external clock signal can also be used as a frequency reference to the 8294; however, the levels are *not* compatible. The signal must be in the 1MHz-6MHz frequency range and must be connected to pins X1 and X2 by buffers with a suitable pull-up resistor to guarantee that a logic "1" is above 3.8 volts. The recommended connection is shown in Figure 18.

## A.C. CHARACTERISTICS ($T_A$ = 0°C to 70°C, $V_{CC}$ = $V_{DD}$ = +5V ± 10%, $V_{SS}$ = 0V)

### DBB READ

| Symbol | Parameter | Min. | Max. | Unit | Test Conditions |
|---|---|---|---|---|---|
| $t_{AR}$ | $\overline{CS}$, $A_0$ Setup to $\overline{RD}$ ↓ | 0 | | ns | |
| $t_{RA}$ | $\overline{CS}$, $A_0$ Hold After $\overline{RD}$ ↑ | 0 | | ns | |
| $t_{RR}$ | $\overline{RD}$ Pulse Width | 250 | | ns | |
| $t_{AD}$ | $\overline{CS}$, $A_0$ to Data Out Delay | | 225 | ns | $C_L$ = 150 pF |
| $t_{RD}$ | $\overline{RD}$ ↓ to Data Out Delay | | 225 | ns | $C_L$ = 150 pF |
| $t_{DF}$ | $\overline{RD}$ ↑ to Data Float Delay | | 100 | ns | |
| $t_{CY}$ | Cycle Time | 2.5 | 15 | μs | 6 MHz Crystal |

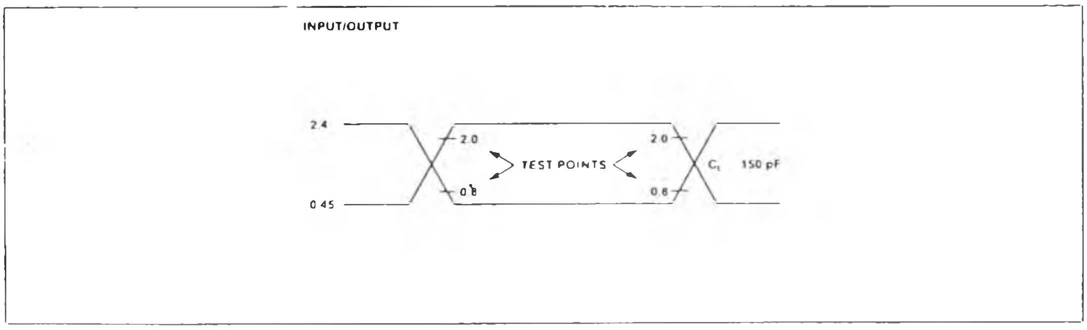
### DBB WRITE

| Symbol | Parameter | Min. | Max. | Unit | Test Conditions |
|---|---|---|---|---|---|
| $t_{AW}$ | $\overline{CS}$, $A_0$ Setup to $\overline{WR}$ ↓ | 0 | | ns | |
| $t_{WA}$ | $\overline{CS}$, $A_0$ Hold After $\overline{WR}$ ↑ | 0 | | ns | |
| $t_{WW}$ | $\overline{WR}$ Pulse Width | 250 | | ns | |
| $t_{DW}$ | Data Setup to $\overline{WR}$ ↑ | 150 | | ns | |
| $t_{WD}$ | Data Hold to $\overline{WR}$ ↑ | 0 | | ns | |

### DMA AND INTERRUPT TIMING

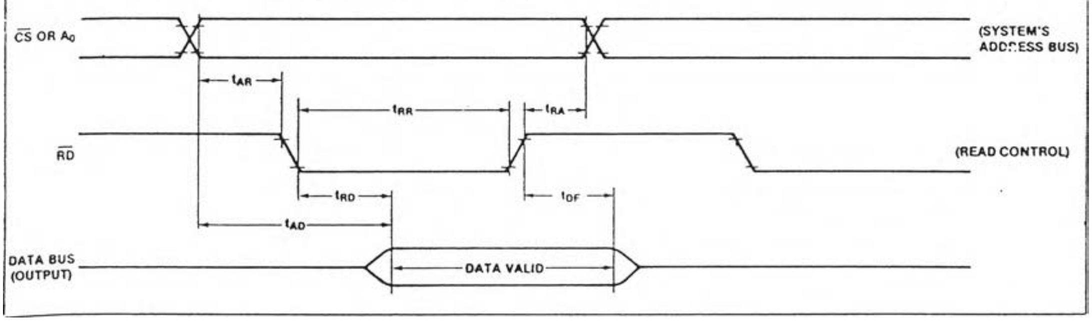
| Symbol | Parameter | Min. | Max. | Unit | Test Conditions |
|---|---|---|---|---|---|
| $t_{ACC}$ | $\overline{DACK}$ Setup to Control | 0 | | ns | |
| $t_{CAC}$ | $\overline{DACK}$ Hold After Control | 0 | | ns | |
| $t_{ACD}$ | $\overline{DACK}$ to Data Valid | | 225 | ns | $C_L$ = 150 pF |
| $t_{CRQ}$ | Control L.E. to DRQ T.E. | | 200 | ns | |
| $t_{CI}$ | Control T.E. to Interrupt T.E. | | $t_{CY}$ + 500 | ns | |

### A.C. TESTING INPUT, OUTPUT WAVEFORM

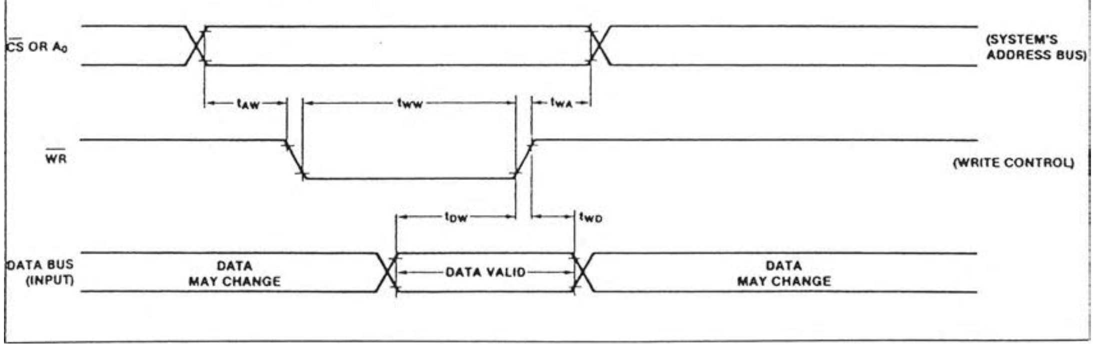


## WAVEFORMS

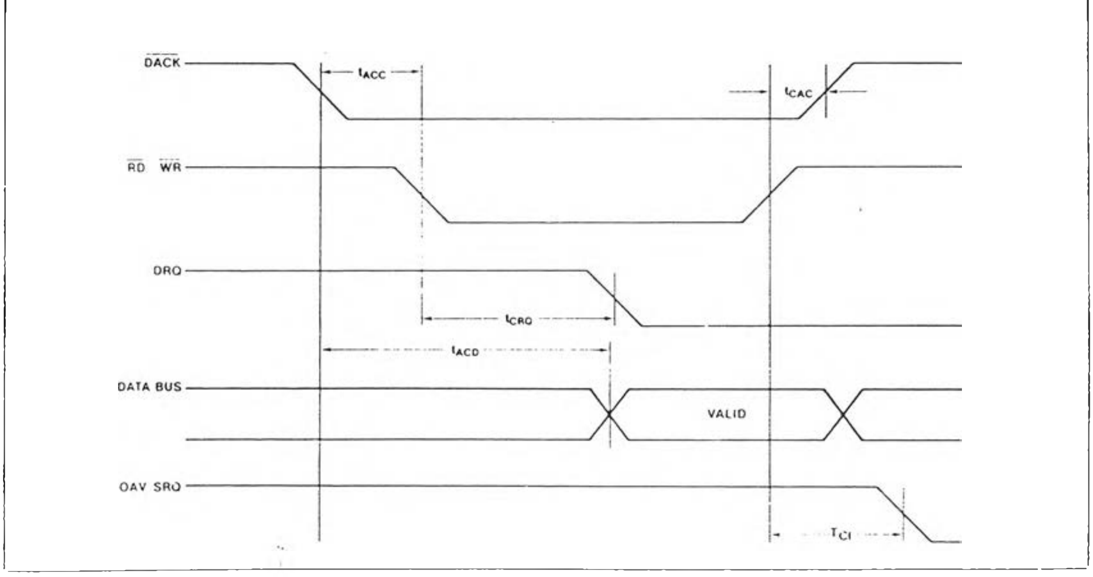
### READ OPERATION—OUTPUT BUFFER REGISTER



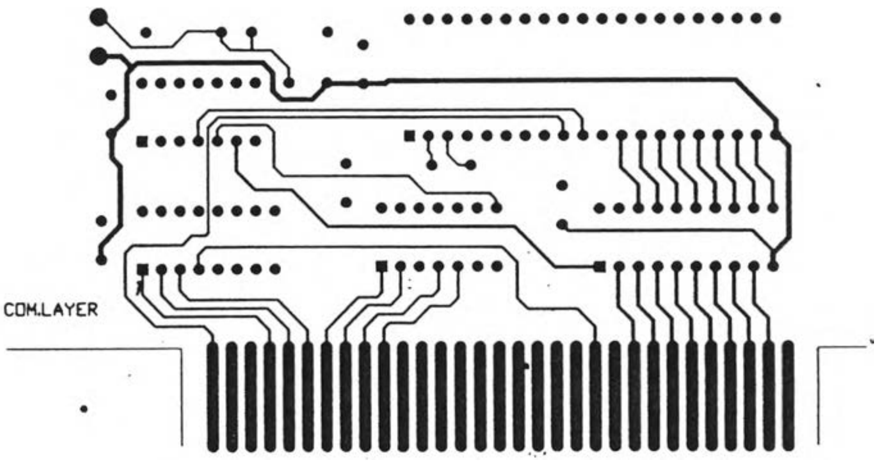
### WRITE OPERATION—INPUT BUFFER REGISTER

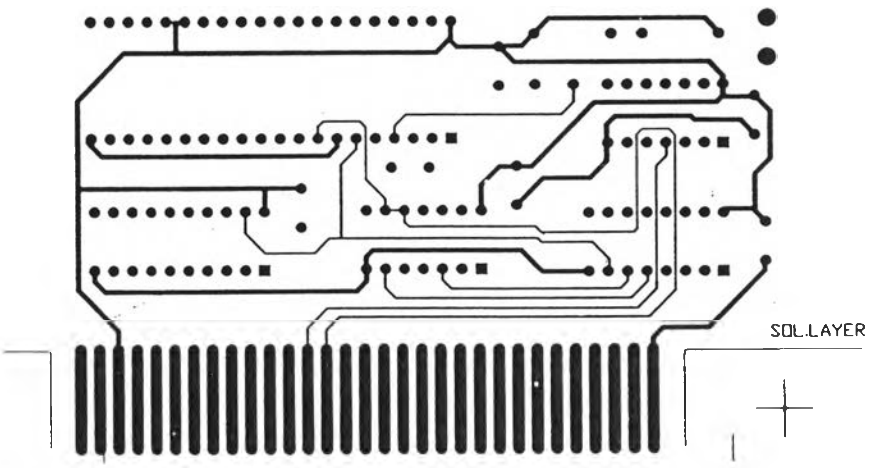


### DMA AND INTERRUPT TIMING



# ภาคผนวก ข.


COM.LAYER


SOL.LAYER

# ภาคผนวก ค.

```
#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <stdlib.h>
#include <graphics.h>
#include <dir.h>
#include <alloc.h>
#include <stdarg.h>
#include <bios.h>
#include <time.h>
#include <io.h>
#include <fcntl.h>
#include <c:\tc\include\sys\stat.h>
#include <c:\tc\include\menukey.h>
#include <c:\tc\c_expres\prototype\turbo_c\printer.h>

#define RESET          0x330
#define DATAINPORT     0x338
#define DATAOUTPORT    0x33A
#define COMMANDPORT    0x339
#define STATUSPORT     0x33B

/****************
 GLOBAL VARIABLE
 ****************/

char *plainfile,*cipherfile,*secretkey,*vector,*ofb_fback;
FILE *ff1,*ff2,*ff3,*ff4;
char string[10][20];
int  filesize[4];
float used_time[4];
int operate_mode,abbr_mode,Key_digit,IV_digit,num_fb;

FILE *IN, *OUT;
int i, last;
float ecb_time,cbc_entime,cfb_time,ofb_time;
char plain,cipher,inFile[15],outFile[15],op[5];

/*  Setting charactor font of printer */
char memory_model = 2;
char time_out = 3;
char error_code;

#define EJECT fprintf(printer,"%c",12)
#define LINE  "========================================"
static  char         *name;
static  char    *head;
static     char       *mode;
static  int     page=1;
struct  time          timep;
struct  date    datep;
FILE     *file1, *printer;


int number_c=0;                      /* number of curve */
int out_menu;                 /* integer value for out of file "menu" */
int static_air=0;            /* determine static air */
int spp[]={10,10,10,10,10,10};
int vi[]={10,10,10,10,10,10};
int maxx,maxy;
int NKey;
void far *rebar[]={0,0,0,0,0,0,0,0,0,0,0,0,0,0};
char *c[6];                  /* expand massage */
int act;                    /* output action control-key */

/********************** MAIN ********************************/


 void main (void)
 {
  int i=0;
  out_menu=0;
  for(i=0;i<6;i++)
  rebar[i]=0;
 SetGraph();
 while(1)
 {
  DispMenu();
  Menu();
  if(out_menu!=0)
   break;
 }
 closegraph();
 return(out_menu);
}
/***************************************************************/

int getkey(void)
{
  int c;
  while (bioskey(1)==0)
   geninterrupt(0x28);
  if (((c=bioskey(0))&255)==0)
    c=(c>>8)|0x80;
  return c&255;
 }

 void SetGraph(void)
 {
  int driver,mode;
  setgraphbufsize(76800);
  detectgraph(&driver,&mode);
  initgraph(&driver,&mode," ");
  maxx=getmaxx();
  maxy=getmaxy();
  setbkcolor(CYAN); /*CAN BE A NAME OR NUMBER TO SELECT THE COLOR*/
}

void write_menutext(int sx,int sy,int ex,int ey)
{
/*DO NOT USE ONLY 'write'  TO BE A FUNCTION NAME BECAUSE IT DUPLICATE TO
 *A NAME USE IN THE C STANDARD DECLARATION */
   void *lab;
   lab=malloc(imagesize(sx,sy,ex,ey));
   getimage(sx,sy,ex,ey,lab);
   setfillstyle(1,7);
   setcolor(7);
   bar3d(sx,sy,ex,ey,0,0);
   putimage(sx,sy,lab,XOR_PUT);
   free(lab);

}

void Saveview(int sx,int sy,int ex,int ey,int re_win)
{
   int size; /*unsigned long size;*/
   size=imagesize(sx,sy,ex,ey);
   rebar[re_win] = malloc(size);/*imagesize(sx,sy,ex,ey)); */
   if(rebar[re_win]==NULL)
    {
     closegraph();
          printf("\nFailed allocate memory");
          printf("\n imagesize=%d",size);
          exit(1);
    }
   getimage(sx,sy,ex,ey,rebar[re_win]);
}

void Restoreview(int sx,int sy,int re_win)
{
   putimage(sx,sy,rebar[re_win],COPY_PUT);
   free(rebar[re_win]);
   rebar[re_win]=0;
}

void DispMenu(void)
{
 settextstyle(DEFAULT_FONT,HORIZ_DIR,1);
 setcolor(15);
 setfillstyie(1,15);
 bar3d(0,0,maxx.maxy*.03125,0,0);
 bar3d(0,maxy*.96667,maxx,maxy,0,0);
 setcolor(8);
 settextjustify(LEFT_TEXT,TOP_TEXT);
 outtextxy(maxx*.03125,maxy*.010416,"Operation");
 outtextxy(112,maxy*.010416,"View");
 outtextxy(174,maxy*.010416,"Print");
 outtextxy(232,maxy*.010416,"!Quit");
 outtextxy(20,470,"  F1 Help    F2 Save    F3 Load    ALT-X Quit    F10 Menu");
}

MAIN_PRINT()
{
   int i;
   int column_menu = 3;
   char *str[]={"Plainfile :","Cipherfile:"};
   static int air=0;
   int count=2;
   int re_menu=3;
   int num_help = 3;
   act=0;
   c[0]="Make a hardcopy of a plainfile";
   c[1]="Make a hardcopy of a cipherfile";
   window_y1(120,0,270,86,0);          /* determine sguare-point of window */

   for(i=0;i<2;i++)
      outtextxy(120+25,20*i+30,str[i]);
   show_under(0);
   re_menu=control_key(air,count,120,270,2,0,column_menu,num_help);
   air=static_air;
   if(act==1)
    {
    switch(air)
    {
          case (0): {
                     PRINT_PLAIN();
                        outtextxy(500,maxy*.010416,"F2=Toggle word");
                      PRINTC(air);
                      Restoreview(120,0,0);
                      } break;
          case (1): {
                     PRINT_PLAIN();
                        outtextxy(500,maxy*.010416,"F2=Toggle word");
                      PRINTC(air);
                      Restoreview(120,0,0);
                      } break;
    }
   }
```

```
return 0;
}

PRINT_PLAIN()

{
int amount_para,line_para,Esc_From_Para;  /*=3 or 4 if not has IV, =5 if has */
 if(abbr_mode==0) amount_para = 1; /*ECB MODE */
  setcolor(15);    /*clear header F1 */
  outtextxy(350,maxy*.010416,"F1=Toggle Help");
  window_y1(200,188,416,265,7);

  for(line_para=1;line_para<=amount_para;line_para++)
   {
    Esc_From_Para = greadfloat(228,220,20,line_para,0,0);
    if(Esc_From_Para == 5).
     {
      line_para = amount_para;
     }
  }
}


void heading()
{
        if (head[1]=='n' || mode[1]=='n')
          fprintf(printer,"\t\t\t   % 4d\n",page);
        else{
          fprintf(printer,"File : %-20s   Date : %02d/%02d/%d     ",
                  plainfile,datep.da_day,datep.da_mon,datep.da_year);
          fprintf(printer,"Time : %2d:%02d     Page : %4d\n",
                  timep.ti_hour,timep.ti_min,page);
          fprintf(printer,"%s%s\n",LINE,LINE);
        }
}
PRINTC(int filetype)
{
        int         c, w_row=0;
        int column = 1;
        file1=fopen(plainfile,"rb");
                if(file1== NULL)
                 {
                  ERROR(210,110,405,190,5,7);
                  Restoreview(200,188,7); /*restore the file to be printed view*/
                  return(1);
                 }
       printer=fopen("prn","wb");
          if(!printer_ready()) /*function in c_express*/
           {
                  ERROR(210,110,405,190,5,8);
              Restoreview(200,188,7); /*restore the file to be printed view*/
                  return(1);
           }
          Restoreview(200,188,7);
        gettime(&timep);
        getdate(&datep);
        if (filetype==0) /*plain file */
          fprintf(printer,"%c%c%c%c%c%c%c%c%c%c%c",
                                  27,77,27,120,1,27,51,50,27,108,10);
        else /* cipher file*/
          fprintf(printer,"%c%c%c%c%c%c%c%c%c%c%c%c%c%c%c",
                                  27,64,27,54,27,120,1,27,77,27,51,50,27,108,10);

        /*27,120,0 for draft mode 27,120,1 for nlq mode */

        heading();
        while((c=getc(file1)) != EOF/*feof(file1)*/)
        {
            if (c == '\n')
            {
                 if (w_row > 4000)
                 {
                   EJECT;
                   w_row = 0;
                   page++;
                  heading();
                 } else w_row++;
            }

        if(filetype==0)
            fprintf(printer,"%c",c);
        else
           {
             if(c==27 || (c>=0 && c <=9) || (c>=10 && c <=31))
                   fprintf(printer,".");
             else  fprintf(printer,"%c",c);
 if(column == 80 ){c='\n'; fprintf(printer,"%c",c);column =0;}
 column++;
           }

 }

 if(filetype == 0)
  fprintf(printer,"%s%s\n",LINE,LINE);
 else
  fprintf(printer,"%c%c%c%s%s\n",'\n','\n','\n',LINE,LINE);

           fprintf(printer,"\t\t         END OF File : %-20s ",plainfile);
         EJECT;
         fclose(file1);
         fclose(printer);
}
Quit()          /* the Quit menu */
{
  int i;
  int column_menu = 4;
  int num_help = 4;
  char *str[]={"EXIT TO DOS"};
  static int air=0;
  int count=1;
  int re_menu=3;
  act=0;
  out_menu=0;
 while(1)
 {
  window_y1(179,0,330,65,0);          /* determine square-point of window */
  c[0]="Finish this program and exit to DOS prompt";
  for(i=0;i<count;i++)
     outtextxy(179+25,20*i+30,str[i]);
     setcolor(15);
   outtextxy(350,maxy*.010416,"F1=Toggle Help");
   getcolor();
  show_under(1);
  re_menu=control_key(air,count,179,330,3,0,column_menu,num_help);
  air=static_air;

  if(act==1)
  {
   switch(air)
   {
    case 0:closegraph();  break;;
              if(NKey=='\r')
               return 0;
              break;
   }
  }
  if(re_menu!=3)  /* for out of menu */
   return(re_menu);
  Restoreview(179,0,0);
  }
}
encryption()
{
  int i;
  char *str[]={ "ELECTRONIC CODEBOOK(ECB)",
                      "CIPHER BLOCK CHAINING(CBC)",
                      "CIPHER FEEDBACK(CFB)",
                      "OUTPUT FEEDBACK(OFB)",
                      "FILE REGISTRATION",
                      "MESSAGE AUTHENTICATION" };
  static int air=0;
  int column_menu = 11;
  int num_help = 11;
  int count=6;
  int re_menu = 3;
  act=0;
  c[0]="This mode is easy to attack";
  c[1]="Use the previous cipher block to generate a present block";
  c[2]="Feedback the cipher block to generate a present block";
  c[3]="Feedback the output to generate the present block";
  c[4]="Register the file by padding with the authentication code";
  c[5]="Authenticate the content of messages";
  while(1){
 window_y1(180,180,430,385,1); /*(300,30,475,195,1);*/
 /* determine square-point of window */
   algorithm_label();  /* draw the menu 'ALGORITHMS ' */
  setcolor(8);
  for(i=0;i<count;i++)
     outtextxy(198,20*i+245,str[i]);  /*(300+25,20*i+60,str[i]);*/
  show_under(3);
 re_menu=control_key_encrypt(air,count,175,434,215,42,1,column_menu,num_help);
  air=static_air;
  if(act==1)
  {
   if((air>=0) || (air<=5)) /*this line must change with num of line in*/
    {  abbr_mode = air;      /* abbr_mode */
         PARAMETER();
    }
  /* return 0;*/
   }
if(re_menu !=6) return(re_menu); /* corresponding to the above comment */
Restoreview(0,0,0);
}
}
PARAMETER( /* ENC or DEC */ /*ECB CBC CFB or OFB */)
{
 /* FILE *ff1,*ff2,*ff3,*ff4;*/
  int i,handle;
  float flength;
  char *str[]={ "Inputfile:.....",
                         "Outputfile:.....",
                         "Key:.............",
                         "Initialize Vector:....",
                         "No.of Fback charactor(1-8):..."
                  };
  static int air=0;
  int count=5;
  int column_menu = 111;
  int num_help = 111;
  act=0;
  c[0]="This algorithm is not use the Initialize Vector";
  c[1]="Use the previous cipher block to generate a present block";
  c[2]="Feedback the cipher block to generate a present block";
  c[3]="Feedback the output to generate the present block";
  c[4]="Multiple of 8 to 64 Output data bit feedback to system to generate next cipher";
  window_y1(220,220,470+50,400+10,2); /*(300,30,475,195,1);*//* determine sguare-point of window
*/
  parameter_label();
  setcolor(8);
  for(i=0;i<count;i++)
        outtextxy(238,20*i+290,str[i]);  /*(300+25,20*i+60,str[i]);*/
  if(abbr_mode==0)   /*USE FOR BAR IV OR NOR FOR EACH ALGORITHMS */
   {    setcolor(4);
        outtextxy(238,20*3+290,str[3]);
        outtextxy(238,20*4+290,str[4]);
   }
  if((abbr_mode==1) || (abbr_mode==2) || (abbr_mode==4) || (abbr_mode==5))
   { setcolor(4);
    outtextxy(238,20*4+290,str[4]);
   }
  show_under(3);
  control_key_parameter(air,count,215,474+50,260,42,1,abbr_mode,column_menu,num_help);
  air=static_air;
  if(act==1)
  {
   switch(air)
   {
        case 0: {
                  PLAINFILE();

                  ff1=fopen(plainfile,"rb");
                   if(ff1== NULL)
                    {
                     ERROR(250,210,475,290,5,1);
                     break;
                    }

                  ff2=fopen(cipherfile,"wb");
                   if(ff2== NULL)
                    {
                     ERROR(250,210,475,290,6,2);
                     break;
                    }

                 handle = open(plainfile,O_CREAT|O_TRUNC|O_BINARY,S_IREAD);
                 flength = filelength(handle);   /*these 5 lines can't place*/
                                                  /* before the ff1 file open,Why?*/
                 setcolor(12);
                 used_time[abbr_mode] = flength;
                 filesize[abbr_mode] = flength;
                   if(Key_digit < 8)
             {
                     ERROR(250,210,500,290,7,3);
                     break;
                   }
                   if(abbr_mode !=0)
                    {
                     if(IV_digit < 8)
                      {
                       ERROR(250,210,500,290,8,4);
                       break;
                      }
                    }
                   if(num_fb == 9)
             {
                     ERROR(250,210,538,306,9,5);
                     break;
                   }
                     setcolor(15);    /*clear header F2 */
                     outtextxy(500,maxy*.010416,"F2=Toggle word");

                     Restoreview(220,220,2);
                     Restoreview(180,180,1);
                     Restoreview(0,0,0);

                  window_whileprocess(120,120,465,240,4);
                  setcolor(BLUE);
                  outtextxy(140,150,"Processing time is about       seconds");
                  outtextxy(250,190,"Please wait");
                  if((abbr_mode ==0) || (abbr_mode==1) ||
                     (abbr_mode==4) ||(abbr_mode == 5))
                  flength = (flength/80) ;
                  if(abbr_mode == 2) flength = (flength/10) ;
                  if(abbr_mode == 3) flength = (flength/(10*num_fb));
                  setcolor(4);printtext(336,150,"%2.2f",flength);
                  ALL();

                  Restoreview(120,120,4);
                 } break;
    case 1:  out_menu=52;
               return 0;
    case 2:  out_menu=53;
               return 0;
    case 3:  out_menu=54;
               return 0;
    case 4:  out_menu=55;
               return 0;
   }
  }
```

```
return 0;
}
int ERROR(int x1,int y1,int x2,int y2,int restore_view,int err_msg)
{
int a;

char *er[11];
er[1]=" Can't open inputfile";
er[2]=" Can't open outputfile";
er[3]="Key must have 8 charactors";
er[4]="IV must have 8 charactors";
er[5]="Number of feedback charactor";
er[6]="Must be 1 to 8(8 to 64 bits)";
er[7]="Can't open file";
er[8]="Printer not ready";
er[9]="WARNING.!! NOT AN AUTHENTICATED FILE";
er[10]="DATA CHANGING IS OCCURED IN THIS FILE";

setcolor(15);    /*clear header F2 */
                        outtextxy(500,maxy*.010416,"F2=Toggle word");
                        printf("\a");
                        window_cannot_openfile(x1,y1,x2,y2,restore_view);
                        setcolor(RED);
           if(err_msg==3 || err_msg==4) outtextxy(x1+20,y1+35,er[err_msg]);
           if((err_msg==9) || (err_msg==10))
           outtextxy(x1+26,y1+57,er[err_msg]);
           else                      outtextxy(x1+20,y1+35,er[err_msg]);
           if(err_msg==5) outtextxy(x1+30,y1+51,er[err_msg+1]);

                        settextstyle(0,0,0);
                        getch();
                        Restoreview(x1,y1,restore_view);
   if(!(err_msg==7 || err_msg == 8 || err_msg ==9 || err_msg ==10))
                        {
                         Restoreview(220,220,2);
                         Restoreview(180,180,1);
                         Restoreview(0,0,0);
                        }
                        return(1);
                        }

 algorithm_label()
{
 void *algorithm_lab;
   algorithm_lab = malloc(imagesize(195,203,410,230));
   getimage(195,203,410,230,algorithm_lab);
   setfillstyle(1,7);
   setcolor(0);
   bar3d(195,203,410,230,0,0);
   putimage(195,203,algorithm_lab,XOR_PUT);
   free(algorithm_lab);
   setcolor(YELLOW); /*YELLOW IS 14*/
   outtextxy(255,215,"ALGORITHMS");
}
  parameter_label()
{
 void *parameter_lab;
   parameter_lab = malloc(imagesize(235,243,450+50,270));
   getimage(235,243,450+50,270,parameter_lab);
   setfillstyle(1,7);
   setcolor(0);
   bar3d(235,243,450+50,270,0,0);
   putimage(235,243,parameter_lab,XOR_PUT);
   free(parameter_lab);
   setcolor(YELLOW); /*YELLOW IS 14*/
    if(operate_mode==0)
           {
            switch(abbr_mode)
             {
             case 0: outtextxy(315,255,"ECB. ENCRYPTION"); break;
             case 1: outtextxy(315,255,"CBC. ENCRYPTION"); break;
             case 2: outtextxy(315,255,"CFB. ENCRYPTION"); break;
             case 3: outtextxy(315,255,"OFB. ENCRYPTION"); break;
             case 4: outtextxy(260,255,"PADDING AUTHENTICATION CODE"); break;
             case 5: outtextxy(299,255,"MESSAGE ORIGINATING"); break;
             }
           }
    if(operate_mode==1)
           {
            setcolor(GREEN);
            switch(abbr_mode)
             {
              case 0: outtextxy(315,255,"ECB. DECRYPTION"); break;
              case 1: outtextxy(315,255,"CBC. DECRYPTION"); break;
              case 2: outtextxy(315,255,"CFB. DECRYPTION"); break;
              case 3: outtextxy(315,255,"OFB. DECRYPTION"); break;
              case 4: outtextxy(307,255,"SHOULD NOT USED"); break;
              case 5: outtextxy(307,255,"RECIEVING MESSAGE"); break;
             }
           }
}

select_authen()
{
outtextxy(150,150,"authentication");
}

void DispFile(int x1,int y1,int x2,int y2)
{
   int i;
   char *str[]={"Encryption","Decryption"};
   window_y1(x1,y1,x2,y2,0);
   for(i=0;i<2;i++)
      outtextxy(22,20*i+30,str[i]);
}

int File(void)          /* menu bar for FILE headline */
{
   static int air=0;
   int count=2;
   int re_menu=3;
   int column_menu = 1;
   int num_help = 1;
   int x1=0,y1=0,x2=160,y2=85/*110*/;
   act=0;
   c[0] = "Encipher a file";
   c[1] = "Decipher a file";
    sefcolor(9);
    outtextxy(350,maxy*.010416,"F1=Toggle Help");
   while(1)
   {
   DispFile(x1,y1,x2,y2);
   show_under(0);
   re_menu=control_key_encfile(air,count,x1,x2,1,0,column_menu,num_help);
   air=static_air;
   operate_mode = air;
  if(act==1) {
    switch(air) {
          case 0:encryption(air); /*PASS air TO ENCRYPTION AND CHANGE TO NAME*/
                                  /*operate_mode TO SELECT THE PARAMETER LABEL*/
           if(NKey=='\r')
      return 0;
      break;
          case 1:encryption(air);/*select_ph();*//*MODIFIED*/
      if(NKey=='\r')
      return 0;
      break;
    }
}
if (re_menu !=3)
return(re_menu);
Restoreview(0,0,0);
}
}
void Disp_view_time(int x1,int x2,int y1,int y2)
{
   int i;
   char *str[]={"File contents","Processing time"};
   window_y1(x1,y1,x2,y2,0);          /* determine sguare-point of window */
   for(i=0;i<2;i++)
      outtextxy(x1+25,20*i+y1+30,str[i]);
}

int MAIN_VIEW(void)             /* set config of curve */
{
   int x1,x2,y1,y2;
   static int air=0;
   int count=2;/*3;*/                  /* number of row */
   int re_menu=3;
   int column_menu = 2;
   int num_help = 2;
   act=0;
   x1=65;x2=240;y1=0;y2=85;/*110;*/
   Disp_view_time(x1,x2,y1,y2);
   show_under(0);
    setcolor(9);
    outtextxy(350,maxy*.010416,"F1=Toggle Help");
   while(1)
   {
    c[0] = "Select this menu to view the plaintext and ciphertext file";
    c[1] = "Select this menu to view the time used by each algorithm";
/*    c[2] = "Select this menu to show the time the process used";*/
    re_menu=control_key(air,count,x1,x2,4,0,column_menu,num_help);
    air=static_air;
    if(re_menu==0 || re_menu==1 || re_menu==2)
     return(re_menu);
   }
}

set_under_white()
{
 setcolor(15);
 setfillstyle(1,15);
 bar3d(0,444,getmaxx(),460,0,0);
}

window_y1(int x1,int y1,int x2,int y2,int re_win)
{
   Saveview(x1,y1,x2,y2,re_win);
   setfillstyle(1,8);
   bar(x1+16,y1+17,x2-11,y2-12);
   setfillstyle(1,15);
   bar(x1+10,y1+17,x2-17,y2-18);
   setcolor(8);
   rectangle(x1+14,y1+22,x2-20,y2-21);
}
window_time(int x1,int y1,int x2,int y2,int re_win)
{
   Saveview(x1,y1,x2,y2,re_win);
   setfillstyle(1,8);
   bar(x1+16,y1+17,x2-11,y2-12);
   setfillstyle(1,7);
   bar(x1+10,y1+17,x2-17,y2-18);
   setcolor(4);
   rectangle(x1+14,y1+22,x2-20,y2-21);
}
window_whileprocess(int x1,int y1,int x2,int y2,int re_win)
{
   Saveview(x1,y1,x2,y2,re_win);
   setfillstyle(1,8);
   bar(x1+16,y1+20,x2-11,y2-12);
   setfillstyle(1,14);
   bar(x1+10,y1+17,x2-17,y2-18);
   setcolor(4);
   rectangle(x1+14,y1+22,x2-20,y2-21);
}
window_cannot_openfile(int x1,int y1,int x2,int y2,int re_win)
{
   Saveview(x1,y1,x2,y2,re_win);
   setfillstyle(1,8);
   bar(x1+16,y1+20,x2-11,y2-12);
   setfillstyle(1,2);
   bar(x1+10,y1+17,x2-17,y2-18);
   setcolor(4);
   rectangle(x1+14,y1+22,x2-20,y2-21);
}
int control_key(int air_before,int count,int x1,int x2,int column,
                        int re_win,int column_menu,int num_help)
{
   int re_menu=3;       /* return menu */
   int air;
/*   int menu = 2;*/
/*   int num_help = 2;*/
   air=air_before;
   act=0;
   write_menutext(x1+20,26+20*air,x2-25,39+20*air);
   set_under_white();
   setcolor(8);
   outtextxy(100,450,c[air]);
   while(1)
   {
      act=0;
      NKey=getkey();
      write_menutext(x1+20,26+20*air,x2-25,39+20*air);
      switch(NKey)
      {
            case  DN: air++;  break;
            case  UP: air--;  break;
            case  '\r': act=1;  break;
            case  LEFT:;
            case  RIGHT:; Restoreview(x1,0,re_win); break;
            case  F1:; ShowHelp(column_menu,num_help); break;
            case  ALT_X:;closegraph();restorecrtmode(); exit(1);
      }
      if(air == count)  air=0;
      if(air<0)  air=count-1;
      static_air=air;                    /* save static air */
      if((NKey==LEFT)||(NKey==RIGHT)) return 2;
    /*  if(NKey==ESC) return 0; */
      if(NKey==ALT_X) return 1;
      set_under_white();
      setcolor(8);
      outtextxy(100,450,c[air]);
      write_menutext(x1+20,26+20*air,x2-25,39+20*air);
      if(act==1)
      {
       if(air==0) re_menu=Show_View(air,column,x1,0,re_win);
       if(air==1) re_menu=Show_Time(air,column,x1,0,re_win);
       if(re_menu==0 || re_menu==3)
            return(re_menu);
      }

   }
}

Show_View(int air,int column,int x1,int y1)
{
  if(column==4)
   {
   VIEW(); /*VIEW THE CONTENT OF THE FILES */
   return 1;
  }
 if(column!=1 && column!=4)
   return 3;
}

VIEW()
   /*********************************/
   /*VIEW THE CONTENT OF THE FILES */
   /*********************************/

 {
 unsigned long art;
 void far * area1;
 void far * area2;
 void far * area3;

 art   = imagesize(0,0,639,159);
 area1 = farmalloc(art+1);
 area2 = farmalloc(art+1);
 area3 = farmalloc(art+1);
 getimage(0,0,639,159,area1);
 getimage(0,160,639,319,area2);
 getimage(0,320,639,479,area3);
 closegraph();clrscr();
 /*   printf("gfkdafjfgdj\n");getch();
 viewf("c:\\tc\\view\\viewf.c",0);*/
system("c:\\tc\\view\\ok-view.exe");
 SetGraph();

 cleardevice();
 putimage(0,0,area1,COPY_PUT);
 putimage(0,160,area2,COPY_PUT);
 putimage(0,320,area3,COPY_PUT);
 farfree(area1);
 farfree(area2);
 farfree(area3);
}
Show_Time(int air,int column,int x1,int y1)
{
  if(column==4)
  {
   TIME(); /*SHOW TIME USED AND FILESIZE */
   return 1;
  }
 if(column!=1 && column!=4)
   return 3;
}
TIME()
     /******************************
      * SHOW TIME USED AND FILESIZE *
      ********************************/
{
char ecb[4],cbc[4],cfb[4],ofb[4];
window_time(100,100,540,390,11);
setcolor(BLUE);
setlinestyle(0,4,3);
line(170,300,500,300);
line(170,300,170,150);
line(170,150,500,150);
line(500,150,500,300);
setlinestyle(0,4,1);
 outtextxy(135,300,"0.0");
 outtextxy(135,270,"7.5"); line(170,270,500,270);
 outtextxy(127,240,"15.0");line(170,240,500,240);
 outtextxy(127,210,"22.5");line(170,210,500,210);
 outtextxy(127,180,"30.0");line(170,180,500,180);
 outtextxy(113,150,"Time(s)");
setfillstyle(1,2); if(used_time[0]/80 /*ecb_time*/ >37.5)
                             { bar3d(190,300-(150),240,300,1,1);
                               setcolor(15); outtextxy(193,225,"excess");
                               setcolor(BLUE);
                             }
                          else
                             {
                              bar3d(190,300-(4*used_time[0]/80 /* ecb_time*/),
                              240,300,0,1);
                             }
setfillstyle(1,4); if(used_time[1]/80 /* cbc_entime */ >37.5)
                             { bar3d(270,300-(150),320,300,1,1);
                               setcolor(15); outtextxy(273,225,"excess");
                               setcolor(BLUE);
                             }
                          else
                             {
                              bar3d(270,300-(4*used_time[1]/80 /* cbc_entime */),
                              320,300,0,1);
                             }
setfillstyle(1,6); if(used_time[2]/10 /*cfb_time */ >37.5)
                             { bar3d(350,300-(150),400,300,1,1);
                               setcolor(15); outtextxy(353,225,"excess");
                               setcolor(BLUE);
                             }
                          else
                             {
                              bar3d(350,300-(4*used_time[2]/8 /*cfb_time*/),
                              400,300,0,1);
                             }
setfillstyle(1,5);
                          if(num_fb ==0)    bar3d(430,300,480,300,0,1);
                          else
                           {
                              if(used_time[3]/(10*num_fb) /*ofb_time*/ >37.5)
                              { bar3d(430,300-(150),480,300,1,1);
                               setcolor(15); outtextxy(433,225,"excess");
                               setcolor(BLUE);
                               }
                               else
                               {
                              bar3d(430,300-(4*used_time[3]/(10*num_fb) /*ofb_time*/ ),
                                    480,300,0,1);
                               }
                          }
setcolor(8);
 outtextxy(120,320,"Algorithm");
 outtextxy(120,336," Filesize");
 outtextxy(120,352,"Used time");
setcolor(15);
 outtextxy(203,320,"ECB       CBC       CFB       OFB");
 sprintf(ecb,"%2.1f",used_time[0]/80 /*ecb_time*/);      outtextxy(203,352,ecb);
```

```
  printtext(203,338,"%d",filesize[0]);
  sprintf(cbc,"%2.1f",used_time[1]/80 /*cbc_entime*/);    outtextxy(283,352,cbc);
   printtext(283,338,"%d",filesize[1]);
  sprintf(cfb,"%2.1f",used_time[2]/8 /*cfb_time*/);     outtextxy(363,352,cfb);
   printtext(363,338,"%d",filesize[2]);
  if(num_fb == 0)
     sprintf(ofb,"%2.1f",/*used_time[3]/(10*num_fb)*/ ofb_time);
  else
     sprintf(ofb,"%2.1f",used_time[3]/(10*num_fb)/* ofb_time */);
     outtextxy(443,352,ofb);
     printtext(443,338,"%d",filesize[3]);
while(27!=getch());
Restoreview(100,100,11);
}
control_key_w1(int air_before,int count,int x1,int x2,
                int y1,int column,int re_win)
{
   int re_menu=0;      /* return menu */
   int adj_y=0;
   int air=0;
   air=air_before;
   act=0;
   if(re_win==2)       /* adj value of y in window 2 */
    adj_y=20;
   write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);
   set_under_white();
   setcolor(8);
   outtextxy(100,450,c[air]);
   while(1)
   {
      set_under_white();
      setcolor(8);
      outtextxy(100,450,c[air]);
      NKey=getkey();
      write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);
      switch(NKey)
      {
            case  F3: break;
            case  DN: air++;  break;
            case  UP: air--;  break;
            case '\r': act=1;  break;
            case  ESC:; Restoreview(x1,y1,re_win); break;
            case  ALT_X:; closegraph();restorecrtmode(); exit(1);
      }
      if(re_win==2)
      {
       if(NKey==F3)
       {
            vi[air]=10;
            spp[air]=10;
       Restoreview(x1,y1,re_win);
       }
      }
      if(air == count)  air=0;
      if(air<0)  air=count-1;
      static_air=air;                 /* save static air */
      if(NKey==ESC)
      {
       if(column==4 || re_win==2)
       {
            c[0] = "Setting number of curve for plotting by program FFT";
            set_under_white();
            setcolor(8);
            outtextxy(100,450,c[0]);
       }
       return 0;
      }
      if(NKey==ALT_X) return 0;
      set_under_white();
      setcolor(8);
      outtextxy(100,450,c[air]);
            write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);
      if (act==1)
      {
       if(column==42)
            return (air);
       else
       {
            return 0;
       }
      }

   }
}
control_key_encrypt(int air_before,int count,int x1,int x2,
int y1,int column,int re_win,int column_menu,int num_help)
{
    int re_menu=0;      /* return menu */
    int adj_y=0;
    int air=0;
  /* int column_menu = 1;
    int num_help = 11;*/
    air=air_before;
    act=0;
    if(re_win==2)       /* adj value of y in window 2 */
     adj_y=0;
    write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);
    setcolor(5);
    outtextxy(500,maxy*.010416,"F2=Toggle word");
    set_under_white();
  setcolor(8);
  outtextxy(100,450,c[air]);
  while(1)
  {
    set_under_white();
    setcolor(8);
    outtextxy(100,450,c[air]);
    NKey=getkey();
    write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);
    switch(NKey)
    {
         case  F3: break;
         case  DN: air++;  break;
         case  UP: air--;  break;
         case '\r': if((air==4) && (operate_mode ==1))
          /*this condition is used to*/
          /*mask wether the file AC.menu in decryption*/
          /*or not if yes show messsages */

                      {
                          NOT_USED_DEMAC();break;
                      }
                    else  act=1;  break;
         case  ESC:;{
                       setcolor(15);
                       outtextxy(500,maxy*.010416,"F2=Toggle word");
                       Restoreview(180,180,re_win);
                       break;
                      }
         case  ALT_X:; closegraph();restorecrtmode(); exit(1);
         case  F1:; ShowHelp(column_menu,num_help); break;
         case  F2:; showword(air); break;
         }
    if(air == count)  air=0;
    if(air<0)  air=count-1;
    static_air=air;                        /* save static air */
    if(NKey==ESC)
    {
     return 0;
    }
    if(NKey==ALT_X) return 0;
    set_under_white();
    setcolor(8);
    outtextxy(100,450,c[air]);
    write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);
    if (act==1)
    {
     if(column==42)
          return (air);
     else
     {
          return 0;
     }
    }

  }
}
 control_key_parameter(int air_before,int count1,int x1,int x2,
 int y1,int column,int re_win,int IV,int column_menu,int num_help)
{
  int re_menu=0;       /* return menu */
  int adj_y=0;
  int air=0;
  int count;
/*   int column_menu = 1;
  int num_help = 111; */    /*is an algo_index in showword(); */
  air=air_before;
  act=0;
  write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);
  setcolor(5);
  outtextxy(500,maxy*.010416,"F2=Toggle word");
  set_under_white();
  setcolor(8);
  outtextxy(100,450,c[air]);
  while(1)
  {
    set_under_white();
    setcolor(15);
    outtextxy(100,450,c[air]);
    NKey=getkey();
    write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);
    switch(NKey)
    {
         case '\r': act=1;  break;
         case  ESC:  {
                       setcolor(15);
                       outtextxy(500,maxy*.010416,"F2=Toggle word");
                       Restoreview(220,220,re_win+1);/*RESTORE PARAMETER BLOCK*/
                       Restoreview(180,180,re_win);  /*RESTORE ALGORITHM BLOCK */
                       break;
                      }
         case  ALT_X:; closegraph();restorecrtmode(); exit(1);
         case F1:; ShowHelp(column_menu,num_help); break;
         case F2:; showword(num_help); break;
         }
         /* USE FOR BAR IV OR NOT FOR EACH ALGORITHMS */
         if(IV==0) count = count1-2;
         if((IV==1) || (IV==2) ||(IV==4) || (IV == 5))
          {
           count = count1-1;
```

```
            } else count = count1;

          if(air == count)  air=0;
     if(air<0)  air=count-1;
     static_air=air;                          /* save static air */

          if(NKey==ESC)  return 0;
     if(NKey==ALT_X) return 0;

          set_under_white();
     setcolor(8);
     outtextxy(100,450,c[air]);
          write_menutext(x1+20,y1+26+20*air+adj_y,x2-25,y1+39+20*air+adj_y);

          if (act==1)
     {
      if(column==42)
            return (air);
           }

    }
}
int control_key_encfile(int air_before,int count,int x1,int x2,int column,
                                  int re_win,int column_menu,int num_help)
{
   int re_menu=3;       /* return menu */
   int air;
   air=air_before;
   act=0;
   write_menutext(x1+20,26+20*air.x2-25,39+20*air);
   set_under_white();
   setcolor(8);
   outtextxy(100,450,c[air]);
   while(1)
   {
     act=0;
     NKey=getkey();
     write_menutext(x1+20,26+20*air,x2-25,39+20*air);
     switch(NKey)
     {
          case  DN:    air++;  break;
          case  UP:    air--;  break;
          case '\r':   act=1;  break;
          case  LEFT:  ;
          case  RIGHT: Restoreview(x1,0,re_win); break;
          case  F1::   ShowHelp(column_menu,num_help); break;
          case  ALT_X: closegraph();restorecrtmode(); exit(1);
     }
     if(air == count)  air=0;
     if(air<0)  air=count-1;
     static_air=air;                     /* save static air */
     if((NKey==LEFT)||(NKey==RIGHT)) return 2;
  /*  if(NKey==ESC) return 0; */
     if(NKey==ALT_X) return 1;
     set_under_white();
     setcolor(8);
     outtextxy(100,450,c[air]);
     write_menutext(x1+20,26+20*air,x2-25,39+20*air);
     if(act==1)
     {
      re_menu=choose_y1_encfile(air,column,x1,0,re_win);
      if(re_menu==0 || re_menu==3)
          return(re_menu);
     }

   }
}
ShowHelp(int column_menu,int num_help)
{
int toggle;

if(column_menu==1)
 {
  window_y1(150,100,470,240,6);
  setfillstyle(1,7);
  bar3d(164,124,450,142,0,0);
  setcolor(1);
  outtextxy(276,129,"Operation");
  setcolor(9);
  outtextxy(170,154,"This menu is the  parameter input");
  outtextxy(170,170,"menu. Use the scroll bar to select");
  outtextxy(170,186,"the operation then and press ENTER ");
  outtextxy(170,202,"to initiate the DES card");
   do {
       (toggle=getch());
      } while (toggle != 59); /*F1== 59*/
  Restoreview(150,100,6);
 }
if(column_menu == 2)
 {
  window_y1(150,100,470,258,6);
  setfillstyle(1,7);
  bar3d(164,124,450,142,0,0);
  setcolor(1);
  outtextxy(290,129,"View");
  setcolor(9);
  outtextxy(170,154,"This menu supports you to view the");
  outtextxy(170,170,"pair of plaintext  and  ciphertext");
  outtextxy(170,186,"file  and also  allows you  to see  ");
  outtextxy(170,202,"the processing  time used by each ");
  outtextxy(170,218,"encryption or decryption algorithm");
  do {
      (toggle=getch());
     } while (toggle != 59); /*F1== 59*/
  Restoreview(150,100,6);
 }
if(column_menu == 3)
 {
  window_y1(150,100,470,293,6);
  setfillstyle(1,7);
  bar3d(164,124,450,142,0,0);
  setcolor(1);
  outtextxy(290,129,"Print");
  setcolor(9);
  outtextxy(170,154,"This menu provides you to make a");
  outtextxy(170,170,"hardcopy  of any files. You must");
  outtextxy(170,186,"know the type of file you want to ");
  outtextxy(170,202,"print out. Select such type by the ");
  outtextxy(170,218,"scroll bar and key in the filename");
  outtextxy(170,234,"( Misselect type of file will give");
  outtextxy(170,250,"you a not proper results.)");
  do {
      (toggle=getch());
     } while (toggle != 59); /*F1== 59*/
  Restoreview(150,100,6);
 }
if(num_help == 11)
 {
  window_y1(150,100,470,265,6);
  setfillstyle(1,7);
  bar3d(164,124,450,142,0,0);
  setcolor(1);
  outtextxy(274,129,"Algorithm");
  setcolor(9);
  outtextxy(170,154,"This menu is a submenu of Operation");
  outtextxy(170,170,"menu. After select the operation ");
  outtextxy(170,186,"you must select an algorithm. Each ");
  outtextxy(170,202,"one has a different characteristic.");
  outtextxy(170,218,"Press F1 and then F2 to find it out");
  do {
      (toggle=getch());
     } while (toggle != 59); /*F1== 59*/
  Restoreview(150,100,6);
 }
if(num_help == 111)
 {
  window_y1(150,100,486,293,6);
  setfillstyle(1,7);
  bar3d(164,124,466,142,0,0);
  setcolor(1);
  outtextxy(276,129,"Parameter");
  setcolor(9);
  outtextxy(170,154," You can see two colors of the");
  outtextxy(170,170,"charactor(except for OFB algorithm)");
  outtextxy(170,186,"The black one is neccesary to the");
  outtextxy(170,202,"operation and algorithm you select");
  outtextxy(170,218,"before while the red one is not.");
  outtextxy(170,238,"Press F1 then F2 to find the meaning");
  outtextxy(170,254,"of them.");
  do {
      (toggle=getch());
     } while (toggle != 59); /*F1== 59*/
  Restoreview(150,100,6);
 }

}


choose_y1_encfile(int air,int column,int x1,int y1)
{
  if(column==1)
  {
   switch(air)
   {
    case 0:  out_menu=0;        /* Load file from disk */
               return 3;
    case 1:  out_menu=0;        /* New data in process */
               return 3;
    case 2:  out_menu=0;        /* Save File           */
               return 3;
    case 3:  break;
    case 4:  NKey=ALT_X;  closegraph(); break;
   }
  }
  if(column==4)
  {
   if(air==1 && number_c==0)
   {
    return 1;
   }
   else
   {
     return 1;
   }
  }
  if(column!=1 && column!=4)
   return 3;
}
```

```
Help()
{
 return 2;
}

le_ri(int x)
{
 char a[3];
 setcolor(BLUE);
 sprintf(a,"%c",26);
 outtextxy(x,470+2,a);
 sprintf(a,"%c",27);
 outtextxy(x,470-3,a);
 setcolor(WHITE);
  outtextxy(10+x,470,"-Move");
}
up_dn(int x)
{
 char a[3];
 setcolor(BLUE);
 sprintf(a,"%c",24);
 outtextxy(x-3,470,a);
 sprintf(a,"%c",25);
 outtextxy(x+3,470,a);
 setcolor(WHITE);
 outtextxy(10+x,470,"-Move");
}
le_ri_up_dn(int x)
{
 char a[3];
 setcolor(BLUE);
 sprintf(a,"%c",26);
 outtextxy(x,470+2,a);
 sprintf(a,"%c",27);
 outtextxy(x,470-3,a);
 sprintf(a,"%c",24);
 outtextxy(x-3+10,470,a);
 sprintf(a,"%c",25);
 outtextxy(x+3+10,470,a);
 setcolor(WHITE);
 outtextxy(20+x,470,"-Move");
}
enter_key(int x,int order)
{
 char a[3];
 sprintf(a,"%c",17);
 setcolor(BLUE);
 outtextxy(x,470,a);
 sprintf(a,"%c",196);
 outtextxy(x+6,470,a);
 sprintf(a,"%c",217);
 outtextxy(x+12,470,a);
 setcolor(WHITE);
 switch(order)
 {
  case 0: outtextxy(18+x,470,"-Select");break;
  case 1: outtextxy(18+x,470,"-Start-Stop Curve"); break;
  case 2: outtextxy(18+x,470,"-Previous Curve"); break;
  case 3: outtextxy(18+x,470,"-Show Report"); break;
  case 4: outtextxy(18+x,470,"-Validation of Data"); break;

 }
}
alt_x(int x)
{
 setcolor(BLUE);
 outtextxy(x,470,"ALT_X");
 setcolor(WHITE);
 outtextxy(x+42,470,"-Quit");
}
esc(int x)
{
 setcolor(BLUE);
 outtextxy(x,470,"ESC");
 setcolor(WHITE);
 outtextxy(x+26,470,"-Exit");
}
esc_menu(int x)
{
 setcolor(BLUE);
 outtextxy(x,470,"ESC");
 setcolor(WHITE);
 outtextxy(x+26,470,"-Menu");
}
cancel(int x)
{
 setcolor(BLUE);
 outtextxy(x,470,"F3");
 setcolor(WHITE);
 outtextxy(x+18,470,"-Cancel Data");
}
tab(int x)
{
 setcolor(BLUE);
 outtextxy(x,470,"TAB");
 setcolor(WHITE);
 outtextxy(x+26,470,"-Next Window");
}
edit_F2(int x)
{
 setcolor(BLUE);
 outtextxy(x,470,"F2");
 setcolor(WHITE);
 outtextxy(x+18,470,"-Edit Curve");
}
edit_F5(int x)
{
 setcolor(BLUE);
 outtextxy(x,470,"F5");
 setcolor(WHITE);
 outtextxy(x+18,470,"-Previous Curve");
}
edit_F6(int x)
{
 setcolor(BLUE);
 outtextxy(x,470,"F6");
 setcolor(WHITE);
 outtextxy(x+18,470,"-Next Curve");
}


show_under(int order)
{
 int dx,d1;
 setcolor(BLACK);
 setfillstvle(1,BLACK);
 bar3d(0,464,getmaxx(),getmaxy(),0,0);

 switch(order)
 {
  case 0: dx=90;d1=100;
             le_ri_up_dn(dx);enter_key(dx+d1+60,0);alt_x(dx+2*d1+60+64);break;
  case 1: dx=90;d1=100;
             le_ri(dx);enter_key(dx+d1+48,0);alt_x(dx+2*d1+48+64);break;
  case 2: dx=40;d1=50;
             up_dn(dx);enter_key(dx+d1+48,0);cancel(dx+2*d1+48+64);esc(dx+3*d1+48+64+104);
             alt_x(dx+4*d1+48+64+96+64);break;
  case 3: dx=80;d1=80;
             up_dn(dx);enter_key(dx+d1+48,0);esc(dx+2*d1+48+64);alt_x(dx+3*d1+48+64+64);break;
  case 4: dx=90;d1=100;
             le_ri(dx);enter_key(dx+d1+48,0);alt_x(dx+2*d1+48+64);break;
  case 5: dx=150;d1=150;
             le_ri_up_dn(dx);alt_x(dx+d1+60);break;
  case 6: dx=20;d1=30;
             le_ri_up_dn(dx);enter_key(dx+d1+60,1);tab(dx+2*d1+60+146);
             esc_menu(dx+3*d1+60+146+120);
             alt_x(dx+4*d1+60+146+120+64);break;
  case 7: dx=20;d1=34;
             le_ri_up_dn(dx);enter_key(dx+d1+60,2);tab(dx+2*d1+60+130);
             esc_menu(dx+3*d1+60+130+120);
             alt_x(dx+4*d1+60+130+120+64);break;
  case 8: dx=20;d1=37;
             le_ri_up_dn(dx);enter_key(dx+d1+60,3);tab(dx+2*d1+60+112);
             esc_menu(dx+3*d1+60+112+120);
             alt_x(dx+4*d1+60+112+120+64);break;
  case 9: dx=90;d1=100;
             edit_F2(dx);esc_menu(dx+d1+106);alt_x(dx+2*d1+106+64);
             break;
  case 10:dx=20;d1=30;
             edit_F5(dx);edit_F6(dx+d1+144);
             enter_key(dx+2*d1+144+112,4);alt_x(dx+3*d1+144+112+178);
             break;
  case 11: dx=50;d1=70;
             le_ri_up_dn(dx);esc_menu(dx+2*d1+60);alt_x(dx+3*d1+60+64);break;
 }
}


PLAINFILE()

{
int amount_para,line_para,Esc_From_Para;
/*=3 or 4 if not has IV, =5 if has */

 if(abbr_mode==0) amount_para = 3; /*ECB MODE */
 if((abbr_mode==1) || (abbr_mode==2) || (abbr_mode==4) ||
    (abbr_mode==5)) amount_para = 4; /*CBC and CFB */
 if(abbr_mode==3) amount_para = 5; /*OFB MODE */
/*THIS 'FOR' LOOP USED FOR COLLECT PARAMETER */
  for(line_para=1;line_para<=amount_para;line_para++)
   {
   Esc_From_Para = greadfloat(328,286+(20*(line_para-1)),20,line_para,111,111);
    if(Esc_From_Para == 5)
     {
      line_para = amount_para;
     }
  }
}

int greadfloat(int xloc,int yloc,int digit,int line_para,
                    int column_menu,int num_help)

 {
  int ch;
  /* char *cipherfile,*secretkey,*vector;*/
  /*  float  value;*/
  register int i,tr,color;
/*  int column_menu = 111;
  int num_help = 111;*/
  i = 0;
```

```
tr = xloc;
ch = 0;
         if(line_para==2) write_menutext(235,306,499,319);
         if(line_para==3) write_menutext(235,306+20,499,319+20);
         if(line_para==4) write_menutext(235,306+40,499,319+40);
         if(line_para==5) write_menutext(235,306+60,499,319+60);
 setfillstyle(1,8);/*getbkcolor());*/
 bar(xloc-8,yloc,xloc+digit*8,yloc+13);
  while(ch != '\r')
         {
         /*   color = getcolor();*/
            setcolor(8);/*getbkcolor());*/
            outtextxy(xloc-8,yloc+3," _");
            setcolor(13); /*color);*/
            outtextxy(xloc,yloc+3," _");

            ch = getkey();/*toupper(getch());*/
            string[line_para][i] = ch;

            if(ch == 27) return(5); /*27 = ESC key and return 5 to greadnfloat*/

            if(ch == '\r')
             {

                     if(i != 0)
                     break;
                     else  { xloc=tr;ch = ' ';continue;}
             }
            if(ch == 187)   /*CHECK FOR F1*/
             {
              ShowHelp(column_menu,num_help);
              setfillstyle(1,8);
             }
            if((ch == 188) && (num_help == 111))   /*CHECK FOR F2*/
             {
              showword(3);
              setfillstyle(1,8);
             }
            if(ch == 8)     /*CHECH FOR BACKSPACE */
             {
                     setcolor(13);
                     xloc = xloc - 8;
                     bar(xloc,yloc,xloc+16,yloc+13);
                      if(i == 0)
                       {
                               repeat:
                               i++;
                               xloc = tr;
                       }
                      i--;
                     continue;
             }
           switch(line_para) {
           case 1: if(i >= digit )  continue; break;
           case 2: if(i >= digit )  continue; break;
           case 3: if(i >= digit-12 ) continue; break;/*LIMIT INPUT CHAR TO 8 */
           case 4: if(i >= digit-12 ) continue; break;
           case 5: if(i >= digit-19 ) continue; break;/*LIMIT CHAR TO 1 OR 2 */
           }
           if((line_para == 1) || (line_para == 2) || (line_para == 5))
             {
             if((ch < 32) || (ch > 126) || (ch == 187) || (ch == 188)) continue;
             printtext(xloc,yloc+3,"%c",string[line_para][i]);
             xloc = xloc + 8;
             i++;
             }
           if((line_para == 3) || (line_para == 4))
             {
             if((ch < 32) || (ch > 126) || (ch == 187) || (ch == 188)) continue;
            /*  if((ch == 59) || (ch == 60) || (ch == 27)) continue;*/
              printtext(xloc,yloc+3,"%s","*");
              xloc = xloc + 8;
              i++;
              if(line_para == 3) Key_digit = i;   /*index to error message*/
              else IV_digit = i;
             }

   }
  string[line_para][i] = '\0';
  if(line_para==1)
    plainfile = string[line_para];
/*     plainfilesize = string[line_para];*/
  if(line_para==2) cipherfile = string[line_para];
  if(line_para==3) secretkey = string[line_para];
  if(line_para==4) vector = string[line_para];
  /* DO NOT FORGET TO USE FUNCTION 'atoi()'TO CONVERT STRING TO INTEGER */
           if(line_para==5)
            { ofb_fback = string[line_para];
              switch(*ofb_fback)
               {
                case '1':num_fb = 1; break;
                case '2':num_fb = 2; break;
                case '3':num_fb = 3; break;
                case '4':num_fb = 4; break;
                case '5':num_fb = 5; break;
                case '6':num_fb = 6; break;
                case '7':num_fb = 7; break;
                case '8':num_fb = 8; break;
                default :num_fb = 9; break;
               }
         }
  bar(xloc,yloc,xloc+8,yloc+13);
}


 printtext(int xloc,int yloc,char *fmt,...)
{
 va_list argptr;
 char    str[140];

 va_start(argptr,fmt);
 vsprintf(str,fmt,argptr);
 outtextxy(xloc,yloc,str);
 va_end(argptr);
}

/* CONGRATULATION THIS PROGRAM IS WORK PROPERLY ! PROVED NOW...*/
/*THE KEY IN THIS PROGRAMM IS ABLE TO CHANGE BEFORE YOU HIT THE ENTER KEY
*/



char *key_reassign();

void waitready(void)

{
   char status ;
         do {
         status = inportb(STATUSPORT) ;
         } while((status&0x02)!=0) ;  /* IBF = 0 ? */
}

 check_complete(void) /* FUNCTION FOR CHECKING THE COMPLETION OF PROCESS */

{
int last;
 last = inportb(STATUSPORT);
   if((last & 0x01) != 00) {
                 printf("\aThe process is not completed.\n");
                 printf("Reset the DEU and try again.\n");
   }/* else  { printf("\nStatus of the port after conversion is %x\n",last);*/
                /*  printf("Conversion completed.\n"); */
/*               }*/
}


ALL()

{
int COMMAND;
/*if(op[0] == 'E')*/
   if(operate_mode == 0)
   COMMAND = 0x30;  /* ENCRYPTION MODE */
   if(operate_mode == 1)
   COMMAND = 0x20;  /* DECRYPTION MODE */
   enterkey(COMMAND);
}


 int Menu(void)
{
   char *cc[5];
   static int arrow=0;
   int flag=1,active=2;
   int count=4, on=0, under=15;  /*former count = 5 */
   int stx[5] = {15, 105, 165, 221, 585};/*set the blacklabel when move arrow*/
   int enx[5] = {95, 149, 215, 274, 650};
   write_menutext(stx[arrow],on,enx[arrow],under);
   while(1)

   {
    out_menu=0;
    flag=1;
   /* if(active != 2)  */  /* this use only for press ENTER again to go into*/
   /*   NKey = getkey();*/ /* File function */
    write_menutext(stx[arrow],on,enx[arrow],under);
    switch(NKey)
    {
     case '\r':flag=1; break;
     case RIGHT: arrow++;     break;
     case  LEFT: arrow--;     break;
     case ALT_X: closegraph();restorecrtmode(); exit(1);
    }
    if(arrow==count)  arrow=0;
    if(arrow<0)        arrow=count-1;
/*    if((arrow==1)&(flag==1))  if(active!=2) break;*/
    set_under_white();
    setcolor(8);
    outtextxy(100,450,cc[arrow]);
    write_menutext(stx[arrow],on,enx[arrow],under);
    if(flag==1)
    {
     switch(arrow)
     {
           case  0: active=File(); break;
           case  1: active=MAIN_VIEW(); break;
           case  2: active=MAIN_PRINT();
                          break;
```

```
          case  3: active=Quit();
                         break;
/*        case  4: active=Help(); break;
          case  5:  break; */
      }
    }
    if(out_menu!=0)
      break;
  /* if(active==1) break;*/
    }
 return NKey;
}
/* ECB MODE HAS NOT THE SELF-SYNCHRONOUS PROPERTY */

ECB(int COMMAND)    /* FOR ENCRYPTION AND DECRYPTION PROCESS */

{
int ff1status,ff2status:
/* time_t  start,end; */

/*start = time(NULL);*/   /*STARTING TIME */
waitready();
 outportb(COMMANDPORT,COMMAND);
 waitready();
do {

    for (i = 0;i<=7;i++){       /*receive plain text*/
    waitready();
    plain = getc(ff1);
    outportb(DATAINPORT,plain);
    waitready();
    }

      while((inportb(STATUSPORT) & 0x02) != 0x00);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while((inportb(STATUSPORT) & 0x01) != 01);
if(COMMAND == 0x30)
  {
       for (i=0;i<=7;i++) {
       while( (inportb(STATUSPORT) & 0x01) != 01);
       cipher = inportb(DATAOUTPORT);
       putc(cipher,ff2);
       }
 }
else
  {
   if(!feof(ff1))
     {
       for (i=0;i<=7;i++) {
       while( (inportb(STATUSPORT) & 0x01) != 01);
       cipher = inportb(DATAOUTPORT);
       putc(cipher,ff2);
       }
     } else
              {   /*use for clear the data in DEU dataout buffer */
               for (i=0;i<=7;i++) {
                while( (inportb(STATUSPORT) & 0x01) != 01);
                cipher = inportb(DATAOUTPORT);
                }
              }
  }
} while(!feof(ff1));
   fclose(ff1);
   fclose(ff2);
   check_complete();
 /*  end = time(NULL);*/ /* ENDIND TIME */
/*   ecb_time = difftime(end,start); */
}

/* CBC MODE HAS THE SELF-SYNCHRONOUS PROPERTY */

CBCEN(int COMMAND)    /* CIPHER BLOCK CHAINING (ENCRYPT) */

{

char IV[8],PCBC,cipher;

waitready();
 outportb(COMMANDPORT,COMMAND);
 waitready();

do {

    for (i = 0;i<=7;i++){       /*receive plain text*/
    waitready();
    plain = getc(ff1);
    PCBC = plain ^ string[4][i];
    outportb(DATAINPORT,PCBC);
    waitready();
    }

      while((inportb(STATUSPORT) & 0x02) != 0);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while( (inportb(STATUSPORT) & 0x01) != 01);

    for (i=0;i<=7;i++) {
    while( (inportb(STATUSPORT) & 0x01) != 01);
    cipher = inportb(DATAOUTPORT);
    putc(cipher,ff2);
    string[4][i] = cipher;
    }
} while(!feof(ff1));
    fclose(ff1);
    fclose(ff2);
    check_complete();
}

CBCDE(int COMMAND)    /* CIPHER BLOCK CHAINING (DECRYPT)*/

{
int temp;
char IV[8],PCBC;
char xcipher0[8],xcipher1[8],xplain;

waitready();
 outportb(COMMANDPORT,COMMAND);
 waitready();

do {

    for (i = 0;i<=7;i++){        /*receive plain text*/
    waitready();
    xcipher1[i] = getc(ff1);
/*     PCBC = xplain ^ string[4][i]; */
    outportb(DATAINPORT,xcipher1[i]);
    waitready();
    }

      while((inportb(STATUSPORT) & 0x02) != 0);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while( (inportb(STATUSPORT) & 0x01) != 01);

    if(!feof(ff1))
    {
            for (i=0;i<=7;i++) {
             while( (inportb(STATUSPORT) & 0x01) != 01);
             xplain = inportb(DATAOUTPORT);
             PCBC = xplain ^ string[4][i];
             putc(PCBC,ff2);
             string[4][i]=xcipher1[i];
            }
    } else
      {
       for(i=0;i<=7;i++)    /*use to clear the DEU dataout buffer */
            {
             while( (inportb(STATUSPORT) & 0x01) != 01);
             xplain = inportb(DATAOUTPORT);
            }
      }
} while(!feof(ff1));
    fclose(ff1);
    fclose(ff2);
    check_complete();

}
/* CFB MODE HAS THE SELF-SYNCHRONOUS PROPERTY */

CFB()  /* CIPHER FEEDBACK ENCRYPT AND DECRYPT MODE *
         * THIS MODE IS A STREAM CIPHER MODE         */

         /* In this mode of operation both the ciphering *
          * and decipherng process use the encryption(not*
          * decryption in deciphering process) so we set *
          * COMMAND to 0x30 */

{

#define  COMMAND 0x30
char     IV[8],PCBC,shift,plain;

 waitready();
 outportb(COMMANDPORT,COMMAND);
 waitready();

do {

    for (i = 0;i<=7;i++){        /*receive plain text*/
    waitready();
    outportb(DATAINPORT,string[4][i]);
    waitready();
    }

      while((inportb(STATUSPORT) & 0x02) != 0);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while((inportb(STATUSPORT) & 0x01) != 01);

       if(operate_mode == 0)  {

            for (i=0;i<=7;i++) {
             while((inportb(STATUSPORT) & 0x01) != 01);
             shift = inportb(DATAOUTPORT);
              if(i == 0) {
                plain = getc(ff1);
                       if(!feof(ff1)) {
                         cipher = shift^plain;
                         putc(cipher,ff2);
                       }
              }
            }
            for(i=0;i<=7;i++) {
              string[4][i] = vector[i+1];
               if(i == 7)
               string[4][i] = cipher;      /* the encrypt and decrypt is  */
                                             /* difference in this line only */
            }
        }  else {

              for (i=0;i<=7;i++) {
               while((inportb(STATUSPORT) & 0x01) != 01);
               shift = inportb(DATAOUTPORT);
                   if(i == 0) {
                     cipher = getc(ff1);
                      if(!feof(ff1)) {
                        plain = shift^cipher;
                        putc(plain,ff2);
                      }
                   }
              }
              for(i=0;i<=7;i++) {
                   string[4][i] = string[4][i+1];
                   if(i == 7)
                   string[4][i] = cipher;      /* the encrypt and decrypt is  */
              }                              /* difference in this line only */
            }
} while(!feof(ff1));
   fclose(ff1);
   fclose(ff2);
   check_complete();

}
/* OFB MODE HAS NOT THE SELF-SYNCHRONOUS PROPERTY */
OFB()

{
#define  COMMAND 0x30
char      PCBC,shift,intext,outtext;
char    /* IV[8]*/  newvector_fback[8] ; /*use vector instead of IV[8] */
int n;/* USE THE 'ofb_fback/8'instead*/

 waitready();
 outportb(COMMANDPORT,COMMAND);
 waitready();

do {

   for (i = 0;i<=7;i++){        /*receive plain text*/
   waitready();
   outportb(DATAINPORT,string[4][i]);
   waitready();
   }

     while((inportb(STATUSPORT) & 0x02) != 0);
     while((inportb(STATUSPORT) & 0x08) != 0x08);
     while((inportb(STATUSPORT) & 0x01) != 01);

            for (i=0;i<=7;i++) {
              while((inportb(STATUSPORT) & 0x01) != 01);
              shift = inportb(DATAOUTPORT);
              newvector_fback[i] = shift;
               if(i < (num_fb)) {
                 intext = getc(ff1);
                 outtext = shift^intext;
                 if(!feof(ff1)) putc(outtext,ff2);
               }
            }
   for(i=0;i<=(7-(num_fb));i++)       /* this 4 lines are for feedback */
    string[4][i] = string[4][i+(num_fb)];
                 /*   the left n bytes to the IV     */
   for(i=0;i<(num_fb);i++)
    vector[8-(num_fb)] = newvector_fback[i];

 } while(!feof(ff1));
   fclose(ff1);
   fclose(ff2);
   check_complete();

}
ENMAC1(int COMMANDO)    /* CIPHER BLOCK CHAINING (ENCRYPT) */

/* OK but if want to see a file(.reg) must use the utility view cipher file */
/* it will allow you to see the MAC(not allow by DOS 'type'program */
{

char plainmac[8];
char PCBC[8],ciphermac[8];
char mac[8];
char foreplain[8]; /*previous block of plain */
int loop = 1;

 waitready();
 outportb(COMMANDPORT,COMMANDO);
 waitready();

while(!feof(ff1)) { /* do { */
 for (i = 0;i<=7;i++){       /*receive plain text*/
      waitready();
            plainmac[i] = getc(ff1);
            PCBC[i] = plainmac[i] ^ string[4][i];
            outportb(DATAINPORT,PCBC[i]);
```

```
                waitready();
                putc(plainmac[i],ff2);
                /*collect the data in file and changed the extention */

 }

      while((inportb(STATUSPORT) & 0x02) != 0);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while((inportb(STATUSPORT) & 0x01) != 01);

    for (i=0;i<=7;i++) {
    while( (inportb(STATUSPORT) & 0x01) != 01);
    ciphermac[i] = inportb(DATAOUTPORT);

      if (loop ==1 )
           mac[i] = ciphermac[i] ;
      else
           mac[i] = ciphermac[i] ^ foreplain[i] ; /*ex-or to build MAC */

      foreplain[i] = mac[i];
      string[4][i] = ciphermac[i];
    }
  loop = loop + 1;

} /*  while(!feof(ff1)); */

    for (i=0;i<=7;i++) {
     putc(mac[i],ff2);
    }

     fclose(ff1);
     fclose(ff2);
     check_complete();
}
ENMAC2(int COMMANDO)    /* CIPHER BLOCK CHAINING (ENCRYPT) */

{
char IV[8]; /*={1,2,3,4,5,6,7,8};*/
char PCBC[8],plainmac[8],ciphermac[8];
/*char xPCBC[8],xplainmac[8],xciphermac[8];*/
char plain0mac[8],cipher0mac[8];
/*char xmac[8],xfore0plain[8],xforeplain[8];*/
char mac[8],fore00plain[8],fore0plain[8],foreplain[8],fore1plain[8];
int loop = 1;
int error_return;
 waitready();
 outportb(COMMANDPORT,COMMANDO);
 waitready();

 for(i=0;i<=7;i++) IV[i] = string[4][i];
while(!feof(ff1)) { /*do {*/

 for (i = 0;i<=7;i++){       /*receive plain text*/
      waitready();
                plain0mac[i]= plainmac[i];
                plainmac[i] = getc(ff1);
                PCBC[i] = plainmac[i] ^ IV[i];
                outportb(DATAINPORT,PCBC[i]);
                waitready();
/* if(!feof(ff1))  putc(plainmac[i],ff2); */
/*collect the data in file and changed the extention */

  }

      while((inportb(STATUSPORT) & 0x02) != 0);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while((inportb(STATUSPORT) & 0x01) != 01);

    for (i=0;i<=7;i++) {
    while( (inportb(STATUSPORT) & 0x01) != 01);
    cipher0mac[i]= ciphermac[i];
    ciphermac[i] = inportb(DATAOUTPORT);
if (loop ==1 )
           mac[i] = ciphermac[i] ;
      else
           mac[i] = ciphermac[i] ^ foreplain[i] ; /*ex-or to build MAC */

      fore00plain[i]=fore0plain[i];
      fore0plain[i]= foreplain[i];
      foreplain[i] = mac[i];
      IV[i] = ciphermac[i];
    }
  loop = loop + 1;
}
   for (i=0;i<=7;i++) {
    if((plain0mac[i] ^ fore00plain[i])) /*compare MAC */
     {
     printf("\a");
     error_return = ERROR(120,120,465,240,10,9);
     break;
    }else error_return = 0;
    }
 /*  fclose(ff1);*/
    check_complete();

/*******************************************/

if(!error_return) /* if the file has change skip the authentic process */
 {
```

```
waitready();
outportb(COMMANDPORT,COMMANDO);
waitready();

rewind(ff1); /*set file pointer to the beginning position */
loop=1;

for(i=0;i<=7;i++) IV[i] = string[4][i];
do {

    for (i = 0;i<=7;i++){        /*receive plain text*/
    waitready();
    plainmac[i] = getc(ff1);
    if(loop == 1)
      mac[i] = plainmac[i];
    else
      mac[i] = plainmac[i] ^ foreplain[i]; /* ex - or to build MAC */

    fore1plain[i]=foreplain[i];
    foreplain[i] = mac[i];  /*shift MAC to ex or with the next round */

    PCBC[i] = plainmac[i] ^ string[4][i];
    outportb(DATAINPORT,PCBC[i]);
    waitready();
    }
     loop = loop + 1;
      while((inportb(STATUSPORT) & 0x02) != 0);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while((inportb(STATUSPORT) & 0x01) != 01);

    for (i=0;i<=7;i++) {
     while((inportb(STATUSPORT) & 0x01) != 01);
     ciphermac[i] = inportb(DATAOUTPORT);
         if(!feof(ff1)) { putc(ciphermac[i],ff2);
         string[4][i] = ciphermac[i] ;
         }
    }
 } while(!feof(ff1));
      for(i=0;i<=7;i++){
         outportb(DATAINPORT,(fore1plain[i] ^ string[4][i]));
         waitready();
      }

      while((inportb(STATUSPORT) & 0x02) != 0);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while( (inportb(STATUSPORT) & 0x01) != 01);

    for (i=0;i<=7;i++) {
      while( (inportb(STATUSPORT) & 0x01) != 01);
      ciphermac[i] = inportb(DATAOUTPORT);
      putc(ciphermac[i],ff2);
      string[4][i] = ciphermac[i];
    }
     fclose(ff1);
     fclose(ff2);
     check_complete();
}
fclose(ff1);
}

DEMAC(int COMMANDO)     /* CIPHER BLOCK CHAINING (DECRYPT)*/

{
int temp;
char IV[8]; /* = {1,2,3,4,5,6,7,8};*/
char PCBC[8];
char xcipher[8],xplain[8];
char fore0PCBC[8],forePCBC[8],mac[8];
char clear_data_in_DEUbuffer[8];
int loop = 1;

waitready();
 outportb(COMMANDPORT,COMMANDO);
 waitready();

while(!feof(ff1))     /*do { */
{
    for (i = 0;i<=7;i++){        /*receive plain text*/
    waitready();
    xcipher[i] = getc(ff1);
/*    PCBC = xplain ^ string[4][i]; */
    outportb(DATAINPORT,xcipher[i]);
    waitready();
    }
      while((inportb(STATUSPORT) & 0x02) != 0);
      while((inportb(STATUSPORT) & 0x08) != 0x08);
      while( (inportb(STATUSPORT) & 0x01) != 01);

           for (i=0;i<=7;i++) {
            while( (inportb(STATUSPORT) & 0x01) != 01);
            xplain[i] = inportb(DATAOUTPORT);

             if(!feof(ff1))
              {
              PCBC[i] = xplain[i] ^ string[4][i];
              putc(PCBC[i],ff2);
              string[4][i]=xcipher[i];
                if(loop == 1)
                     mac[i] = PCBC[i];
                else
```

```
                    mac[i] = PCBC[i] ^ forePCBC[i];

              fore0PCBC[i] = forePCBC[i];
              forePCBC[i]  = mac[i];
             }
            else
             {
              if((PCBC[i] ^ fore0PCBC[i])) /*compare MAC */
                {
                 printf("\a");
                 ERROR(120,120,465,240,11,10);
                     for (i=0;i<=6;i++) {/*only 6 cause the first has read above*/
                      while( (inportb(STATUSPORT) & 0x01) != 01);
                      clear_data_in_DEUbuffer[i] = inportb(DATAOUTPORT);
                 }
                 break;
                }
             }
      }
           loop = loop +1;
}
    fclose(ff1);
    fclose(ff2);
    check_complete();
/*  cbc_entime = difftime(end,start);*/ /*ENTIME AND DETIME IS ASSUME EQUAL */

}
NOT_USED_DEMAC()
{
int toggle;

   window_y1(150,100,475,256,10);
   setfillstyle(1,7);
   bar3d(164,124,455,142,0,0);
   setcolor(6);
   outtextxy(246,129,"Messages to User");
   setcolor(9);
   outtextxy(170,154,"This menu is not used in receiving");
   outtextxy(170,170,"an authenticated file because a MAC");
   outtextxy(170,186,"is already existed so not necessary");
   outtextxy(170,202,"to create again just only check it.");
   setcolor(5);
   outtextxy(170,218,"        Continue by ESC key");
    do {
        (toggle=getch());
       } while (toggle != 27); /*ESC KEY== 27*/
   Restoreview(150,100,10);
}
char *key_reassign(char *temp)
{
    int i;
    for(i=0;i<=7;i++)

    switch (temp[i]){
    case    '!' : temp[i] =  0x80 ;break;
    case    '"' : temp[i] =  0x83 ;break;
    case    '$' : temp[i] =  0x85 ;break;
    case    '(' : temp[i] =  0x89 ;break;
    case    '+' : temp[i] =  0x8a ;break;
    case    '-' : temp[i] =  0x8c ;break;
    case    '.' : temp[i] =  0x8f ;break;
    case    '0' : temp[i] =  0x91 ;break;
    case    '3' : temp[i] =  0x92 ;break;
    case    '5' : temp[i] =  0x94 ;break;
    case    '6' : temp[i] =  0x97 ;break;
    case    '9' : temp[i] =  0x98 ;break;
    case    ':' : temp[i] =  0x9b ;break;
    case    '<' : temp[i] =  0x9d ;break;
    case    '?' : temp[i] =  0x9e ;break;
    case    'A' : temp[i] =  0xa1 ;break;
    case    'B' : temp[i] =  0xa2 ;break;
    case    'D' : temp[i] =  0xa4 ;break;
    case    'G' : temp[i] =  0xa7 ;break;
    case    'H' : temp[i] =  0xa8 ;break;
    case    'K' : temp[i] =  0xab ;break;
    case    'M' : temp[i] =  0xad ;break;
    case    'N' : temp[i] =  0xae ;break;
    case    'P' : temp[i] =  0xb0 ;break;
    case    'S' : temp[i] =  0xb3 ;break;
    case    'U' : temp[i] =  0xb5 ;break;
    case    'V' : temp[i] =  0xb6 ;break;
    case    'Y' : temp[i] =  0xb9 ;break;
    case    'Z' : temp[i] =  0xba ;break;
    case    '_' : temp[i] =  0xbf ;break;
    case    'c' : temp[i] =  0xc1 ;break;
    case    'e' : temp[i] =  0xc2 ;break;
    case    'f' : temp[i] =  0xc4 ;break;
    case    'i' : temp[i] =  0xc7 ;break;
    case    'j' : temp[i] =  0xc8 ;break;
    case    'l' : temp[i] =  0xcb ;break;
    case    'o' : temp[i] =  0xcd ;break;
    case    'q' : temp[i] =  0xce ;break;
    case    'r' : temp[i] =  0xd0 ;break;
    case    't' : temp[i] =  0xd3 ;break;
    case    'w' : temp[i] =  0xd5 ;break;
    case    'x' : temp[i] =  0xd6 ;break;
    case    '{' : temp[i] =  0xd9 ;break;
    case    '}' : temp[i] =  0xda ;break;
  }
    return temp;

  }
```

หอสมุดกลาง สถาบันวิทยบริการ
จุฬาลงกรณ์มหาวิทยาลัย

Chulalinet
3 0021 00095027 9

# ประวัติผู้เขียน

นาย นริศ รังษีนพมาศ เกิดเมื่อวันที่ 7 มีนาคม พ.ศ.2509 ที่กรุงเทพฯ จบการศึกษาขั้นอุดมศึกษา จาก คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า พระนครเหนือ ได้รับปริญญาวิศวกรรมศาสตรบัณฑิต สาขาวิศวกรรมไฟฟ้าในปีการศึกษา 2533 และเข้าศึกษาต่อในหลักสูตรวิศวกรรมศาสตรมหาบัณฑิตที่ จุฬาลงกรณ์มหาวิทยาลัย เมื่อ พ.ศ.2534 ปัจจุบันเป็นอาจารย์ประจำภาควิชาวิศวกรรมไฟฟ้า คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้า พระนครเหนือ