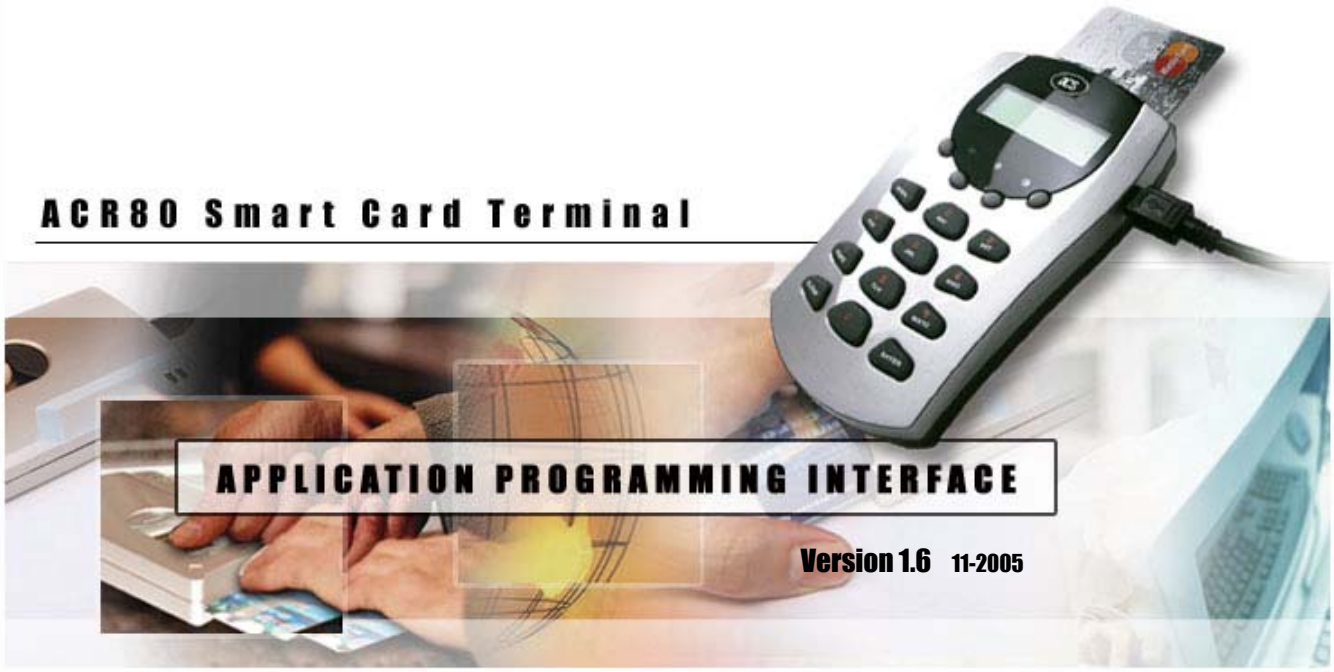
Advanced Card Systems Ltd.

ACR80 Smart Card Terminal



APPLICATION PROGRAMMING INTERFACE

Version 1.6 11-2005

Unit 1008, 10th Floor, Hongkong International Trade and Exhibition Centre
1 Trademart Drive, Kowloon Bay, Hong Kong
Tel: +852 2796 7873 Fax: +852 2796 1286 Email: info@acs.com.hk Website: www.acs.com.hk

Contents

1. INTRODUCTION

This manual describes the use of interface software (ACR80.dll) to program the ACR80 smart card reader. This interface software is a set of library functions implemented for the application programmers to operate the ACR80 smart card reader and the inserted smart cards. Currently, it is supporting a number of commonly used platforms and development tools.

ACR80 communicates with the PC via a RS232 or USB interface and the command / response messages between these two entities are relatively complex. This interface software is responsible for handling all these communication details, parameter conversions, error handling and gives the ACR80 reader the application programmer a clear and easy to use functional interface as possible. The schematic diagram is as follows.

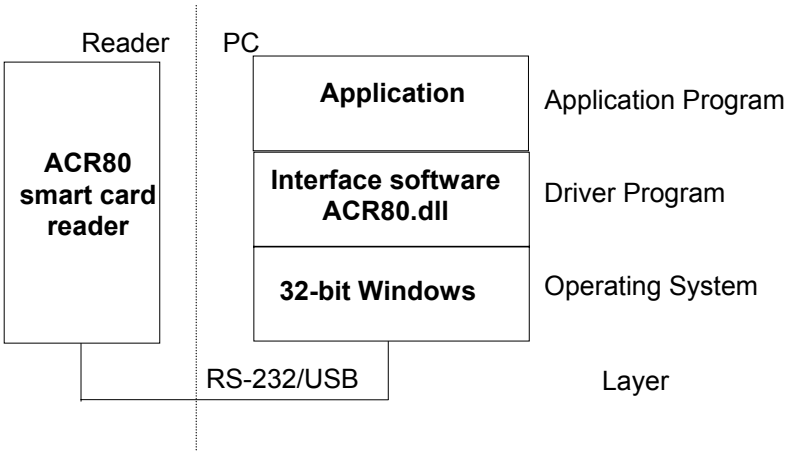


Figure 1.1

2. ACR80

2.1 Overview

The ACR80 Smart Card Reader/Writer Terminal is an interface for the communication between a computer (for example, a PC) and a smart card. Different types of smart cards have different commands and different communication protocols. This, in most cases, prohibits the direct communication between a smart card and a computer. The ACR80 Reader/Writer establishes a uniform interface from the computer to the smart card for a wide variety of cards. By taking care of the card specific particulars, it saves the computer software programmer from getting involved with the technical details of the smart card operation, which are in many cases not relevant for the implementation of a smart card system. The ACR80 Smart Card Reader/Writer Terminal is connected to the computer through a serial asynchronous interface (RS-232) or an USB interface. The reader accepts commands from the computer, carries out the specified function on the smart card and returns the requested data or status information.

2.2 Status LEDs

Three LEDs on the front of the reader indicate the presence of the power supply to the reader and the activation status of the two smart card interfaces:

Green LED Indicates 5V power supply to the reader is present.

Red/Green LED Indicates the status on the corresponding smart card slot (for slot 1 and slot 2 only):
Green: card in slot is powered on.
Yellow: card in slot is being accessed.
Red: any error detected at the slot.

2.3 Slot Definition

The slots are numbered as follows:

Figure 1.2 Top View

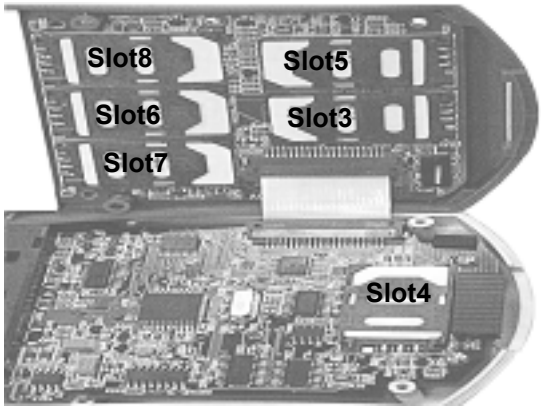


Figure 1.3 Inside View

3. ACR80 API

3.1 Overview

ACR80 API (ACR80.dll) is a set of high-level functions provided for the implementation of the application software. It provides a consistent programming interface for the application to operate on the card reader and the corresponding inserted cards. It communicates with the ACR80 reader via the communication port facilities provided by the operating system. ACR80 API is supposed to be platform independent provided that there is a minor modification on the communication module of the API to adapt to different operating environment.

The ACR80 Application Programming Interface (API) defines a common way of accessing the ACR80 reader. Application programs invoke it through the interface functions and perform operations on the inserted card through the use of ACI commands. The header file ACR80.H is available for the program developer, which contains all the function prototypes and macros described below, as well as a list of Constants that can be used together with the commands.

3.2 Interface Data Structures

The ACR80 API makes use of several data structures to pass parameters between application programs and the library driver. These data structures are defined in the header file ACR80.H and they are discussed below:

3.2.1 AC_READER_INFO

```
typedef struct {
      BYTE  szRev[10];         // The 10 bytes firmware type and
                               // revision code
      INT16 nMaxC;             // Maximum number of command data bytes
      INT16 nMaxR;             // Maximum number of data bytes that
                               // can be requested in a response
      INT16 CType;             // card types supported by the reader
      INT16  nLibVer;          // Library version
      WORD32 lBaudRate;        // Current Running Baud Rate
      BYTE  exchangeMode;      // APDU exchange mode (ISO or EMV)
      BYTE  clockStatus;       // The status of the clock
                               // (0:running and 1:error)
      BYTE  BackLightStatus;   // On=1 or Off=0
      BYTE  BuzzerStatus;      // On=1 or Off=0
} AC_READER_INFO;
```

The AC_READER_INFO data structure is used in the AC_Get_Reader_Info function call for the retrieval of reader related information. Their meanings are described as follows:

Name	Input/Output	Description
nMaxC	Output	The maximum number of command data byte (DataIn) that can be accepted in an ExchangeAPDU command
nMaxR	Output	The maximum number of response data byte (DataOut) returned after an ExchangeAPDU command
CType	Output	The card types supported by the reader (reports MCU cards only)
szRev[10]	Output	The firmware revision code
nLibVer	Output	Library version (e.g. 310 is equal to version 3.10)
lBaudRate	Output	Current running baud rate
exchangeMode	Output	Current APDU exchange mode (ISO=2 or EMV=1)
ClockStatus	Output	Status of the clock (running = 0, error = 1)
BackLightStatus	Output	Status of the LCD backlight (LCD_OFF=0, LCD_ON=1)
BuzzerStatus	Output	Status of the LCD backlight (BUZZER_OFF=0, BUZZER_ON=1)

3.2.2 AC_SLOT_INFO

```
typedef     struct {
      BYTE  CStat;      // The status of the card slot
      BYTE  CSel;       // The current selection of card type for the
                        // slot
} AC_SLOT_INFO;
```

The AC_SLOT_INFO data structure is used in the AC_Get_Slot_Info function call for the retrieval of card slot related information. Their meanings are described as follows:

Name	Input/Output	Description
CStat	Output	The status of the card slot 0 = no card inserted 1 = card inserted, not powered up 2 = card powered up
CSel	Output	The current selection of the card type for the slot

3.2.3 AC_READER_CLOCK

```
typedef struct {
      BYTE  year        //0-99
      BYTE  month       //1-12
      BYTE  day         //0-31
      BYTE  weekday     //1-7
      BYTE  Hour        //0-23
      BYTE  minute      //0-59
      BYTE  second      //0-59
} AC_READER CLOCK;
```

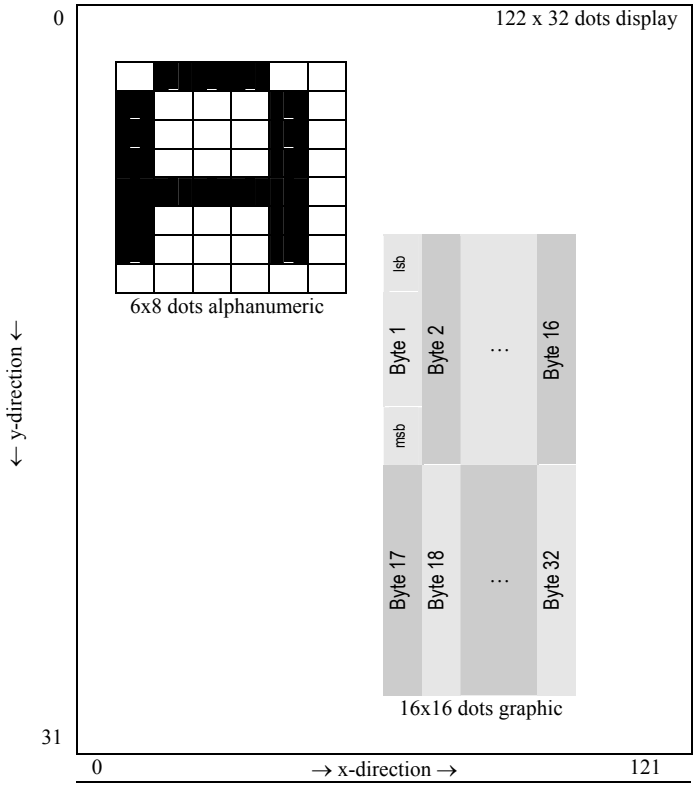
The AC_READER_CLOCK data structure is used in AC_Format_Reader_Clock function call to retrieve/set the raw format of reader built-in real time clock.

Name	Input/Output	Description
year	Input/Output	0 – 99 starts with 0 = 2000 (Binary Coded Decimal)
month	Input/Output	1 – 12 (Binary Coded Decimal)
day	Input/Output	0 – 31 (Binary Coded Decimal)
weekday	Input/Output	1 – 7 (Binary Coded Decimal)
hour	Input/Output	0 – 23 (Binary Coded Decimal)
minute	Input/Output	0 – 59 (Binary Coded Decimal)
second	Input/Output	0 – 59 (Binary Coded Decimal)

3.2.4 AC_GRAPHIC

```
typedef struct {
      BYTE  startXPos         // starting position on x-coordinate
      BYTE  startYPos         // starting position on y-coordinate
      BYTE  type              // type of graphic
      BYTE  map[488];         // ACSII string or bit-map of
                              // graphic
} AC_GRAPHIC;
```

The AC_GRAPHIC data structure is used in the AC_Draw function call for the representation of an ASCII string or a bit-map graphic to be displayed on the LCD of the reader:



Name	Input/Output	Description
startXPos	Input	The starting position (upper-left corner) of the graphic on the x-coordinate (00_H – 79_H)
startYPos	Input	The starting position (upper-left corner) of the graphic on the y-coordinate (00_H to $1F_H$)
type	Input	Type of graphic AC_ALPHANUMERIC = an 6X8 dots ASCII character alpha AC_MAP = a 16x16 dots graphic AC_FULL_MAP = a 122x32 dots graphic
map	Input	The array of bit-map or ASCII code. For array of bit-map, the byte sequence should be formatted as shown in the figure above.

3.2.5 AC_SESSION

```
typedef struct {
      BYTE  CardType;          // Card type selected
      BYTE  CardSlot;          // Selected card slot
      BYTE  ATRLen;            // Length of the ATR
      BYTE  ATR[128];          // ATR string
      BYTE  Protocol;          // Protocol supported by the card
      BYTE  ActiveProtocol;    // Currently selected protocol
      BYTE  HistLen;           // Length of the Historical data
      BYTE  HistOffset;        // Offset of the Historical data
                               // from the beginning of ATR
      INT16 APDULenMax;        // Max. APDU supported
} AC_SESSION;
```

The AC_SESSION data structure is used in the AC_StartSession function call for the retrieval of ATR information from the smart card. Before calling AC_StartSession, the program needs to specify the value of CardType. After calling the function, the ATR string can be found in ATR field and the length is stored in ATRLen.

Name	Input/Output	Description
CardType	Input	The card type selected for operation (preferred card protocol) AC_T0 = T0 protocol preferred AC_T1 = T1 protocol preferred AC_AUTO = T0 or T1 protocol automatically selected by the card AC_MC_SLE4428 = Siemens SLE 4418/4428 Intelligent 1K Byte Memory Card preferred AC_MC_SLE4442 = Siemens SLE 4432/4442 Intelligent 256 Byte Memory Card preferred
CardSlot	Input	The card slot selected for operation
ATRLen	Output	Length of the ATR string
ATR	Output	Answer to reset (ATR) string
Protocol	Output	The protocol(s) supported by the card AC_T0 = supports only T0 protocol

		AC_T1 = supports only T1 protocol AC_DUAL = supports both T0 and T1 protocol AC_MC_SLE4428 = supports SLE 4418/4428 memory card AC_MC_SLE4442 = supports SLE 4432/4442 memory card
ActiveProtocol	Output	The protocol this session selected to use for further transmission AC_T0 = T0 protocol selected AC_T1 = T1 protocol selected AC_MC_SLE4428 = SLE 4418/4428 memory card selected AC_MC_SLE4442 = SLE 4432/4442 memory card selected
HistLen	Output	Obsolete field – not used anymore
HistOffset	Output	Obsolete field – not used anymore
APDULenMax	Output	Obsolete field – not used anymore

3.2.6 AC_APDU

```
typedef struct {
      BYTE        CLA;
      BYTE        INS;
      BYTE        P1;
      BYTE        P2;
      INT16       Lc;
      INT16       Le;
      BYTE        DataIn[256];
      BYTE        DataOut[256];
      WORD16      Status;
} AC_APDU;
```

The AC_APDU data structure is used in the AC_ExchangeAPDU function for the passing of commands and data information into the smart card. For MCU card (T=0,T=1) operations, these values are specific to the smart card operating system. You must have the card reference manual before you can perform any valid operations on the card. Please notice that Lc representing the data length going into the card and Le representing the data length expecting from the card. Le should be set to –1 if only data-in operation is needed and Lc should be set to –1 if only data-out operation is needed. Le and Lc should be both set to –1 if no data - in and no data-out operation is needed (if a command is issued to ask reader to do something e.g. Clear Card command in ACOS Card).

Name	Input/Output	Description
CLA	Input	Instruction Class
INS	Input	Instruction Code
P1	Input	Parameter 1
P2	Input	Parameter 2
Lc	Input	Length of command data (DataIn) (Lc must be set to –1 if no DataIn)
Le	Input/Output	Length of response data (DataOut) (Le must be set to –1 if no DataOut)
DataIn	Input	Command data buffer

DataOut	Output	Response data buffer
Status	Output	Execution status of the command (status returned by the card)

***Notes:**

1) **Lc must be set to –1 if there is no DataIn.**
2) **Le must be set to –1 if there is no DataOut.**

3.2.7 AC_MAGNETIC_CARD

```
Typedef struct {
      BYTE  track1len;//Length of Track1 data read from magnetic card
      BYTE  track2len;//Length of Track2 data read from magnetic card
      BYTE  track3len;//Length of Track3 data read from magnetic card
      BYTE  track1[20]; //Data buffer to store data read from track 1
      BYTE  track2[60]; //Data buffer to store data read from track 2
      BYTE  track3[160];//Data buffer to store data read from track 3
} AC_MAGNETIC_CARD;
```

The AC_MAGNETIC_CARD data structure is used to store data read from each of 3 tracks on the magnetic card and also the respective length of data stored on each track.

Name	Input/Output	Description
track1len	Output	Length of the data from card track 1
track2len	Output	Length of the data from card track 2
track3len	Output	Length of the data from card track 3
track1	Output	Data from the card track 1
track1	Output	Data from the card track 2
track3	Output	Data from the card track 3

3.2.8 AC_PIN_APDU

```
typedef struct {
      BYTE        CLA;
      BYTE        INS;
      BYTE        P1;
      BYTE        P2;
      INT16       Lc;
      INT16       Le;
      BYTE        DataIn[256];
      BYTE        DataOut[256];
      WORD16      Status;
      BYTE        PIN_Length;
```

```
        BYTE        PIN_Position;
        BYTE        PIN_Coded_Scheme;
        BYTE        PIN_SD_Position;
        BYTE        PIN_Length_Counter;
        BYTE        Length_Counter_Position;
        BYTE        Timeout;
} AC_PIN_APDU;
```

The AC_PIN_APDU data structure is used in the AC_Exchange_PIN_APDU function for the passing of commands and data information into the smart card. For MCU card (T=0,T=1) operations, these values are specific to the smart card operating system. You must have the card reference manual before you can perform any valid operations on the card. Please notice that Lc representing the data length going into the card and Le representing the data length expecting from the card. To use this command, the Lc should be at least the length of the PIN and the DataIn can be set as any values (e.g. 00) which will be replaced by the PIN later.

Name	Input/Output	Description
CLA	Input	Instruction Class
INS	Input	Instruction Code
P1	Input	Parameter 1
P2	Input	Parameter 2
Lc	Input	Length of command data (DataIn) (Lc must be > 0) Remark: Lc includes the data into the card plus the pin.
Le	Input/Output	Length of response data (DataOut) (Le must be set to –1 if no DataOut)
DataIn	Input	Command data buffer
DataOut	Output	Response data buffer
Status	Output	Execution status of the command (status returned by the card)
PIN_Length	Input	Length of the PIN
PIN_Position	Input	The position in the databytes, counted from Byte=1
PIN_Coded_Scheme	Input	0 : PIN is coded as ASCII (e.g. PIN entered in terminal = “1234” will become “31 32 33 34”) 1 : PIN is coded as binary (e.g. PIN entered in terminal = “1234” will become “01 02 03 04”)
PIN_SD_Position	Input	0 : MSD (most significant digit) to the left side in the PIN position (e.g. PIN length = 4 and Lc = 6, DataIn = 01 02 03 04 05 06, and PIN position = 1, the PIN will be 01 02 03 04) 1 : LSD (least significant digit) to the right side in the PIN position (e.g. PIN length = 4 and Lc = 6, DataIn = 01 02 03 04 05 06, the PIN will be 03 04 05 06) Remark: if PIN Length = Lc, there will not have any difference.
PIIN_Length_Counter	Input	0 : without length counter 1 : with length counter
Length_Counter_Position	Input	0 : the length counter is placed as the most left byte in the PIN position 1 : the length counter is placed as next following byte left from the PIN
Timeout	Input	1 – 63 minutes (in HEX 00 – 3F). If 0 is chosen, the default timeout (1minute) is used.

3.3 Reader Related Functions

Several functions are available to obtain info about the reader and the card slots. Next to that, functions are available to control the LCD screen, buzzer, the internal clock, obtaining a key from the keypad and reading or writing to the internal EEPROM of the reader.

3.3.1 AC_Open

This function opens a port and returns a valid reader handle for the application program.
The reader handle should be retrieved before any other operation can be done.

Format:

```
INT16 AC_DECL AC_Open (INT16 ReaderType, INT16 ReaderPort);
```

Input Parameters:

The table below lists the parameters for this function.

Parameters	Definition / Values
ReaderType	Reader type, *ACR80* = Target reader is ACR80
ReaderPort	The port that the reader is connected to. *AC_COM1* = Standard Communication Port 1 *AC_COM2* = Standard Communication Port 2 *AC_COM3* = Standard Communication Port 3 *AC_COM4* = Standard Communication Port 4 *AC_COM5* = Standard Communication Port 5 *AC_USB1* = Universal Serial Bus Port 1 *AC_USB2* = Universal Serial Bus Port 2 *AC_USB3* = Universal Serial Bus Port 3 *AC_USB4* = Universal Serial Bus Port 4

Returns:

The return value is negative and contains the error code when the function encounters an error during operation. Otherwise, it returns a valid reader handle. Please refer to appendix A for the detailed description and meaning of the error codes.

Example:

```
// Open a port to a ACR80 reader connected to COM1
INT16 hReader;
hReader = AC_Open(ACR80, AC_COM1);
```

3.3.2 AC_Get_Reader_Info

This function retrieves information about the ACR80 and stores them in the AC_READER_INFO–data structure (see also chapter 3.2.1).

Format:

```
INT16 AC_DECL AC_Get_Reader_Info (INT16 hReader,
                                  AC_READER_INFO *Info);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
Info.szRev	Revision code for the selected reader.
Info.nMaxC	The maximum number of command data bytes
Info.nMaxR	The maximum number of data bytes that can be requested to be transmitted in a response
Info.CType	The card types supported by this reader
Info.nLibVer	Current library version (e.g. 310 is equal to version 3.10)
Info.lBaudRate	The current running baud rate
Info.exchangeMode	The mode of exchange APDU (EMV=1, ISO=2)
Info.clockStatus	The status of the clock (running=0, error=1)
Info.BackLightStatus	Status of the LCD backlight (LCD_OFF=0, LCD_ON=1)
Info.BuzzerStatus	Status of the LCD backlight (BUZZER_OFF=0, BUZZER_ON=1)

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

Example:

```
//Get the revisioncode and baud rate of the currently selected reader
INT16 RtnCode;
AC_READER_INFO Info;

RtnCode = AC_Get_Reader_Info(hReader, &Info);
printf ("Reader Operating System ID : %s", Info.szRev);
printf ("Baud Rate: %ld", Info.lBaudRate);
```

3.3.3 AC_Get_Slot_Info

This function retrieves information related to the currently selected card slot of the currently selected reader and stores it in the AC_SLOT_INFO-data structure (see also chapter 3.2.2).

Format :

```
INT16 AC_DECL AC_Get_Slot_Info (INT16 hReader, BYTE slot,
                                AC_SLOT_INFO *Info);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()
slot	The selected card slot

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
Info.CStat	The current status of the slot 0 = no card inserted 1 = card inserted, not powered up 2 = card powered up Note: for slot 3-8, the function always assumes that a card is inserted.
Info.CSel	The current selection of the card type for the slot

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

Example:

```
// Get the specified slot status of the currently selected reader
AC_SLOT_INFO Info;
INT16 RtnCode;
char status[3][16] = {"No Card","Card Inserted","Card Powered Up" };

RtnCode = AC_Get_Slot_Info(hReader, AC_SLOT_1, &Info);
printf ("Status Slot 1: %s", asi.CStat > 0 && asi.CStat < 2 ?
        status[asi.CStat] : "Unknown");
```

3.3.4 AC_SetOptions

This function sets various options for the reader.

Format:

```
INT16 AC_DECL AC_SetOptions (INT16 hReader, WORD16 Type,
                             WORD16 Value);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open() (except for the ACO_RESET_READER option)
Type	Type of option that is going to set
Value	Value parameter for the selected option type

Options:

Options	Type	Value
Set the communication baud rate with the reader	ACO_BAUDRATE	BAUDRATE_19200 = 19200 bps BAUDRATE_9600 = 9600 bps
Turn on/off LCD backlight	ACO_LCD_LIGHT	LCD_OFF (0) = turn off LCD_ON (1) = turn on
Turn on/off buzzer	ACO_BUZZZER_VOLUME	BUZZER_OFF (0) = turn off BUZZER_ON (1) = turn on
Select APDU exchange mode	ACO_EXCHANGE_MODE	EMVMode (1) = EMV mode ISOMode (2) = ISO mode
Control the status of each LED light	ACO_LED	Bit 0 to 1 = LED for card slot 1(b1 b0) Bit 2 to 3 = LED for card slot 2(b3 b2) Bit 4 to 7 are reserved 00 = no light 01= green light 10 = red light 11 = yellow light

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.5 AC_Format_Reader_Clock

This function is used to set or get the internal real time clock of reader depending on the value of 'options' using the reader clock structure (see also paragraph 3.2.3).

Format :

```
INT16 AC_DECL AC_Format_Reader_Clock  (INT16 hReader, INT16 options,
                                       AC_READER_CLOCK *ReaderCLOCK);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
HReader	Handle returned by AC_Open()
Options	GETCLOCK (0) = Get Clock SETCLOCK (1) = Set Clock
ReaderCLOCK	Store the human-readable format of the reader internal RTC For Get Clock, provide this buffer and the clock value will be stored in the structure. For Set Clock, fill this structure with clock values and call the function.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.6 AC_Get_Single_Key

This function requests the reader to detect a single key press event on its keyboard. A key event is recognized when a key is pressed. This function will block further execution until a key press is detected or a timeout occurs.

Format:

```
INT16 AC_DECL AC_Get_Single_Key (INT16 hReader, INT16 timeout);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
HReader	A valid reader handle returned by AC_Open()
Timeout	Reader will abort this key request if no key is pressed within this specified timeout time. The unit is second. (The timeout may not be accurate because it doesn't include the communication time, Therefore, the input timeout may differ from the actual timeout.) **A timeout of 0 disables the timeout.**

Returns:

The return value is negative and contains the error code when the function encounters timeout or an error during operation. Otherwise, it returns a value of the key being pressed. Please refer to appendix A for the detailed description and meaning of the error codes.

Example:

```
// Get a key from the keyboard of the reader with a timeout of 60
// seconds
INT16 Key;

Key = AC_Get_Single_Key(hReader, 60);
if (key > 0) {
      printf("key %d pressed.", Key);
}
else if (key = -1132) {
      printf("Keypad Timeout occurred.");
}
```

3.3.7 AC_Draw

This function draws a graphic or alphanumeric character to the display RAM of the reader but does not display it on the LCD-screen of the ACR80.

Format:

```
INT16 AC_DECL AC_Draw (INT16 hReader, AC_GRAPHIC *Gfx);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()
Gfx.startXPos	Starting page of the string or graphic
Gfx.startYPos	Starting segment of the string or graphic
Gfx.type	Graphic type: AC_ALPHANUMERIC = ASCII character AC_MAP = 16x16 dots bit-map graphic AC_FULL_MAP = 122x32 dots full bit-map graphic
Gfx.map	If Gfx.type is AC_MAP, this is the pointer to a 32-byte array containing the bit-map of the graphic. If Gfx.type is AC_FULL_MAP, this is the pointer to a 488-byte array containing the bit-map of the full graphic. If Gfx.type is AC_ALPHANUMERIC, this is the pointer to a 1-byte array containing the ASCII code of the alphanumeric chanracter.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.8 AC_Display

This function instructs the reader to display the content of its display RAM to the LCD immediately.

Format:

```
INT16 AC_DECL AC_Display (INT16 hReader);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.9 AC_Display_String

This function is used to display a text on one of the 4 pages of the reader.

Format:

```
INT16 AC_DECL AC_Display_String (INT16 hReader, BYTE page, BYTE offset,
                                 BYTE LCDCharacterSize, BYTE dataLength,
                                 BYTE* characters);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
page	0-3: the page where the text will be displayed Note: If the LCD character size is set to double size the characters displayed will occupy 2 pages, only page 1 and 3 can be used.
Offset	0-121: start-pixel on the page to write the text
dataLength	The number of characters in the input characters buffer
LCDCharacterSize	Single or double character height and width for the page being displayed (SINGLESIZE, DOUBLESIZE)
characters	Buffer containing the ASCII codes of the characters to display in the LCD.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

Example:

//Display a string in the first page, with an offset of 8 pixels.

```
INT16 RtnCode;
char characters[6] = "ACR80";

RtnCode = AC_Display_String (hReader, 0, 8, strlen(characters), (BYTE
*)characters);
```

3.3.10 AC_Clear_LCD_Pages

This function is used to clear the LCD screen of the reader. The LCD-screen consists of 4 pages. Each of those pages can be cleared with this function.

Format:

```
INT16 AC_DECL AC_Clear_LCD_Pages  (INT16 hReader, INT16 pages);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
Hreader	Handle returned by AC_Open()
Pages	1-15: Each bit of this byte represents the action on a page: Bit 0 = Page 0 to be cleared (1) or left unchanged (0) Bit 1 = Page 1 to be cleared (1) or left unchanged (0) Bit 2 = Page 2 to be cleared (1) or left unchanged (0) Bit 3 = Page 3 to be cleared (1) or left unchanged (0) Bit 4 to 7 are reserved, values are ignored.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.11 AC_Read_EEPROM

This command reads the content of the built-in EEPROM on the reader.

Format:

```
INT16 AC_DECL AC_Read_EEPROM(INT16 hReader, BYTE AdrHI, BYTE AdrLOW,
                             INT16 &dataOutLen, BYTE* dataOut);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
AdrHI	The HI byte of the EEPROM address
AdrLOW	The LOW byte of the EEPROM address
dataOutLen	Number of bytes to get from the EEPROM. From 1 - 256.
dataOut	Data buffer to store the returned data

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
dataOutLen	Number of bytes read from the EEPROM
dataOut	Data read from the EEPROM

Remark:

The EEPROM address is from 0400_H -- $7FFF_H$

Reading from an address below 0400_H will be denied!

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.12 AC_Write_EEPROM

This command writes data to the built-in EEPROM on the reader.

Format:

```
INT16 AC_DECL AC_Write_EEPROM(INT16 hReader, BYTE AdrHI, BYTE AdrLOW,
                              INT16 dataInLen, BYTE* dataIn);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
AdrHI	The HI byte of the EEPROM address
AdrLOW	The LOW byte of the EEPROM address
dataInLen	Number of bytes to store in the EEPROM. From 1 - 256.
dataIn	Data buffer to send

Remark:

The EEPROM address is from 0400_H -- $7FFF_H$

Writing to an address below 0400_H will be denied!

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.13 AC_DirectCommand

This function is used to exchange data and command codes directly with the reader. In order to use this function, a Smart Card Terminal Reference Manual, like the ACOS138 Manual, for the instruction codes of the reader functions and the data required is needed.

Format:

```
INT16 AC_DECL AC_DirectCommand(INT16 hReader, BYTE INS,
                               short dataInLen, BYTE* inData,
                               short &dataOutLen, BYTE* outData);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
Hreader	Handle returned by AC_Open()
INS	The instruction code of reader command
DataInLen	The data length of the inData
InData	The input data storage
dataOutLen	The data length of the outData
OutData	The output data storage

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.14 AC_GetSendRecvBuffer

This function is used to check the data in the receive- and send-buffers of the COM-port (or USB-port). This data will contain the complete message sent to or received from the reader (including STX, ETX, checksum etc.) For debugging purposes only.

Format:

```
INT16 AC_DECL AC_GetSendRecvBuffer (INT16 hReader, int &sendLen,
                                    BYTE* sendBuff, int &recvLen,
                                    BYTE* recvBuff)
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hreader	Handle returned by AC_Open()
sendLen	The data length of the sendBuff
sendBuff	The data buffer of the sent buffer from the driver
recvLen	The data length of the recvBuff
recvBuff	The data buffer of the received buffer from the driver

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.3.15 AC_Close

This function closes a previously opened reader port.
Handle should be closed after all operations are done.

Format:

```
INT16 AC_DECL AC_Close (INT16 hReader);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle previously opened by AC_Open()

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.4 Card Control Functions

The ACR80 API has a set of functions to exchange data between the reader and an inserted card. Generally, a program is required to call AC_Open first to obtain a handle to the ACR80 (see also chapter 3.3.1). The handle is required for subsequent calls to AC_StartSession to initialize the card. AC_ExchangeAPDU and AC_Read_Magnetic_Card are used for issuing commands and exchanging data. And eventually AC_EndSession has to be called to power down an card and AC_Close (see also chapter 3.3.13) to close the connection to the reader.

For an overview of commands for the ACOS1 Smart Card, please refer to the ACOS138 Manual (to be found in the same folder as this document) and Appendix B for a list of commands for the SLE4428 and SLE 4442-memory cards. To see the commands in action, please refer to the test programs and demos, provided on the ACR80 SDK CD.

*Notes:

1) Memory cards can only be read and written in Slot 1 and Slot 4.
2) Magnetic cards can only be read with an optional magnetic card module.

3.4.1 AC_StartSession

This function starts a session with a selected card type and updates the AC_SESSION data- structure (see also chapter 3.2.5) with the values returned by the card Answer-To-Reset (ATR). A session is started by a card reset and it is ended by either another card reset, a power down of the card or the removal of a card from the reader. Note that this function will power up the card and perform a card reset.

Format:

```
INT16 AC_DECL AC_StartSession (INT16 hReader, AC_SESSION *Session);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()
Session.CardType	The card type selected for operation (preferred card protocol) AC_T0 = T0 protocol preferred AC_T1 = T1 protocol preferred AC_AUTO = T0 or T1 protocol automatically selected by the card AC_MC_SLE4428 = Siemens SLE 4418/4428 Intelligent 1K Byte Memory Card preferred AC_MC_SLE4442 = Siemens SLE 4432/4442 Intelligent 256 Byte Memory Card preferred
Session.CardSlot	The selected card slot. AC_SLOT_1 = Card slot 1 AC_SLOT_2 = Card slot 2

Parameters	Definition / Values
	AC_SLOT_3 = Card slot 3 AC_SLOT_4 = Card slot 4 AC_SLOT_5 = Card slot 4 AC_SLOT_6 = Card slot 4 AC_SLOT_7 = Card slot 4 AC_SLOT_8 = Card slot 4

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
Session.ATR	Answer to Reset returned by the card
Session.ATRLen	Length of the answer to reset
Session.HistLen	Length of the historical data
Session.HistOffset	Offset of the historical data
Session.APDULenMax	Maximum length of APDU supported

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

Example:

```
// Start a new session
AC_SESSION  *session;
INT16 RtnCode;

session->CardType = AC_AUTO;
session->CardSlot = AC_SLOT1;
RtnCode = AC_StartSession(hReader, session);
```

3.4.2 AC_EndSession

This function ends a previously started session and powers off the card.

Format:

```
INT16 AC_DECL AC_EndSession (INT16 hReader, BYTE slot);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	A valid reader handle returned by AC_Open()
slot	The selected card slot

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

Example:

```
//End session on a previously started session
RtnCode = AC_EndSession(hReader, Session.CardSlot);
```

3.4.3 AC_ExchangeAPDU

This function sends an APDU command to a card via the opened port and returns the card's response. The AC_APDU data-structure is used to exchange the data (see also chapter 3.2.6). For an overview of commands for the ACOS1 Smart Card, please refer to the ACOS138 Manual (to be found in the same folder as this document), see also Appendix B for a list of commands for the SLE4428 and SLE 4442-memory cards.

Format:

```
INT16 AC_DECL AC_ExchangeAPDU (INT16 hReader, AC_APDU *Apdu,
                               BYTE slot);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
HReader	A valid reader handle returned by AC_Open()
Apdu.CLA	Instruction Class
Apdu.INS	Instruction Code (Please see ACI command format for detailed description)
Apdu.P1	Parameter 1
Apdu.P2	Parameter 2
Apdu.DataIn	Command data buffer
Apdu.Lc	Length of command data (DataIn) (Lc must be set to –1 if no DataIn)
Apdu.Le	Length of response data (DataOut) (Le must be set to –1 if no DataOut)
slot	The selected card slot

***Notes:**

1) **Lc must be set to –1 if there is no DataIn.**
2) **Le must be set to –1 if there is no DataOut.**

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
Apdu.DataOut	Response data buffer
Apdu.Le	Number of bytes received in Apdu.DataOut
Apdu.Status	Status bytes SW1, SW2 returned by the card

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.4.4 AC_Read_Magnetic_Card

This function is reads data stored on the tracks (1-3) of a magnetic card and stores them in the AC_MAGNETIC_CARD data-structure (see also chapter 3.2.7).

Format:

```
INT16 AC_DECL AC_Read_Magnetic_Card(INT16 hReader, INT16 track,
                                    INT16 timeout,
                                    AC_MAGNETIC_CARD *magCard);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
track	1-7: 3 bits are used to read the tracks, bit 4 to 7 are not used. 001 (01_H)= read track 1 010 (02_H)= read track 2 100 (03_H) = read track 3 … 111 (07_H) = read all tracks
timeout	Number of minutes for the operation to (since the application should wait for the user to slide the magnetic card).
MagCard	AC_MAGNETIC_CARD structure containing the values read from the magnetic card.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5 Encryption and Decryption Functions

The ACR80 is capable of performing DES/TRIPLE DES encryption/decryption using four types of encryption/decryption modes. These are ECB (Electronic Code Book), CBC (Cipher Block Chaining), CFB (8 bit Cipher FeedBack) and OFB (8bit Output Feedback) encryption /decryption techniques. For a detailed description of these modes and DES see also Appendix D. The keys and vectors used for the encryption/decryption are stored in the internal EEPROM of the reader. In order to do this, the functions as described below are available.

3.5.1 AC_Inject_IV

This command injects an Initial Vector for the later calculation in CBC/CFB/OFB-en(de)cryption mode. 16 different 8 byte initial vectors can be stored.

Format:

```
INT16 AC_DECL AC_Inject_IV(INT16 hReader, BYTE vectorNumber,
                           INT16 vectorLen, BYTE* vector);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
vectorNumber	The number under which the vector is stored and can be recalled. From 0-15.
vectorLen	8 (vectors are 8 bytes long)
vector	The 8 bytes vector data

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.2 AC_Inject_Key

This command injects a Key for the later calculation in DES/3DES/ECB/CBC/CFB/OFB-en(de)cryption mode. 16 different keys can be stored.

Format:

```
INT16 AC_DECL AC_Inject_Key(INT16 hReader, BYTE keyNumber,
                            INT16 keyLen, BYTE* key);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
keyNumber	The number under which the key is stored and can be recalled. From 0-15.
keyLen	The key length is either 8 or 16 or 24, depending on the encryption method used.
key	The key data (length equal to keyLen)

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.3 AC_Generate_Random_IV

This command generates and stores an Initial Vector for the later calculation in CBC/CFB/OFB-en(de)cryption mode. 16 different initial vectors can be stored.

Format:

```
INT16 AC_DECL AC_Generate_Random_IV(INT16 hReader, BYTE vectorNumber,
                                    BYTE &vectorLen, BYTE *vector);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
vectorNumber	The number under which the vector is stored and can be recalled. From 0-15.
vectorLen	8 (vectors are 8 bytes long)
vector	A buffer for the returned vector

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
vectorLen	Always 8, since the terminal only supports 8 bytes vectors
vector	The 8 bytes vector data

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.4 AC_Set_Encryption_Type

This command sets the type of en(de)cryption mode for the en(de)cryptions that follow. This settings remains in the reader until a next AC_Set_Encryption_Type is called on a power on occured.

Format:

```
INT16 AC_DECL AC_Set_Encryption_Type(INT16 hReader,
                                     BYTE vectorNumber,
                                     BYTE keyNumber,
                                     INT16 DESType,
                                     INT16 DESKeyLen,
                                     INT16 DESBlock);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
vectorNumber	The number under which the vector is recalled. From 0-15.
keyNumber	The number under which the key is recalled. From 0 - 15.
DESType	The encryption type used. DES for single DES encryption TRIPLE_DES for triple DES encryption.
DESKeyLen	The key length used DOUBLELEN for 16 bytes key TRIPLELEN for 24 bytes key
DESBlock	The cipher block chaining mechanism used. ECB for Electronic Code Book CBC for Cipher Block Chaining CFB for 8 Bit Cipher Feedback OFB for 8 Bit Output Feedback

Remarks:

1. Key lengths larger than 8 bytes will only be used in Triple DES. For single DES, always a key length = 8 bytes is used. Therefore, just input a 8 into the parameter DESKeyLen if the parameter DESType is DES.
2. For ECB, the initial vector is ignored, since ECB doesn't use vectors.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.5 AC_Encryption_Decryption

This command performs an encryption or a decryption using the settings set by AC_Set_Encryption_Type.

Format:

```
INT16 AC_DECL AC_Encryption_Decryption(INT16 hReader,
                                       BYTE encryption,
                                       INT16 &plainLen,
                                       BYTE *plaintext,
                                       INT16 &cipherLen,
                                       BYTE *ciphertext);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
encryption	DO_ENCRYPT (0): perform encryption DO_DECRYPT (1): perform decryption
plainLen	The length of the input plaintext in case of encryption or the length of the returned plaintext in case of decryption. (1-256)
plaintext	The data buffer containing the input plaintext in case of encryption or the data buffer for the returned ciphertext in case of decryption.
cipherLen	The length of the input ciphertext in case of encryption or the return data length of ciphertext in case of decryption. (1-256)
ciphertext	The data buffer containing the input ciphertext for encryption or the data buffer for the returned ciphertext in case of decryption.

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
plainLen	The length of the returned plaintext in case of decryption.
plaintext	The data buffer containing the returned ciphertext in case of decryption.
cipherLen	The return data length of ciphertext in case of decryption.
ciphertext	The data buffer containing the returned ciphertext in case of decryption.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.6 AC_Calculate_PIN_FTK

This command performs an FTK-encryption according to the previously defined type of en(de)cryption with the plaintext and the entered PIN. The plaintext and the PIN are calculated as in the DUKTP-function. The result is then ciphered according to the previously defined type of encryption.

Format:

```
INT16 AC_DECL AC_Calculate_PIN_FTK(INT16 hReader, INT16 plainLen,
                                   BYTE *plaintext, INT16 &cipherLen,
                                   BYTE *ciphertext, BYTE timeout);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
plainLen	The length of input plaintext
plaintext	The data buffer containing the input plaintext
timeout	Hexadecimal: 01_H~$3F_H$ (1~63 minutes). This value is used to terminal if no one presses a key in the terminal for the defined time. If the timeout is reached, an error-code is sent back.

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
cipherLen	The length of return ciphertext. This value is always 8 if no error occurred.
ciphertext	Data buffer containing the 8 bytes ciphertext

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.7 AC_Inject_DUKPT

This command injects the key serial number and the DUKPT key.

For details, please refer to the ANSI X9.24 standard.

Format:

```
INT16 AC_DECL AC_Inject_DUKPT(INT16 hReader, INT16 keyLen,
                              BYTE *DUKPTKey, INT16 serialNumberLen,
                              BYTE* serialNumber);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
keyLen	The length of input plaintext
DUKPTKey	The data buffer containing the input plaintext
serialNumberLen	The serial number length. Currently, this value must be set to 10.
serialNumber	The 10 bytes serial Number.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.8 AC_Request_PIN_Entry

This command performs a DUKPT-encryption with the plaintext and the entered PIN.
For details, please refer to the ANSI X9.24 standard.

Format:

```
INT16 AC_DECL AC_Request_PIN_Entry(INT16 hReader, INT16 plainLen,
                                   BYTE *plaintext,
                                   INT16 &serialNumberLen,
                                   BYTE* serialNumber,
                                   INT16 &PINBlockLen,
                                   BYTE* PINBlock,
                                   BYTE timeoutInMin);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
plainLen	The length of the input plaintext
plaintext	The data buffer containing the input plaintext
timeoutInMin	Hexadecimal: 01_H~$3F_H$ (1~63 minutes). This value is used to terminal if no one presses a key in the terminal for the defined time. If the timeout is reached, an error-code is sent back.

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
serialNumberLen	The serial number length. Currently, this value is always 10.
serialNumber	The 10 bytes serial Number
PINBlockLen	The length of the encrypted PIN blocks. This value is always 8 if no error occured.
PINBlock	Data buffer containing The 8 bytes encrypted PIN blocks.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.9 AC_Calculate_PIN_FTK

This command returns the PIN previously entered in the keypad during AC_Calculate_PIN_FTK.

Format:

```
INT16 AC_DECL AC_Get_FTK_PIN(INT16 hReader, INT16 plainLen,
                             BYTE *plaintext, INT16 cipherLen,
                             BYTE* ciphertext, INT16 &PINLen,
                             BYTE* PIN);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
plainLen	The length of the input plaintext
plaintext	The data buffer containing the input plaintext entered previously in AC_Calculate_PIN_FTK
cipherLen	The length of ciphertext. This value is always 8 if no error occurred.
ciphertext	The data buffer containing the input plaintext returned previously in AC_Calculate_PIN_FTK

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
PINLen	The length of the entered PIN. (0-14)
PIN	Data buffer containing the PIN code entered on the keypad previously in AC_Calculate_PIN_FTK.

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

3.5.10 AC_Exchange_PIN_APDU

This command prompts the user to key in the PIN to the terminal. The terminal will then send the combined data (the PIN and the input data if it has) to the card.

Format:

```
INT16 AC_DECL AC_Exchange_PIN_APDU(INT16 hReader,
                                   AC_PIN_APDU *PINApdu,
                                   BYTE slot);
```

Input Parameters:

The table below lists the parameters for this function

Parameters	Definition / Values
hReader	Handle returned by AC_Open()
PINApdu.CLA	Instruction Class
PINApdu.INS	Instruction Code (Please see ACI command format for detailed description)
PINApdu.P1	Parameter 1
PINApdu.P2	Parameter 2
PINApdu.DataIn	Command data buffer
PINApdu.Lc	Length of command data (DataIn) (Lc must be > 0) Remark: Lc includes the data into the card plus the pin.
PINApdu.Le	Length of response data (DataOut) (Le must be set to –1 if no DataOut)
PINApdu.PIN_Length	Length of the PIN
PINApdu.PIN_Position	The position in the databytes, counted from Byte=1
PINApdu.Coded_Scheme	0 : PIN is coded as ASCII (e.g. PIN entered in terminal = “1234” will become “31 32 33 34”) 1 : PIN is coded as binary (e.g. PIN entered in terminal = “1234” will become “01 02 03 04”)
PINApdu.PIN_SD_Position	0 : MSD (most significant digit) to the left side in the PIN position (e.g. PIN length = 4 and Lc = 6, DataIn = 01 02 03 04 05 06, and PIN position = 1, the PIN will be 01 02 03 04) 1 : LSD (least significant digit) to the right side in the PIN position (e.g. PIN length = 4 and Lc = 6, DataIn = 01 02 03 04 05 06, the PIN will be 03 04 05 06) Remark: if PIN Length = Lc, there will not have any difference.
PINApdu.PIN_Length_Counter	0 : without length counter 1 : with length counter
PINApdu.Length_Counter_Position	0 : the length counter is placed as the most left byte in the PIN position 1 : the length counter is placed as next following byte left from the PIN
PINApdu.Timeout	1 – 63 minutes (in HEX 00 – 3F). If 0 is chosen, the default timeout (1minute) is used.
slot	The selected card slot

Output Parameters:

The table below lists the parameters returned by this function

Parameters	Definition / Values
PINApdu.DataOut	Response data buffer
PINApdu.Le	Number of bytes received in PINApdu.DataOut
PINApdu.Status	Status bytes SW1, SW2 returned by the card

Returns:

The return value is zero if the function is successful. Otherwise, it returns a negative value containing the error code. For the detailed meaning of the error code, please refer to appendix A.

Appendix A: Table of Error Codes

Code	Meaning
Reader Errors	
-1001	Invalid reader handle
-1002	Port has already been occupied
-1003	The session has not been started
-1004	Reader does not respond
-1005	Unknown status from reader
-1006	Error reader response length
-1007	Reader unknown error
-1008	Reader Buffer Overflow
Card Communication Errors	
-1021	Card is not inserted
-1022	Card is not powered up
-1023	Selected protocol is not supported by the card
-1024	Parity error when communicating with card
-1025	Checksum error when communicating with card
-1026	Card not supported
-1027	Timeout when communicating with card
-1028	Fail to send data to card
-1029	Fail to receive data from card
-1030	Card does not respond
PPS Exchange Specific Errors	
-1041	Wrong initial byte received during PPS exchange
-1042	PPS exchange failed (card does not response to PPS request)
-1043	Unexpected PPS response
Reader Command Specific Errors	
-1051	Incorrect command parameter
-1052	Invalid instruction
-1053	Invalid command length
T0 Protocol Errors	
-1061	Wrong APDU command
-1062	Wrong response length
-1063	Wrong SW1 received from card
-1064	Wrong ACK received from card
T1 Protocol Errors	
-1091	T1 module reached the maximum number of resynchronizations
-1092	Incorrect EDC in T1 block detected
-1093	Invalid length of INF field of T1 frame received from card using T=1 protocol
-1094	User provided buffer is too small to receive all the T1 data
-1095	The length of a returned T1 block exceeds the Max IFS of reader
-1096	User provided buffer is not valid or NULL
-1097	There are 3 failure transmission attempts at the beginning of T1 protocol
-1098	Internal to driver meaning more data will be available for a transaction using T1
-1099	Internal to driver meaning a T1 abort chain request is received to abort chaining
-1100	A T1 abort chain request is received to abort chaining

-1101	The card requested a too long wait time extension (WTX) which the reader does not support
-1102	NAK received from card
-1103	ABORT request received from card
-1104	Sequence number mismatch
-1105	Data size from card exceeds the buffer limit of the reader
-1106	Invalid APDU input
DLL Internal Specific Errors	
-1111	Internal buffer has not been allocated
LCD Specific Errors	
-1121	LCD Draw function error
Keypad Specific Errors	
-1131	Invalid key is pressed
-1132	Keypad Timeout
EMV Specific Errors	
-1151	Le Overflow in chaining
Reader Clock Specific Errors	
-1161	Clock response data length invalid
-1162	Real time clock error
Magnetic Card Errors	
-1171	Magnetic card data invalid
-1172	Magnetic card data length invalid
Other errors	
-1181	Input parameter to the API function is invalid
-1182	Card session has been disconnected
-1183	The reader housing has been opened unconditionally
-1184	Internal system error
Memory Card Error	
-1191	The card is blocked
-1192	Wrong PRESENT CODE in Present Security Code-command
EEPROM errors	
-1201	Internal EEPROM is defected
-1202	EEPROM access denied
DUKPT errors	
-1211	DUKPT Counter overflow
-1212	PIN entry timeout occurred.
Secure PIN errors	
-1221	No PIN is entered to the terminal
-1222	The PIN is longer than the preferred length
I2C errors	
-1231	Invalid number of output data from read command

Appendix B: Memory Card Commands

SLE4428 Commands

Read Main Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	READ_MAIN_MEMORY_4428
P1	High byte of the start address	0x00
P2	Low byte of the start address	0x00
Lc	Number of bytes of the input data	-1 (no input)
Le	Number of bytes of the returned data	10
DataIn	Input data buffer	NULL
DataOut	Output data buffer	10 bytes returned
Status	The operation status	0x9000 (successful)

Write Main Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	WRITE_MAIN_MEMORY_4428
P1	High byte of the start address	0x00
P2	Low byte of the start address	0x00
Lc	Number of bytes of the input data	10
Le	Number of bytes of the returned data	-1 (no output)
DataIn	Input data buffer	10 bytes input data
DataOut	Output data buffer	NULL
Status	The operation status	0x9000 (successful)

Read Protected Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	READ_PROTECTION_MEMORY_4428
P1	High byte of the start address	0x00
P2	Low byte of the start address	0x00
Lc	Number of bytes of the input data	-1 (no input)
Le	Number of bytes of the returned data	10
DataIn	Input data buffer	NULL
DataOut	Output data buffer	10 bytes returned
Status	The operation status	0x9000 (successful)

Write Protected Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	WRITE_PROTECTION_MEMORY_4428
P1	High byte of the start address	0x00
P2	Low byte of the start address	0x00
Lc	Number of bytes of the input data	10
Le	Number of bytes of the returned data	-1 (no output)
DataIn	Input data buffer	10 bytes input data
DataOut	Output data buffer	NULL
Status	The operation status	0x9000 (successful)

Present Security Code

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	PRESENT_CODE_4428
P1	No Use	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	2
Le	Number of bytes of the returned data	1
DataIn	Input data buffer	2 bytes security code
DataOut	Output data buffer	1 byte error counter
Status	The operation status	0x9000 (successful)

Read Error Counter

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	READ_ERR_COUNTER_4428
P1	No Use	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	-1 (no input)
Le	Number of bytes of the returned data	1
DataIn	Input data buffer	NULL
DataOut	Output data buffer	1 bytes error counter
Status	The operation status	0x9000 (successful)

SLE4442 Commands

Read Main Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	READ_MAIN_MEMORY_4442
P1	Start address	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	-1 (no input)
Le	Number of bytes of the returned data	10
DataIn	Input data buffer	NULL
DataOut	Output data buffer	10 bytes output data
Status	The operation status	0x9000 (successful)

Read Protected Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	READ_PROTECTION_MEMORY_4442
P1	Start address	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	-1 (no input)
Le	Number of bytes of the returned data	10
DataIn	Input data buffer	NULL
DataOut	Output data buffer	10 bytes output data
Status	The operation status	0x9000 (successful)

Write Main Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	WRITE_MAIN_MEMORY_4442
P1	Start address	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	10
Le	Number of bytes of the returned data	-1 (no output)
DataIn	Input data buffer	10 bytes input data
DataOut	Output data buffer	NULL
Status	The operation status	0x9000 (successful)

Write Protected Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	WRITE_PROTECTION_MEMORY_4442
P1	Start address	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	10
Le	Number of bytes of the returned data	-1 (no output)
DataIn	Input data buffer	10 bytes input data
DataOut	Output data buffer	NULL
Status	The operation status	0x9000 (successful)

Present Security Code

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	PRESENT_CODE_4442
P1	No Use	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	3
Le	Number of bytes of the returned data	4
DataIn	Input data buffer	3 bytes security code
DataOut	Output data buffer	1 byte error counter + 3 byte security codes
Status	The operation status	0x9000 (successful)

Read Security Memory

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	READ_SECURITY_MEMORY_4442
P1	No Use	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	-1 (no input)
Le	Number of bytes of the returned data	4
DataIn	Input data buffer	4 bytes security memory data
DataOut	Output data buffer	NULL
Status	The operation status	0x9000 (successful)

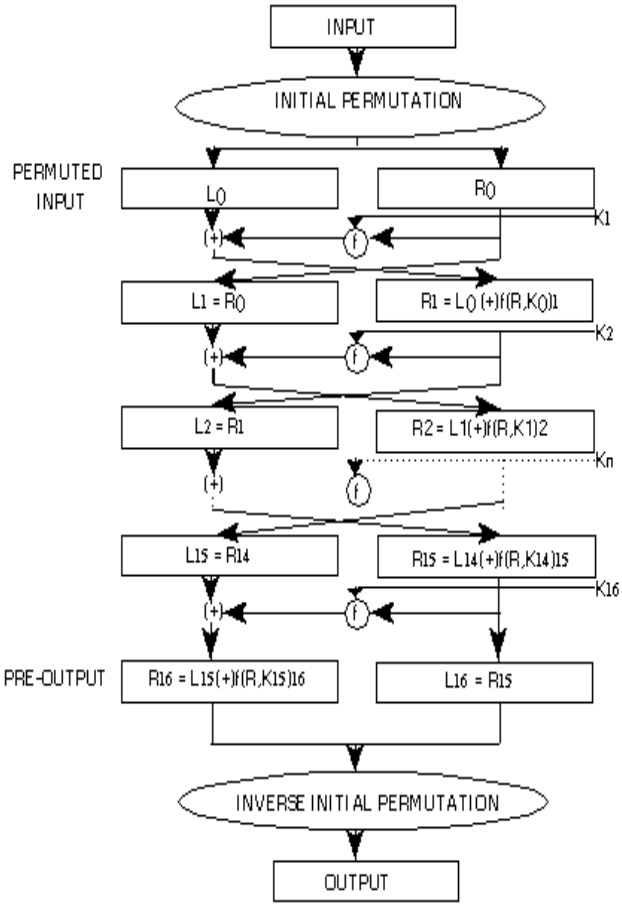
Change Security Code

APDU	Description	Example
CLA	No Use	0x00
INS	The instruction code of the command	CHANGE_CODE_4442
P1	No Use	0x00
P2	No Use	0x00
Lc	Number of bytes of the input data	3
Le	Number of bytes of the returned data	-1 (no output)
DataIn	Input data buffer	3 bytes security code
DataOut	Output data buffer	NULL
Status	The operation status	0x9000 (successful)

Appendix C: Encryptions, Decryptions

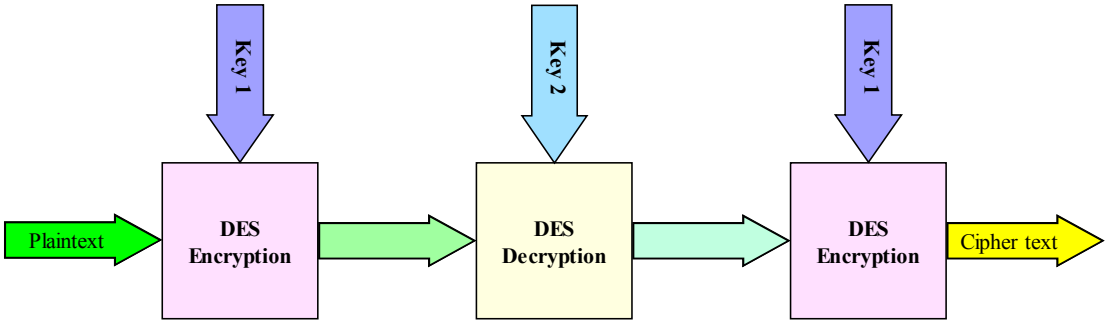
The DES algorithm

Following picture shows the DES encryption algorithm. The decryption algorithm follows the same structure. Both, input and key have a length of 8 byte.

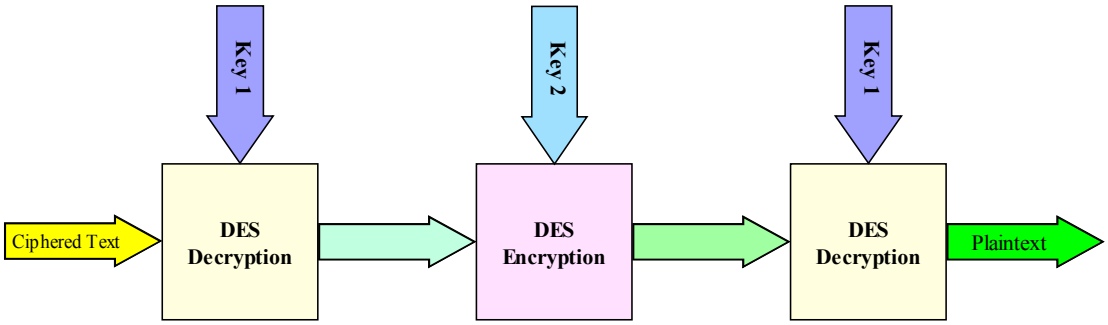


The DES decryption follows the same structure. Only the INITIAL PERMUTATION and the INVERSE INITIAL PERMUTATION has a different matrix from the encryption.

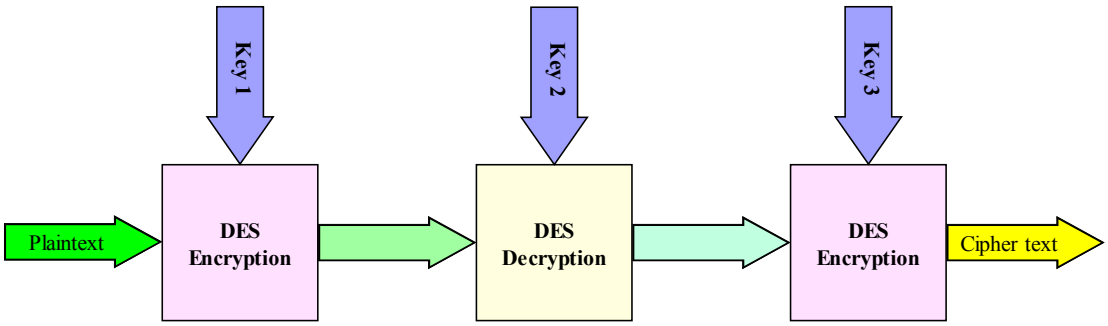
Triple DES encryption with double length key



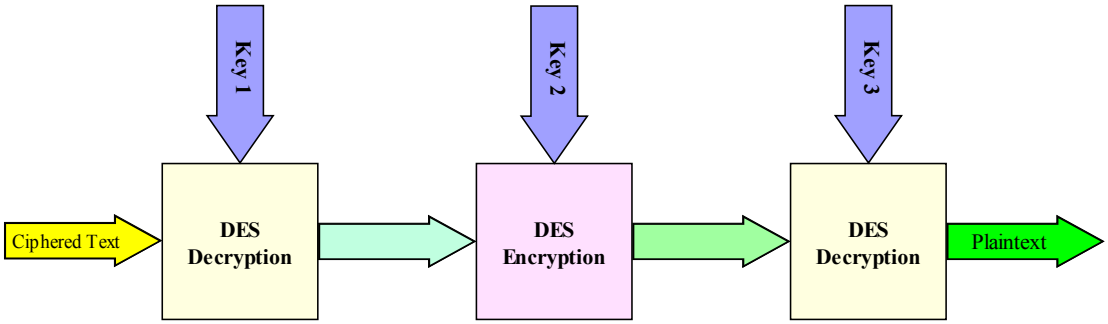
Triple DES encryption with double length key



Triple DES encryption with triple length key



Triple DES decryption with triple length key

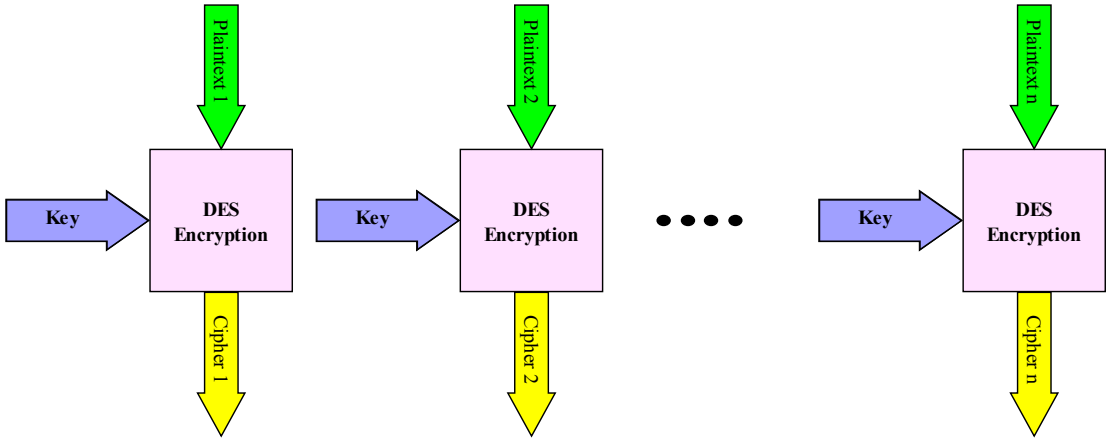


En(De)cryption Modes

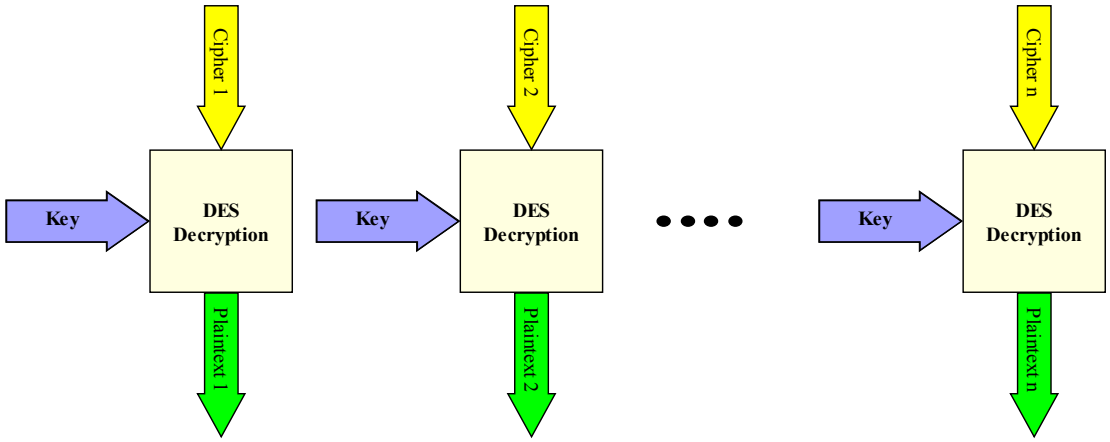
The Electronic Codebook Mode

In ECB mode, each plaintext block (8 bytes) is encrypted independently The result is for every plaintext block an 8 byte Ciphertext.

The Encryption



The Decryption

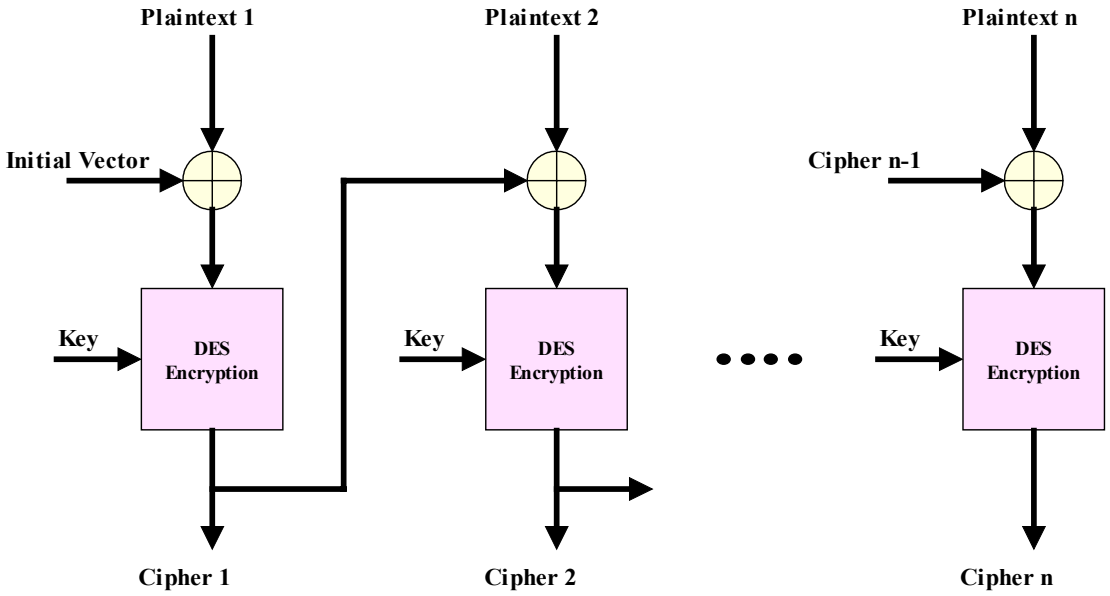


The Cipher Block Chaining Mode

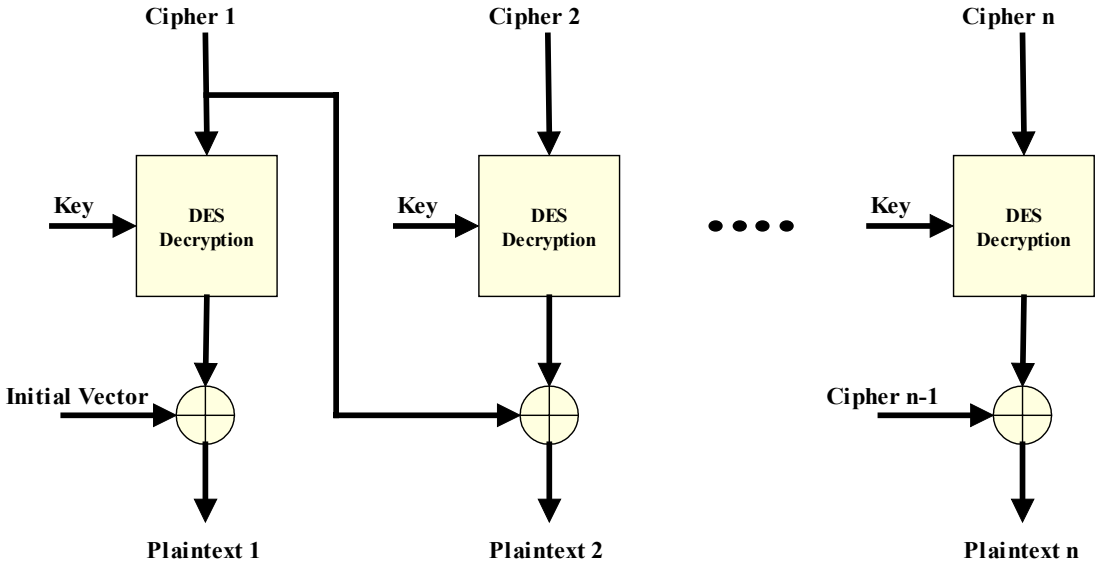
In CBC mode, each plaintext block is exclusive-ored with previous ciphertext block, then encrypted. An initialization vector or value c_0 is used as a "seed" for the process.

P = Plaintext, C = Ciphertext, K = Key and IV = Initial Vector.
P,C,IV are 8 bytes blocks. K is an 8, 16 or 24 bytes long key.

The Encryption



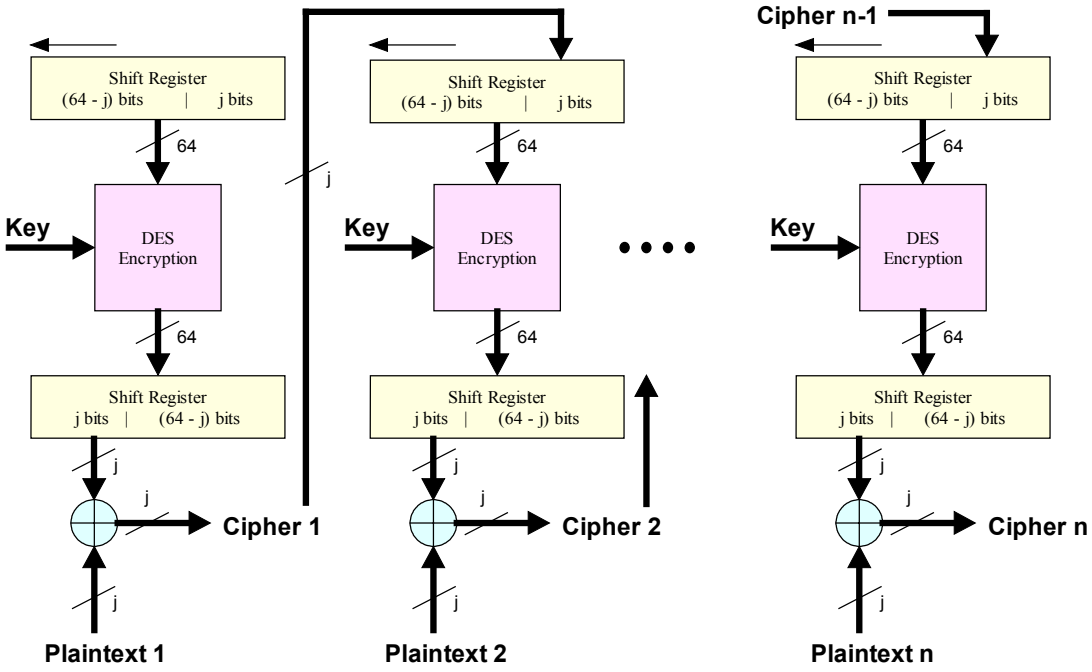
The Decryption



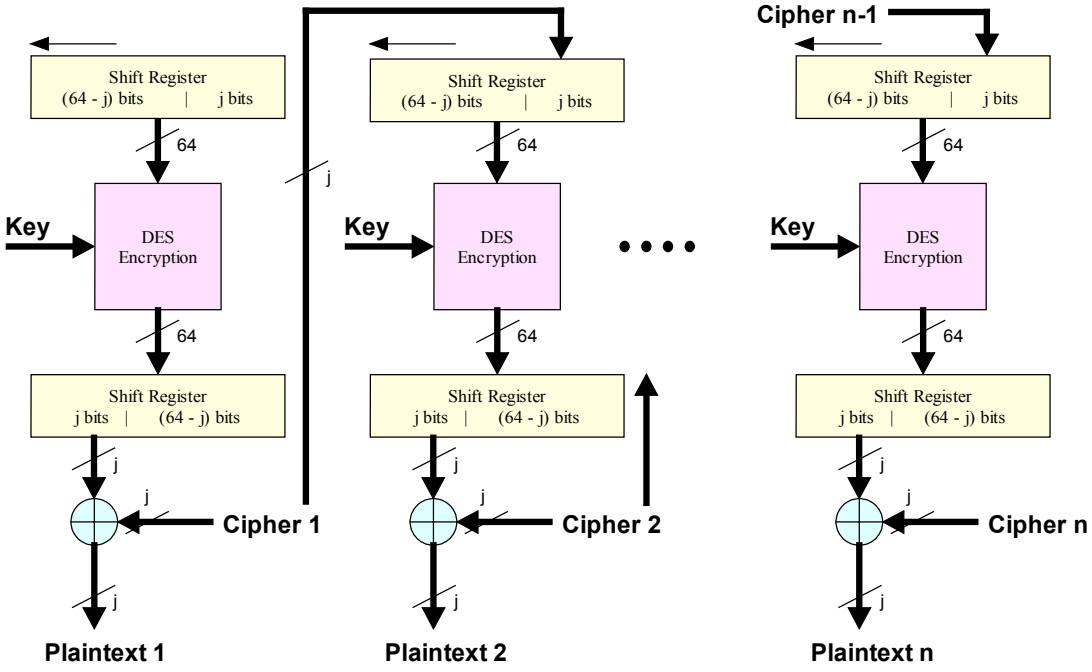
The Cipher Feedback Mode

P = Plaintext, C = Ciphertext, K = Key.
The shift register is initially loaded with an Initial Vector.
K is an 8, 16 or 24 bytes long key, j is selected to 8.

The Encryption



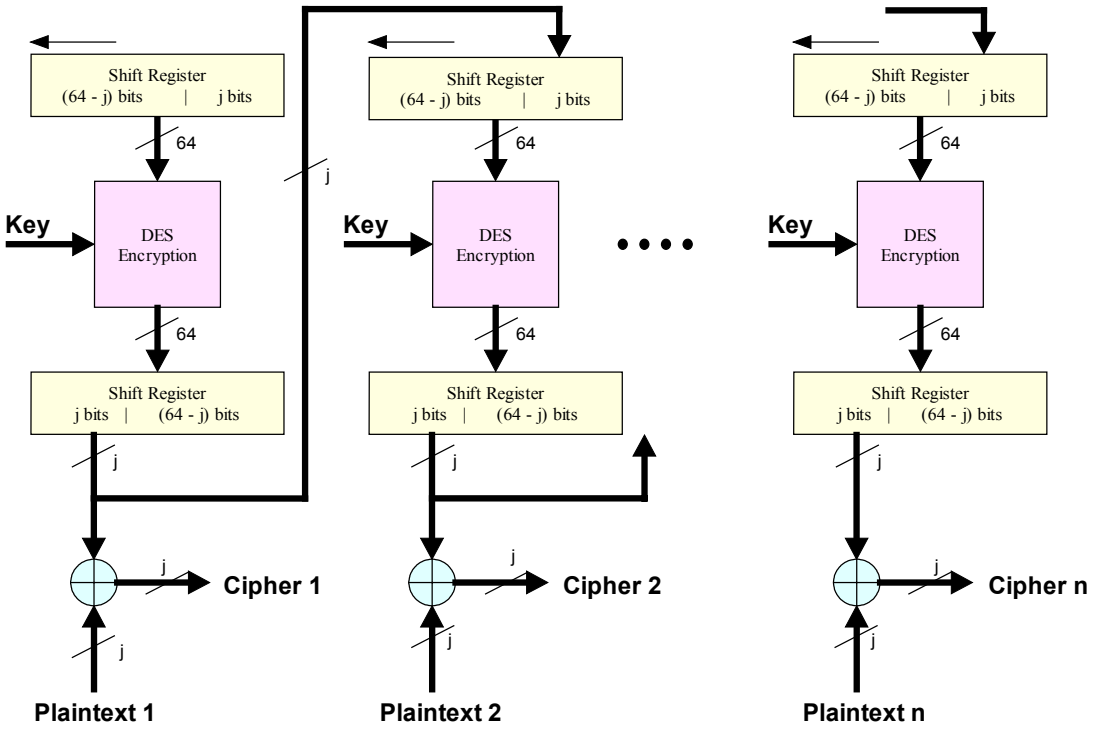
The Decryption



The Output Feedback Mode

P = Plaintext, C = Ciphertext, K = Key.
The shift register is initially loaded with an Initial Vector.
K is an 8, 16 or 24 bytes long key, j is selected to 8.

The Encryption



The Decryption