Classification:

Confidential

Scope of Distribution:

Beijing Novel-Super Digital TV Technology Co., Ltd. and SMS Manufacturers

Method of Distribution:

PDF with watermark

CDCAS3.0_SMS Interface Specification

<V1.1>

Beijing Novel-Super Digital TV Technology Co., Ltd.

Statement

This document describes in detail the interfaces of CDCAS3.0 with SMS, and aims to provide both integration sides with guidance and basis for successful integration.

This document will be only disclosed in Beijing Novel-Super Digital TV Technology Co., Ltd. as well as authorized SMS manufacturers. No person will be allowed to disclose it to others without prior permission.

Either person who gets this document via non-legal way shall use it or disclose it to any third party, otherwise may bear legal responsibilities.

This document shall be subject to the interpretation by Beijing Novel-Super Digital TV Technology Co., Ltd. (hereinafter referred to as NSTV).

Table of Contents

1 Overview

1.1 Introduction to CDCAS3.0

Digital TV has led a new revolution in the information industry in the world. NSTV, as a leading team in China, is dedicated to technical research of the digital TV industry, and has invested lots of human resources and materials in recent years. CDCAS3.0 is the main product of NSTV, which has been successfully deployed by many operators in China and has won praise and recognition from the customers such as SARFT, CCTV, Beijing TV, etc. The product hs also been deployed in overseas markets, such as, Thailand, Malaysia, Indonesia, Venezuela, Myanmar, Mongolia, etc.

CDCAS3.0 is a conditional access system (CAS) provided by NSTV with independent intellectual property rights. The system offers a platform for digital TV operators to guarantee the security of their contents and distribute the entitlements to their authentic subscribers.

NSTV is ready to work together with SMS vendors of powerful technical strength. The CDCAS3.0_SMS interface would be easy for the SMS vendor to integrate the SMS system with CDCAS3.0.

1.2 Definition

CDCAS3.0: NSTV's CAS with independent intellectual property right;

Integration: CDCAS3.0 exchanges data with SMS via this interface to implement the management of the subscribers.

1.3 Abbreviations

CAS: Conditional Access System;
SMS: Subscriber Management System;
BOSS: Business Operation & Support System;
STB: Set-Top-Box;
OSD: On Screen Display
IPPV: Impulse Pay Per View
MAC: Message Authentication Code;
3DES: Triple Data Encryption Standard;
MD5: Message Digest5

2 Communication Protocol

2.1 Basic Workflow

CDCAS3.0_SMS interface adopts the C/S architecture, where CDCAS3.0 works as server and SMS works as client. In order to reduce the resource usage and improve the system security, CDCAS3.0_SMS interface will create a session on TCP connection, and exchange data on the session level.

Fig 2.1 illlustrates the basic workflow of CDCAS3.0_SMS interface.

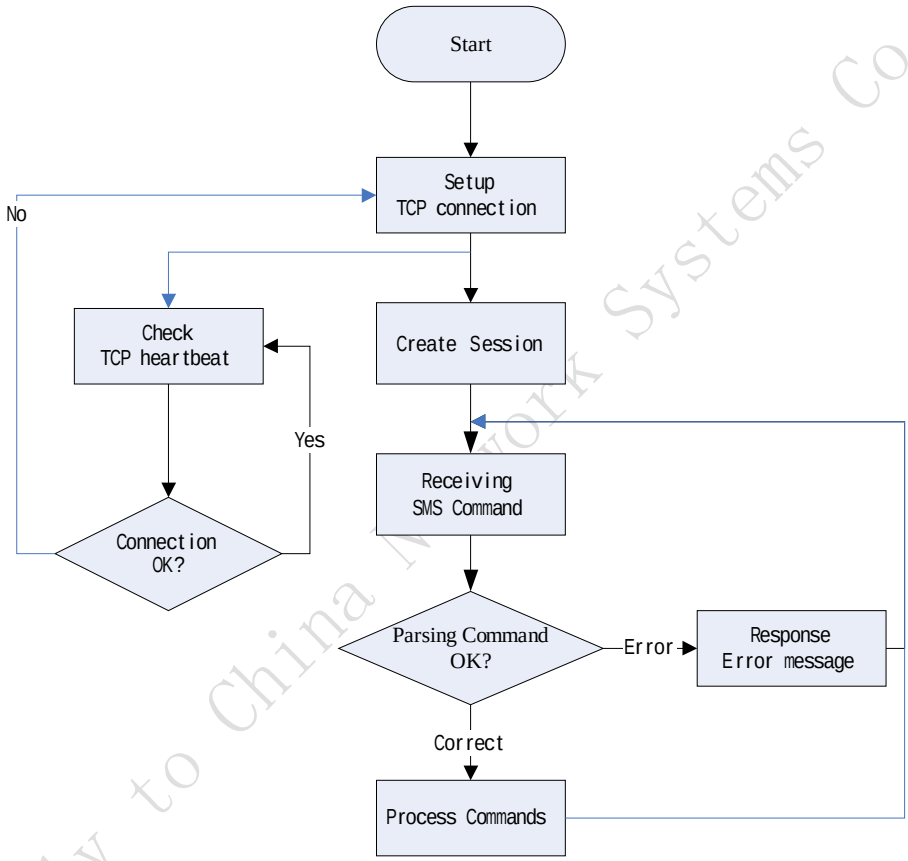


Figure 2.1 Basic Workflow

CDCAS3.0 supports multiple TCP connections from one or more SMS system. However, only one session can be created on one TCP connection simultaneously.

2.2 TCP Connection

Principles for TCP connection are as follow:

1. To setup the TCP connection, SMS should specify the IP address and the port number of the CDCAS3.0 server.
2. Heartbeat packets should be sent every 15 minutes to check if the TCP connection is still active. The data sent by SMS are {0x00, 0x04, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00}, while the response from CDCAS3.0 are {0x01, 0x06, 0xff, 0xff, 0xff, 0xff, 0x00, 0x00}.

   Note: The disconnection might be because of the bad network performance, or because that the firewall cuts off the connection due to the long-time silence on it.
3. While the TCP connection is broken (no heartbeat response received), SMS should re-setup the TCP connection, and re-setup the session then.
4. While exchanging data, SMS should check whether the TCP packet has been successfully received by CDCAS3.0 (TCP connection level). If not, SMS should re-send the data.

2.3 Create Session

Every time after setting up the TCP connection, SMS should firstly create a session with the CDCAS3.0. Then they could exchange the data. A main function of creating the session is to generate a session key, which will be used to protect the data for the consequent data exchange.

The steps to create a session are as follow:

1. SMS sends 'SMS _CA_CREATE_SESSION_REQUEST' to CDCAS3.0 in plaintext.
2. CDCAS3.0 replies 'CA_SMS _CREATE_SESSION_RESPONSE' to SMS, in which the data is encrypted by the SMS root key. The session key would be involved in the packet if the request from SMS is correct.
3. SMS will use the SMS root key to decrypt and check the data.
4. For the normal response packet, SMS will obtain the session key then, which is used for encrypting consequent data change for business operations (such as account opening, entitlement, etc.)

Note: It is a mistake to re-create a new session after one time data exchange. The session link should be kept until SMS complete all the requests and will not connect CDCAS3.0 for a long time.

2.4 Data Exchange

After creating the session, SMS could send the request packet (command) to CDCAS3.0 via this session link, while CDCAS3.0 will handle the received command and reply the response packet to SMS with the execution result of the command. This process is called 'Data Exchange', and both the request command and the result response are called 'Message'.

For different scenarios, every time SMS could send one request command or a batch of commands to CDCAS3.0 via the session link. CDCAS3.0 will process all the commands and then reply the results to SMS.

As CDCAS3.0 supports multiple connections from SMS, it has the mechanism to concurrently handle messages from multiple SMS.

The data structure of the 'Message' packet is shown as Table 2.1. Please be noted that:

1. The maximum length of the packet is **4096** bytes.
2. **Big-endian** is applied for both byte-order and bit-order.

Table 2.1 CDCAS3.0_SMS Message Packet

SYNTAX	bits	Notes
Data_Section(){		
Proto_Ver	8	Protocol version number
Crypt_Ver	6	Encryption version number
Key_Type	2	Key Type
OPE_ID	16	Operator ID
SMS _ID	16	SMS ID
DB_Len	16	Length of Data_Body
Data_Body()		Data
}		

Proto_Ver: Protocol version number for 'Data Exchange', which is defined by NSTV. Currently the version number should be '1'.

Crypt_Ver: The version number for the encryption scheme, which is defined by NSTV. Currently the version number should be '1'.

Key_Type: The type of the key applied for encrypting the message, which is defined as follow (refer to 'Encryption Scheme' section for details):

"00": root key

"01": current session key

"10": no key (plaintext for this message)

"11": reserved

OPE_ID: Operator ID in Hex, which is assigned by NSTV for different network operators.

SMS _ID: SMS ID is used to specify different SMS systems connecting to CDCAS3.0, which is assigned by CDCAS3.0. Each SMS system would have its own root key.

DB_Len: The length of the 'Data_Body' in bytes.

Data_Body(): The detail of the 'Message' with the data structure as Table 2.2.

Table 2.2 CDCAS3.0_SMS Data_Body

SYNTAX	bits	Notes
Data_Body(){		
DB_ID	16	Data_Body ID
Msg_ID	16	Message ID
Data_Len	16	Data length
Data_Cont()		Data
For(i=0; i<N; i++){		
Padding_Byte	8	Padding bytes
}		
MAC	128	Checksum
}		

DB_ID: Data_Body ID is generated by SMS, which is a unique number in one session to identity the data exchange on current session.

Msg_ID: Message ID indicates the type (function) of this message. The values of the message types are defined in Appendix 1 'Message ID'.

Data_Len: Refers to the length of Data_Cont() in bytes.

Data_Cont: The contents of the message, where the parameters are carried. The details are described in next section.

Padding_Byte: For encryption requirement, the Data_Body length should be multiples of 8 bytes. When the length does not match multiples of 8 bytes, the padding bytes should be applied to meet the requirement.

MAC(Message Authentication Code): The checksum against all data (excluding the MAC field) in Data_Body(). The algorithm for calculating the MAC is defined by NSTV. For this version, MD5 (Message Digest 5) is applied.

Notes:

1. An example of the MAC field is shown in 'Encryption Scheme' section.
2. For MD5 algorithm, the reference code in C++ is offered by NSTV in Appendix 5, where the following function could be called to generate the MAC.

   unsigned char TFCA_MD5
   *(unsigned char *pInPut, uint4 dwInput_Length, unsigned char *pOutPut)*

   pInPut: pointer addressing to the first byte of the data for calculating MAC.
   dwInput_Length: length of the data for calculating MAC, in bytes.
   pOutPut: pointer addressing to the first byte of the returned MAC.

There might be two types of transmission error happen on CDCAS3.0_SMS interface: unpacking failed (e.g, format un-match) and decryption failure. In such cases, CDCAS3.0 will return the packet with following format:

Table 2.3 Format of Response to Illegal Data Packet

SYNTAX	bits	Notes
Data_Section(){		

Proto_Ver	8	interface protocol version number
Crypt_Ver	6	encryption scheme version number
Key_Type	2	encryption key type
Reserved	16	fixed to be 0xFFFF
Reserved	16	fixed to be 0xFFFF
Reserved	16	fixed to be 0x0000

}

2.5 Encryption Scheme

To guarantee the security and the integrity of data transmission, the Data_Body() field of the Message (packet) should be encrypted and attached with digit-digest.

There are two types of keys to be used for this interface: **the root key** and **the session key**.

The root key is used to encrypt the session key while creating a session. Each SMS connected to CDCAS3.0 would be assigned a unique root key by NSTV. Generally, the root key would be configured in both CDCAS3.0 and SMS, and is relatively fixed.

The session key is generated by CDCAS3.0 while creating the session. It is used to encrypt Data_Body() of all Messages except *SMS _CA_CREATE_SESSION_REQUEST*.

In this version protocol, 3DES (Data Encryption Standard) algorithm is applied while using root key, and DES algorithm is applied while using session key. Refer to Appendix 6 for the algorithm implementation. The two functions below could be directly called for 3DES and DES encryption.

bool TFCA_3DES(bool bEnspot, unsigned char pbyKey,int nLength, unsigned char* pbySource, unsigned char* pbyTarget)*

bool TFCA_DES(bool bEnspot, unsigned char pbyKey,int nLength, unsigned char* pbySource, unsigned char* pbyTarget)*

Parameters:
bEnspot: true means encryption, false means decryption
pbyKey: point to encryption key
nLength: length of data for encryption (with unit of bytes) should be multiples of 8
pbySource: point to data first pointer for encryption
pbyTarget: point to first pointer of returned encrypted data

Here we take ‘create session’ as an example to illustrate the encryption scheme, including the MAC check.

1. Firstly, ‘Create Session’ request will be in plaintext as follow:

unsigned char session[32] =
{0x01,0x06,0x00,0x01,0x00,0x01,0x00,0x18,

0x00,0x01,0x00,0x01,0x00,0x00,0xf0,0xf0,0xe1,0xee,0x84,0x14,0x74,
0xbb,0x8e,0x30,0xad,0xd7,0xe8,0xb1,0xbd,0x52,0x0b,0xd7};

The details of the packet header and the Data_Body() are illustrated as Fig. 2.2 and Fig. 2.3.

Figure 2.2 'Create Session' Request Packet Header

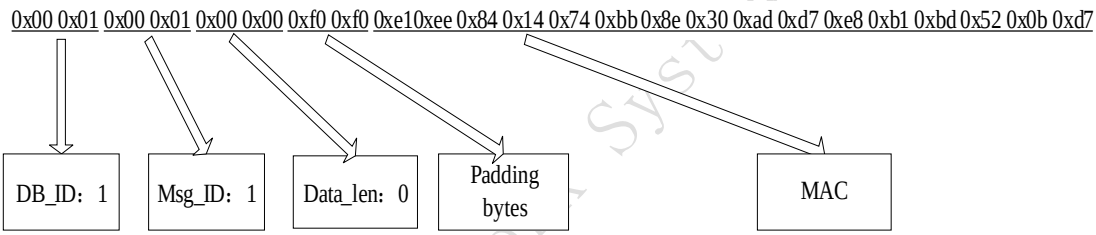


Figure 2.3 'Create Session' Request Data_Body()

2. The response to this request will be as follow. Here, supposing the root key is 'ABCDEFGHIJKLMNOP' and the returned session key is '12345678'. Then,

1) The packet header of the response will in plaintext as follow:

unsigned char head[8] =
{0x01,0x04,0x00,0x01,0x00,0x01,0x00,0x28}

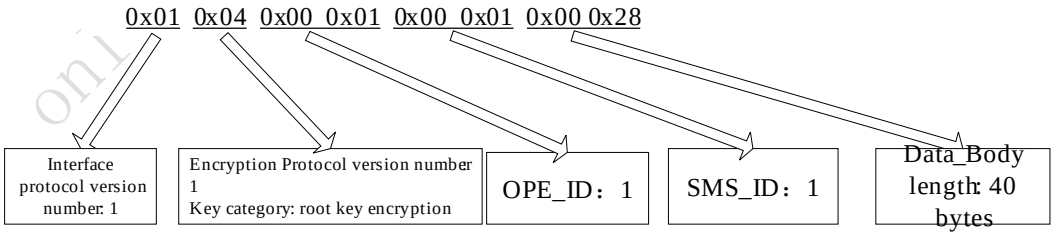


Figure 2.4 'Create Session' Response Packet Header

2) The Data_Body() of the response will be in cipher-text as follow:

Part of returned data packet Data_Body() is ciphertext with the following contents:

unsigned char enbody[40] =

{0x0c,0xb8,0x8c,0x3f,0x86,0xce,0x15,0x0d,0xb5,0x22,0x65,0x9a,0x56,0xee,0x40, 0xa5,0xef,0xab,0xec,0x4b,0x71,0xf0,0xa7,0xe5,0x7b,0x0a,0xd4,0xcd,0x65,0xc4, 0xd3,0xe1,0xa3,0x54,0x28,0x16,0x07,0xfa,0x13,0x02}

3) Decrypting it with the root key 'ABCDEFGHIJKLMNOP', we could got the plaintext Data_Body() as follow, including MAC field.

unsigned char plainbody[40] =
{0x00,0x01,0x80,0x01,0x00,0x0E,0x00,0x00,0x00,0x00,0x00,0x08,0x31,0x32, 0x33,0x34,0x35,0x36,0x37,0x38,0xF0,0xF0,0xF0,0xF0,0x2c,0x90,0x85,0x0d, 0xf3,0xfa,0x1a,0x64,0x93,0xcc,0xad,0xa2,0xb4,0x30,0x72,0xb2}

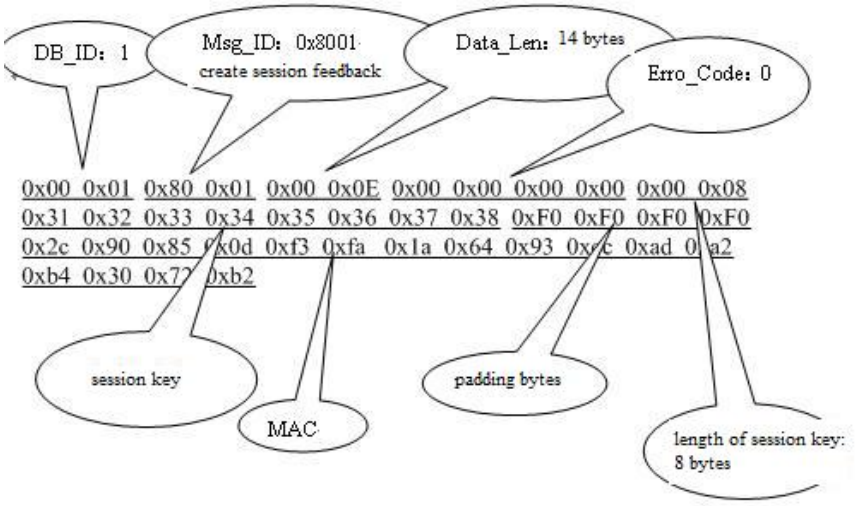


Figure 2.5 'Create Session' Response Data_Body() in plaintext

Note: For details of 'Create Session' request and response, please refer to Section 3.1.

2.6 Requirements for SMS Commands Processing

To ensure the SMS commands could be successfully processed, some requirements should be followed.

1. SMS should send 'Open Account' at first to activate a smart card. Before the card is activated, CDCAS3.0 will reject to conduct any other commands, such as entitlement, OSD, mail, etc., against this card.
2. SMS could use 'Close Account' to deactivate a smart card. In the case, CDCAS3.0 will clear all entitlement information for this card. SMS could use it until sending 'Open Account' to it again.

3. The DB_ID is used to identify the Data_Body() of the request and its response. SMS should ensure the uniqueness of DB_ID of the message pack within the same SMS connection (session). For different SMS connections, the DB_ID might be overlapped.

4. SMS will get the command processing result from the CDCAS3.0 response. If the response shows the command has been successfully executed, SMS can confirm that this command has become effective. If the response shows the command is not successful, SMS should have a mechanism to re-send this command manually or automatically.

3 Description of Messages

For 'Data Exchange', all messages will follow the syntax described in last section, but with the difference contents in Data_Cont() field. So, in this section, only the contents of Data_Cont() are described for each message.

The Message_ID is used to indicate messages for different functions, and the request and response messages will be described in pairs in this section.

3.1 Create Session

After setting up the TCP connection, SMS and CDCAS3.0 will use this pair of messages to create a session.

3.1.1 SMS _CA_CREATE_SESSION_REQUEST

For this message, the Data_Cont() is empty and the Data_Body() structure is shown in Table 3.1. The data in Data_Body() will not be encrypted, and will be sent in plaintext.

Table 3.1 Data Body Structure of SMS _CA_CREATE_SESSION_REQUEST

SYNTAX	bits	NOTES
Data_Body(){		
DB_ID	16	Data_Body ID
Msg_ID	16	*SMS _CA_CREATE_SESSION_REQUEST*
Data_Len	16	Data length is 0
For(i=0; i<N; i++){		
Padding_Byte	8	padding bytes
}		
MAC	128	checksum
}		

Msg_ID: *SMS _CA_CREATE_SESSION_REQUEST*, refers to Appendix 1.
Data_Len: length of Data_Cont(), '0' for this message.

3.1.2 CA_SMS _CREATE_SESSION_RESPONSE

Data_Cont() of this message is shown in Table 3.2. Data_Body() is encrypted with the SMS root key.

Table 3.2 Data_Cont() of CA_SMS _CREATE_SESSION_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	Error codes
Key_Len	16	Key length
For(i=0; i< Key_Len; i++){		

Key_Char	8	Session key
}		
}		

Key_Len: length of session key, in bytes.
Key_Char: session key.

3.2 Open Account

Open Account is to setup the relationship between the subscriber and the smart card. It is an initialization process should be conducted before the smart card could be used.

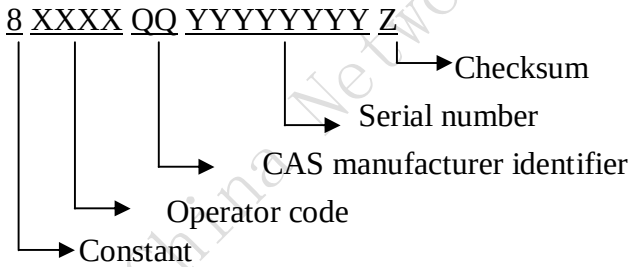
3.2.1 SMS _CA_OPEN_ACCOUNT_REQUEST

Data_Cont() of this message is shown in Table 3.3:

Table 3.3 Data_Cont() of SMS _CA_OPEN_ACCOUNT_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
}		

CARD_SN: The smart card serial number with following coding rule:



The CARD_SN is expressed in ASCII with hexadecimal system. For example, if the CARD_SN is 800……751, it would be expressed as {0x38 0x30 0x30 ……0x37 0x35 0x31}.

Note: The CARD_SN mentioned in following sections will use the same rules.

3.2.2 CA_SMS _OPEN_ACCOUNT_RESPONSE

Data_Cont() of this message is shown in Table 3.4:

Table 3.4 Data_Cont() of CA_SMS _OPEN_ACCOUNT_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.3 Close Account

Close Account is to stop the service against the subscriber, which is equal to revoking all entitlements of the subscriber.

3.3.1 SMS _CA_STOP_ACCOUNT_REQUEST

Data_Cont() of this message is shown in Table 3.5:

Table 3.5 Data_Cont() of SMS _CA_STOP_ACCOUNT_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
}		

3.3.2 CA_SMS _STOP_ACCOUNT_RESPONSE

Data_Cont() of this message is shown in Table 3.6:

Table 3.6 Data_Cont() of CA_SMS _ STOP_ACCOUNT _RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.4 Lock Card

Lock card is to disable the smart card and forbid it to decrypt the data until the card is unlocked. If the card is locked, the subscriber could not watch scrambled programs any more.

3.4.1 SMS _CA_SET_LOCK_REQUEST

Data_Cont() of this message is shown in Table 3.7:

Table 3.7 Data_Cont() of SMS _CA_SET_LOCK_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
}		

3.4.2 CA_ SMS _SET_LOCK_ RESPONSE

Data_Cont() of this message is shown in Table 3.8:

Table 3.8 Data_Cont() of CA_ SMS _SET_LOCK _RESPONSE

SYNTAX	bits	NOTES

Data_Cont(){		
Erro_Code	32	error codes
}		

3.5 Unlock Card

Unlock card is to resume the disabled smart card.

3.5.1 SMS _CA_SET_UNLOCK_REQUEST

Data_Cont() of this message is shown in Table 3.9:

Table 3.9 Data_Cont() of SMS _CA_SET_UNLOCK_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
}		

3.5.2 CA_SMS _SET_UNLOCK_ RESPONSE

Data_Cont() of this message is shown in Table 3.10:

Table 3.10 Data_Cont() of CA_ SMS _SET_UNLOCK _RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.6 Set Properties

There are 10 property items could be set for the smart card with different values. These properties could be used by the system to address the card.

3.6.1 SMS _CA_SET_CHARACTER_REQUEST

Data_Cont() of this message is shown in Table 3.11:

Table 3.11 Data_Cont() of SMS _CA_ SET_CHARACTER _REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
No	8	property index, from 0 to 9
Cha	32	Property value
}		

No: Property Index, scope from 0-9. Refer to Appendix 3 for details.

Cha: Value of the specified property item, which could be set via this message. Default

value is zero for each property.

3.6.2 CA_SMS _SET_CHARACTER_RESPONSE

Data_Cont() of this message is shown in Table 3.12:

Table 3.12 Data_Cont() of CA_SMS _ SET_CHARACTER_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.7 STB-Card Pairing

This message is to pair the specified STBs with the specified smart card, so that the card could only do decryption when it is used in the pairing STB.

3.7.1 SMS _CA_REPAIR_REQUEST

Data_Cont() of this message is shown in Table 3.13:

Table 3.13 Data_Cont() of SMS _CA_REPAIR_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
Card_SN	128	Smart card serial number
for (i=0; i<5; i++)		
{		
STBID	48	STB serial number
}		
}		

Note:

1. The Card_SN field is required to specify which smart card would be paired.
2. The STBID field can be 0 byte or 30 bytes.
3. If there is 0 byte, the smart card will be set to ‘un-paired with any STB’.
4. If there are 30 bytes, each 6 bytes refer to a STB. Maximum 5 STBs could be paired with the card. When the number of STBs is less than 5, the specified STBIDs should be listed in front, and the unused bytes should be set to 0.
5. While the smart card receives this message, the old pairing list will be replaced by the new received one totally.

3.7.2 CA_SMS _REPAIR_RESPONSE

Data_Cont() of this message is shown in Table 3.14:

Table 3.14 Data_Cont() of CA_SMS _REPAIR_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.8 Reset PIN

The smart card has PIN codes to protect some operations, which could be changed by the user. This message is to reset PIN codes to factory default value in case the user forgets the PIN.

3.8.1 SMS _CA_RESETCARDPIN_REQUEST

Data_Cont() of this message is shown in Table 3.15:

Table 3.15 Data_Cont() of SMS _CA_RESETCARDPIN_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
}		

3.8.2 CA_SMS _RESETCARDPIN_RESPONSE

Data_Cont() of this message is shown in Table 3.16:

Table 3.16 Data_Cont() of CA_SMS _RESETCARDPIN_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.9 Charge E-Slot

This message is to set the credit-limit value for the E-slot (electric wallet)of the card.

3.9.1 SMS _CA_SETSLOTMONEY_REQUEST

Data_Cont() of this message is shown in Table 3.17:

Table 3.17 Data_Cont() of SMS _CA_SETSLOTMONEY_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
Slot_ID	8	E-slot ID, range from 1 to 4
Cred_Lim	16	Value of credit limit
}		

Cred_Lim: The credit sum of the E-slot, which is the total points being recharged into this E-slot. The maximum value of credit-limit is 65535.

3.9.2 CA_SMS _SETSLOTMONEY_RESPONSE

Data_Cont() of this message is shown in Table 3.18:

Table 3.18 Data_Cont() of CA_SMS _SETSLOTMONEY_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.10Entitlement

This message is to send entitlements to the smart card.

3.10.1SMS_CA_ENTITLE_REQUEST

Data_Cont() of this message is shown in Table 3.19:

Table 3.19 Data_Cont() of SMS _CA_ENTITLE _REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
ProdCount	8	Number of products
for(I=0;I<ProdCout;I++) {		
Enti_Type	1	Entitlement type
TF_Reserved	5	NSTV reserved
Prod_ID	10	product ID
Tape_Ctrl	8	Video control
Start_Time	32	Start time
End_Time	32	End time
}		
}		

ProdCount: The number of products, the maximum is 190.

Enti_Type: Entitlement type, ‘0’ for detitle, ‘1’ for entitle, others are reserved. While it is ‘0’ (detitle), Tape_Ctrl, Start_Time and End_Time fields are invalid.

TF_Rerserved: NSTV reserved.

Prod_ID: ID of product for entitlement, with value scope of: 1-1023.

Tape_Ctrl: Video control flag, 0x00 for un-recordable, 0x01 for recordable, and others are reserved not to be used.

Start_Time/ End_Time: The start time and the end time of the entitlement in ‘time_t’ format. The scope is from 2000-Jan-1-0:0:0 to 2030-Dec-31-23:59:59.

The samples for ‘time_t’:

42 2D 4D D4 refers to 2005-03-08-15:01:40

72 BB 4A 00 refers to 2030-12-31-00:00:00

3.10.2 CA_SMS _ENTITLE_RESPONSE

Data_Cont() of this message is shown in Table 3.20:

Table 3.20 Data_Cont() of CA_SMS _ENTITLE _RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.11Extended Entitlement

Comparing with Entitlement message, this message extends the value scope of product ID from '1-1023' to '1-65535'.

3.11.1SMS _CA_ENTITLEEXT_REQUEST

Data_Cont() of this message is shown in Table 3.29:

Table 3.29 Data_Cont() of SMS _CA_ENTITLEEXT_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
ProdCount	8	number of product
for(I=0;I<ProdCout;I++) {		
Enti_Type	1	entitlement type
TF_Reserved	7	NSTV reserved
Prod_ID	16	product ID, range from 1-65536
Tape_Ctrl	8	video control
Start_Time	32	start time
End_Time	32	end time
}		
}		

3.11.2CA_SMS _ENTITLEEXT_RESPONSE

Data_Cont() of this message is shown in Table 3.30:

Table 3.30 Data_Cont() of CA_SMS _ENTITLEEXT_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.12Set Child Card

This message is to appoint a smart card to be the child card of another.

3.12.1 SMS _CA_SET_CHILD_REQUEST

Data_Cont() of this message is shown in Table 3.21:

Table 3.21 Data_Cont() of SMS _CA_SET_CHILD_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
Card_SN	128	Smart card serial number of the child
Parent_Card_SN	128	Smart card serial number of the parent
Feed_Interval_Hour	16	Feed interval (hours)
}		

Feed_Interval_Hour: Define the feed interval in hours. The child card is required to be fed with the data from the parent card periodically. If it has not been fed after the feed interval expires, the child card will be disabled.

3.12.2 CA_SMS _SET_CHILD_RESPONSE

Data_Cont() of this message is shown in Table 3.22:

Table 3.22 Data_Cont() of CA_SMS _SET_CHILD_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.13 Release Child Card

This message is to release a card not to be a child card any more.

3.13.1 SMS _CA_CANCEL_CHILD_REQUEST

Data_Cont() of this message is shown in Table 3.23:

Table 3.23 Data_Cont() of SMS _CA_CANCEL_CHILD_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number of the child card
}		

3.13.2 CA_SMS _CANCEL_CHILD_RESPONSE

Data_Cont() of this message is shown in Table 3.24:

Table 3.24 Data_Cont() of CA_SMS _CANCEL_CHILD_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.14Send Mail

Send mails to specified subscribers addressed by the addressing expression.

Note:

Addressing expression could be used for grouping subscribers in Messages, such as, 'Send Mail', 'Send OSD', 'Switch Channel', 'Release Lock'. Refer to Appendix 4 for the details of addressing expression.

3.14.1SMS _CA_SEND_EMAIL_REQUEST

Data_Cont() of this message is shown in Table 3.31:

Table 3.31 Data_Cont() of SMS _CA_SEND_EMAIL_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
Exp_Len	8	expression length
For(i=0; i< Exp_Len; i++){		
Exp_Char	8	expression
}		
Title_Len	8	Title length
For(i=0; i< Title_Len;i++){		
Title_Char	8	title
}		
Cont_Len	8	Text length
For(i=0; i< Cont_Len; i++){		
Cont_Char	8	text
}		
Importance	8	Importance
}		

Exp_Len: length of the addressing expression.
Exp_Char: addressing expression, maximum 40 char, should NOT be null.
Title_Len: length of the mail title.
Title_Char: mail title, maximum 30, should NOT be null.
Cont_Len: length of the mail text.
Cont_Char: mail text bytes, maximum 160, should NOT be null.
Importance: mail importance, '0' for ordinary, '1' for important, others reserved.

3.14.2CA_SMS _SEND_EMAIL_RESPONSE

Data_Cont() of this message is shown in Table 3.32:

Table 3.32 Data_Cont() of CA_SMS _SEND_EMAIL_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.15 Send OSD

Send OSD to specified subscribers addressed by the addressing expression.

3.15.1 SMS _CA_SEND_OSD_REQUEST

Data_Cont() of this message is shown in Table 3.33:

Table 3.33 Data_Cont() of SMS _CA_SEND_OSD_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
Exp_Len	8	expression length
For(i=0; i< Exp_Len; i++){		
Exp_Char	8	expression
}		
Cont_Len	8	text length
For(i=0; i< Cont_Len; i++){		
Cont_Char	8	text
}		
Style	8	Display method
Duration	32	duration
}		

Exp_Len: length of the addressing expression.

Exp_Char: addressing expression, maximum 40 char, should NOT be null.

Cont_Len: length of the OSD text.

Cont_Char: OSD text, maximum 180 char, should NOT be null.

Style: Define the display method,

'1' - display on top of screen and left to right scrolling;

'2' - display on botom of screen and left to right scrolling;

Others - reserved

Duration: Duration of the OSD display on the screen, in seconds (s), with value scope of 1~900.

3.15.2 CA_SMS _SEND_OSD_RESPONSE

Data_Cont() of this message is shown in Table 3.34:

Table 3.34 Data_Cont() of CA_SMS _SEND_OSD_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.16Switch Channel

Force the specified subscribers, addressed by the addressing expression, to switch to a specified channel. The parameters of the channel, such as the frequency, the symbol rate, the modulation mode, the component number, the component type (audio and video), the respectively PID and ECM PID, etc., should be defined in the message.

3.16.1SMS _CA_LOCK_SERVICE_REQUEST

Data_Cont() of this message is shown in Table 3.35:

Table 3.35 Data_Cont() of SMS _CA_ LOCK_SERVICE_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
Exp_Len	8	expression length
For(i=0; i< Exp_Len; i++){		
Exp_Char	8	expression
}		
LockFlag	8	0: means only switching but not locking. 1: means locking.
Reserved	16	reserved field
Frequency	32	frequency
Reserved	12	
fec_outer	4	Forward Error Correction outer codes
Modulation	8	modulation method
symbolrate	28	symbol rate
fec_inner	4	Forward Error Correction inner codes
PCR PID	16	clock synchronization PID code
ComponentNum	8	Number of programs components
For(int I=0;i< ComponentNum;i++){		
CompType	8	underlying stream type
CompPID	16	underlying stream PID
ECMPID	16	PID of ECM pack of descrambled underlying stream CW
}		
Reserved	8	fixed to be 0x00
Reserved	8	fixed to be 0x00
Reserved	8	fixed to be 0x00
Reserved	8	fixed to be 0x00
Reserved	8	fixed to be 0x00
Reserved	8	fixed to be 0x00
}		

Exp_Len: length of the addressing expression.

Exp_Char: addressing expression, maximum 40 char, should NOT be null.

LockFlag: Lock mode, '0' for just switching the channel but not locking the control panel, '1' means switching and locking the control panel.

Reserved: NSTV reserved field.

Frequency: Channel frequency in MHz, with value scope of 0-9999.9999. 4-bytes BCD code is used to indicate this value, the first 2 bytes are for integer part, and the last 2 bytes are for decimal part. If the valid bits are less than 4 bytes, use 0 to make up the relevant bits. For example:

For frequency 4358.59MHz, the code is 0x43585900;

For frequency 879.564MHz, the code is 0x08795640.

Reserved: NSTV reserved field.

fec_outer: Forward Error Correction Outer codes.

Modulation: modulation mode , with value scope of 0-255. For detailed definitions, see below:

0: Reserved;
1: QAM16;
2: QAM32;
3: QAM64;
4: QAM128;
5: QAM256;
6-255: Reserved.

symbolrate: Symbol rate in MB with value scope of 0-999.9999. 4-bytes BCD code is used to indicate this value just as the 'frequency' field.

fec_inner: Forward Error Correction inner codes.

PCR PID: clock synchronization PID code.

ComponentNum: Number of component streams

CompType: type of component underlying stream.

CompPID: PID of component underlying stream.

ECMPID: PID of ECM pack of descrambled underlying stream CW.

3.16.2 CA_SMS _LOCK_SERVICE_RESPONSE

Data_Cont() of this message is shown in Table 3.36:

Table 3.36 Data_Cont() of CA_SMS _ LOCK_SERVICE_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.17 Release Lock

Release the control panel lock (set by 'Switch Channel' message) for the specified subscribers, addressed by the addressing expression.

3.17.1 SMS _CA_UNLOCK_SERVICE_REQUEST

Data_Cont() of this message is shown in Table 3.37:

Table 3.37 Data_Cont() of SMS _CA_UNLOCK_SERVICE_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
Exp_Len	8	expression length
For(i=0; i< Exp_Len; i++){		
Exp_Char	8	expression
}		
}		

Exp_Len: length of the addressing expression.

Exp_Char: addressing expression, maximum 40 char, should NOT be null.

3.17.2 CA_SMS _UNLOCK_SERVICE_RESPONSE

Data_Cont() of this message is shown in Table 3.38:

Table 3.38 Data_Cont() of CA_SMS _UNLOCK_SERVICE_RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes
}		

3.18 Refresh Card Data

This message is to inform CDCAS3.0 to re-send all EMM packets against the specified smart card. It is useful while the subscribers miss the packets and want the packets to be supplied again quickly.

3.18.1 SMS _CA_CARD_REFRESH_DATA_REQUEST

Data_Cont() of this message is shown in Table 3.39:

Table 3.39 Data_Cont() of SMS _CA_CARD_REFRESH_DATA_REQUEST

SYNTAX	bits	NOTES
Data_Cont(){		
CARD_SN	128	smart card serial number
Reserved	8	fixed to be 0x01
}		

3.18.2 CA_SMS _CARD_REFRESH_DATA_RESPONSE

Data_Cont() of this message is shown in Table 3.40:

Table 3.40 Data_Cont() of CA_SMS _CARD_REFRESH_DATA _RESPONSE

SYNTAX	bits	NOTES
Data_Cont(){		
Erro_Code	32	error codes

}

3.19 High advance and controllable preview

Through this command, SMS can set the Advanced Preview-related parameters of the specified card. When the parameters are 0, then the advanced preview control function is canceled.

3.19.1 SMS_CA_SET_ADV_CONTROL_PREVIEW_REQUEST

Data contents are shown in Table 3.42:

Table 3.42 SMS_CA_SET_ADV_CONTROL_PREVIEW_REQUEST data contents

SYNTAX	bits	NOTE
Data_Cont(){		
CARD_SN	128	Smart card No.
PreviewDuration	8	Free preview time unit
WatchTime	8	Watch time unit
TotalCount	8	Total number of times allowed for watch every day
TotalTime	16	Total time allowed for watch every day
}		

3.19.2 CA_SMS_SET_ADV_CONTROL_PREVIEW_RESPONSE

Data contents are shown in Table 3.43 as follows:

Table 3.43 CA_SMS_SET_ADV_CONTROL_PREVIEW _RESPONSE data contents

SYNTAX	bits	NOTE
Data_Cont(){		
Erro_Code	32	Error code
}		

PreviewDuration: Free preview time unit. The watch time unit is from initial switch by the user to the current channel, where it is allowed for watch all or part, depending on the contents of WatchTime. The unit is minute, ranging from 0 to 255. ;

WatchTime: Watch time unit. Watch time allowed at a PreviewDuration or TotalCount. The unit is minute, ranging from 0 to 255;

TotalCount: Total number of times allowed for watch every day. It is counted as 1 from the user's first time to channel switching, and if the switching happens during the time interval represented by [WatchTime, PreviewDuration), it will not be counted; if channel switching is not done, but the continuous watch time reaches the WatchTime, then it will be counted as 1 when it is longer than PreviewDuration. The unit is 4 times,

ranging from 0 to 63; note: when TotalCount = 1, in fact, 4 times is allowed for watch; when it is 2, 8 times is allowed, and so on.

TotalTime: Total time allowed for watch every day in minute ranging from 0 to 1023.

3.20 Send Super OSD

By this command, SMS can send OSDs with specified font, background color, display area, duration, and other attributes.

3.20.1 SMS_CA_SEND_SUPEROSD_REQUEST

Data contents are shown in Table 3.44:

Table 3.44 SMS_CA_SEND_SUPEROSD_REQUEST data contents

SYNTAX	bits	NOTE
Data_Cont(){		
Exp_Len	8	Length of expression
For(i=0; i< Exp_Len; i++){		
Exp_Char	8	Expression
}		
Cont_Len	16	Length of context
For(i=0; i< Cont_Len; i++){		
Cont_Char	8	Context
}		
Style	8	Display style
Duration	32	Duration
ForcedDisplay	8	Forced display or not
FontSize	8	Font size
FontColor	8	Font color
BackgroundColor	8	Background color
BackgroundArea	8	Background area
}		

Exp_Len: Length of expression. (Up to 40char, and can not be empty)

Exp_Char: Bytes of expression.

Cont_Len: Length of OSD context. (Up to 256char, and can not be empty)

Cont_Char: Bytes of OSD context.

Style: Display style, with the range of values being 1 and 2. 1 indicates the scrolling display from left to right at the top of the screen; 2 indicates the scrolling display from left to right at the bottom of the screen.

Duration: Duration in second(s) ranging from 1 to 900.

ForcedDisplay: 0: unforced display, 1: forced display

FontSize: 0: default 1: large, 2: small

FontColor: Support 256 colors

BackgroundColor: Support 256 colors

BackgroundArea: indicates the percentage of the area in the central portion of the

screen, with the default of 80 and the range from 20 to 80

3.20.2 CA_SMS_SEND_SUPEROSD_RESPONSE

Data contents are shown in Table 3.45:

Table 3.45 CA_SMS_SEND_SUPEROSD_RESPONSE data contents

SYNTAX	bits	NOTE
Data_Cont(){		
Erro_Code	32	Error code
}		

4 Suggestions for SMS Implementation

4.1 Reducing the Number of Commands

In some cases, SMS could reduce the number of commands to improve the efficiency of CDCAS3.0_SMS interface, such as, the case of the subscriber renewing his subscription.

Here is an example. A subscriber purchases product 1 with service period from January 1st, 2007 to December 31st 2007. He renews his service on December 25th, 2007 to order the service until end of next year, that is, service period is renewed to January 1st 2008 to December 31st 2008. In this case, SMS only needs to send one command to update the service period of product 1 (January 1st 2008 to December 31st 2008). It's no necessary to send two commands: canceling the entitlement on December 31st 2007 and then sending a new entitlement on January 1st 2008.

4.2 Refreshing Data

If the subscriber does not receive the data, the operator could use 'Refresh card data' message to timely send the relevant data to this subscriber. The message details are described in Section 3.18.

4.3 Recharging for IPPV Service

Recharging operation is used for IPPV service. In CDCAS3.0, recharging is a process that SMS sets up the credit limit in the E-Slot of the smart card, using the message

'Charge E-slot' (Section 3.9). The value in this message is the total credit of the E-slot, including which has been consumed. So, SMS should make sure that the credit limit value here should be the total amount of the history recharges.

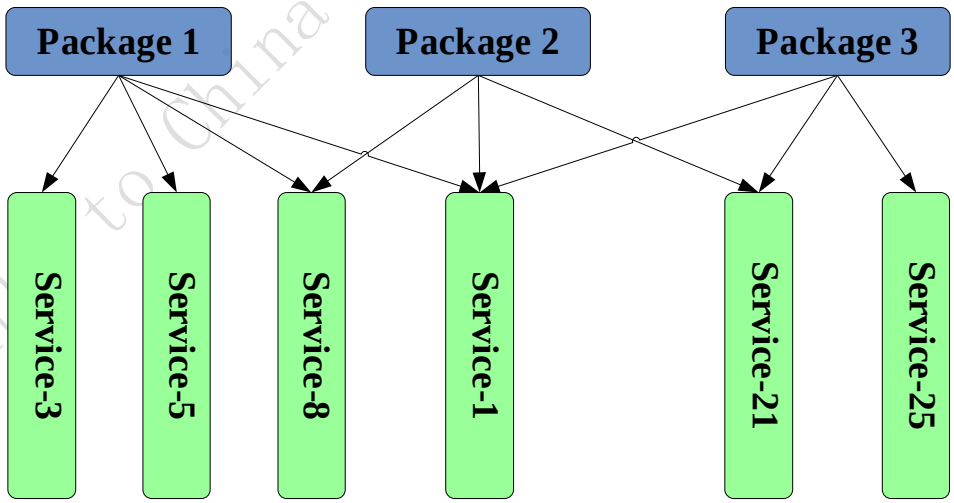
The credit limit is indicated in points, and the exchange rate between the points and the real money could be defined by the operator.

Take an example here. Provided 1 dollar equal to 1 point, a card has the credit limit of 200 points currently, and the subscriber recharges 800 dollars. Then, the subscriber has totally recharged 1000 dollar, SMS should set credit limit value in the 'Charge E-slot' message to 1000, and the smart card will change the value of credit limit to 1000 after receiving the command.

4.4 Package of Services

CDCAS3.0 supports flexible programs package strategy. The operator can freely pack different services into one package, and pack one service into different packages as well. The services could be sold based on the packages .

For example: Services 1, 3, 5 & 8 could be packed into Product package 1, and Services 8, 1 & 21 could be packed into Product package 2, and Services 1, 21, & 25 could be packed into Product package 3. See figure below:



Note: SMS send the entitlement commands based on the products rather than the services.

For the above example, provided the subscribers want to watch Services 3, 5, and 8,

SMS just needs to send entitlement of Product 1. Provided the subscribers want to watch Services 1, SMS could send the entitlement either of Product 1, 2 or 3.

Appendix 1 Message ID

Table A1.1 Message ID List

Message ID	Value
SMS _CA_CREATE_SESSION_REQUEST	0x0001
CA_SMS _CREATE_SESSION_RESPONSE	0x8001
SMS _CA_OPEN_ACCOUNT_REQUEST	0x0201
CA_SMS _OPEN_ACCOUNT_RESPONSE	0x8201
SMS _CA_STOP_ACCOUNT_REQUEST	0x0202
CA_SMS _STOP_ACCOUNT_RESPONSE	0x8202
SMS _CA_SET_LOCK_REQUEST	0x0203
CA_ SMS _SET_LOCK_ RESPONSE	0x8203
SMS _CA_SET_UNLOCK_REQUEST	0x0204
CA_ SMS _SET_UNLOCK_ RESPONSE	0x8204
SMS _CA_REPAIR_REQUEST	0x0206
CA_SMS _REPAIR_RESPONSE	0x8206
SMS _CA_RESETCARDPIN_REQUEST	0x0207
CA_SMS _RESETCARDPIN_RESPONSE	0x8207
SMS _CA_SETSLOTMONEY_REQUEST	0x0208
CA_SMS _SETSLOTMONEY_RESPONSE	0x8208
SMS _CA_SET_CHARACTER_REQUEST	0x020C
CA_SMS _ SET_CHARACTER_RESPONSE	0x820C
SMS _CA_ENTITLE_REQUEST	0x020D
CA_SMS _ENTITLE_RESPONSE	0x820D
SMS _CA_ENTITLEEXT_REQUEST	0x020E
CA_SMS _ENTITLEEXT_RESPONSE	0x820E
SMS_CA_SET_ADV_CONTROL_PREVIEW_REQUEST	0x020F
CA_SMS_SET_ADV_CONTROL_PREVIEW_RESPONSE	0x820F
SMS _CA_SEND_EMAIL_REQUEST	0x0301
CA_SMS _SEND_EMAIL_RESPONSE	0x8301
SMS _CA_SEND_OSD_REQUEST	0x0304
CA_SMS _SEND_OSD_RESPONSE	0x8304
SMS _CA_LOCK_SERVICE_REQUEST	0x0306
CA_SMS _LOCK_SERVICE_RESPONSE	0x8306
SMS _CA_UNLOCK_SERVICE_REQUEST	0x0307
CA_SMS _UNLOCK_SERVICE_RESPONSE	0x8307
SMS_CA_SEND_SUPEROSD_REQUEST	0x0308
CA_SMS_SEND_SUPEROSD_ RESPONSE	0x8308
SMS _CA_CARD_REFRESH_DATA_REQUEST	0x0401
CA_SMS _CARD_REFRESH_DATA_RESPONSE	0x8401
SMS _CA_SET_CHILD_REQUEST	0x0402
CA _SMS _SET_CHILD_RESPONSE	0x8402
SMS _CA_CANCEL_CHILD_REQUEST	0x0403
CA_SMS _CANCEL_CHILD_RESPONSE	0x8403

Appendix 2 Error Codes

Table A2.1 Error Codes List

Codes	Value	Meaning
TFCA_OK	0x0000	Successful execution
CARD_NOT_EXIST	0x0002	Card not exists
CARD_NOT_OPEN	0x0003	Account not opened yet
BATCH_ENTITLE_PID_REPEAT	0x000A	Batch entitle instruction has repeated product numbers
SLOT_NOT_EXIST	0x000B	wallet to recharge does not exist
INVILID_PROD	0x000C	illegal product number
INVILID_TIME	0x000D	Illegal time
ENTITLE_NOT_EXSIT	0x000E	When de-entitle, it's found that entitle does not exist
ENTITLE_EXCEED_SECTION_LIMIT	0x000F	Reaching upper limit of entitle records
ADDRCMD_EXPRESSION_TOO_LONG	0x0014	addressing expression too long
CONENT_TOO_LONG	0x0016	contents too long
TITLE_TOO_LONG	0x0017	Title too long
DURATION_TOO_LONG	0x0018	OSD duration out of range
ADDRCMD_EXPRESSION_WRONG	0x001D	expression error (including number of left and right brackets not correct)
INVALID_OSD_STYLE	0x0020	Invalid OSD display style
PERSONALMAP_TYPE_INVALID	0x0027	illegal personal information
TFCAERR_MSG_INVILID	0xBBBB	illegal message
INTERFACE_UNDER_CONSTRUCTION	0xCCCC	interface under construction
CASSERVER_INNER_ERROR	0xDDDD	CAS internal error
SETCHILD_PARENT_ERR	0x0029	setting child card error: parent card is child card
SETCHILD_CHILD_ERR_HAVEBEPARENT	0x002A	setting child card error: child card is parent card of other card
SETCHILD_CHILD_ERR_HAVEBECHILD	0x002B	setting child card error: child card is child card of other card
CANCELCHILD_ERR	0x002C	canceling child card error: child card canceled
CARD_NOT_SUPPORT	0x002D	Card not supporting current operation
INVALID_ TOTALCOUNT	0x0031	Total number of times allowed for watch the controllable preview error
INVALID_ TOTALTIME	0x0032	Total time allowed for watch the controllable preview error
INVALID_ FORCEDISPLAY	0x0033	Forced display OSD error
INVALID_ FONTSIZE	0x0034	OSD font size error

INVALID_ BACKGROUNDAREA	0x0035	OSD background area error
SYSTEM_NOT_SUPPORT	0x003C	System not supporting current operation

Appendix 3 Property Items

The usage of property items is shown as below:

Property Index	Index	Usage
0	Area	Area code
1	Bouquet	Identify that the smart card belongs to which bouquet. Different services might be offered for different bouquets.
2	Reserved	Reserved for further use.
3	character 0	Defined by SMS
4	character 1	Defined by SMS
5	character 2	Defined by SMS
6	character 3	Defined by SMS
7	character 4	Reserved for indicating high 4 bytes of STB number
8	character 5	Reserved for indicating low 4 bytes of STB number
9	character 6	Defined by SMS

Appendix 4 Addressing Expression

1 Addressing Elements in CDCAS3.0

Elements Type	Label	Remarks
Subscriber card number	“card”	Inner ID of the smart card
ordered product number	“prod”	Against subscribers who has ordered certain products
current service number	“serv”	Against subscribers who are currently watching certain services
character 0 (area)	“area”	User area
Bouquet	“bouq”	Users can be divided into different bouquets to realize different demands, for example, display of different channel list.
character 1	“cha1”	
character 2	“cha2”	
character 3	“cha3”	
character 4	“cha4”	
character 5	“cha5”	
character 6	“cha6”	

2 Operators

The character list for the operators is shown as below:

Type	Contents	Character
Arithmetic operation operator	bitwise AND	*
	bitwise OR	+
	Greater than	>
	Less than	<
	Be equal to	=
Logical operation operator	AND	&
	OR	|
	NOT	~
	Left bracket	(
	Right bracket	)

3 Expression

1) The addressing expression is represented with character strings, which consists of addressing elements and operators.

2) The value in the expression could be represented in hexadecimal or decimal. If the value is in hexadecimal, '0x' prefix must be added.
3) For 'subscriber card number' element, the value should be either 0, or an existing card number, and it should be represented in decimal.

[Examples]

Example 1: area code equal to 20

"area=20"

"area=0x14"

Example 2:

"(~(cha2<234))&(cha3>675)"

Example 3:

"card>0"

"card>8000302100016651"

4 Addressing Elements Supported in Commands

Commands / Addressing Elements	OSD	E-Mail	Switch Channel / Release Lock
subscriber card number	√	√	√
ordered product number	√	×	×
current service number	√	×	×
characters	√	√	×

5 Operators Supported by Addressing Elements

Operators / Addressing Elements	*	+	>	<	=
subscriber card number	√	√	√	√	√
ordered product number	×	×	×	×	√
current service number	×	×	×	×	√
characters	√	√	√	√	√

Appendix 5 MD5 Codes

1 MD5.h file

```
#ifndef CDCAS3.0_MD5_H
#define CDCAS3.0_MD5_H

#define S11 7
#define S12 12
#define S13 17
#define S14 22
#define S21 5
#define S22 9
#define S23 14
#define S24 20
#define S31 4
#define S32 11
#define S33 16
#define S34 23
#define S41 6
#define S42 10
#define S43 15
#define S44 21
typedef unsigned  long uint4;
typedef unsigned  short uint2;
typedef unsigned  char uint1;

#ifdef __cplusplus
extern "C" {
#endif
unsigned char TFCA_MD5(unsigned char *pInPut,uint4 dwInput_Length,unsigned char
*pOutPut);
/*
TFCA_MD5 performs data summary. Parameter meaning as follows:
pInPut: point to first address of data to have data summary
dwInput_Length:length of data for summary, with unit of bytes
pOutPut: point to first pointer of returned MAC data
*/
#ifdef __cplusplus
}
#endif

#endif //CDCAS3.0_MD5_h
```

2 MD5.c

```
#include <stdio.h>
#include <string.h>
#include "md5.h"

void encode (uint1 *output, uint4 *input, uint4 len)
{
 uint4 i, j;

 for (i = 0, j = 0; j < len; i++, j += 4)
 {
 output[j] = (uint1) (input[i] & 0xff);
 output[j+1] = (uint1) ((input[i] >> 8) & 0xff);
 output[j+2] = (uint1) ((input[i] >> 16) & 0xff);
 output[j+3] = (uint1) ((input[i] >> 24) & 0xff);
 }
}
void decode(uint4 *output, uint1 *input, uint4 len)
{
 uint4 i, j;

 for (i = 0, j = 0; j < len; i++, j += 4)
 output[i] = ((uint4)input[j]) | (((uint4)input[j+1]) << 8) |
  (((uint4)input[j+2]) << 16) | (((uint4)input[j+3]) << 24);
}

uint4 rotate_left(uint4 x, uint4 n)
{
 return (x << n) | (x >> (32-n)) ;
}

uint4 F(uint4 x, uint4 y, uint4 z)
{
 return (x & y) | (~x & z);
}

void FF(uint4 * a, uint4 b, uint4 c, uint4 d, uint4 x,uint4 s, uint4 ac)
{
        *a += F(b, c, d) + x + ac;
        *a = rotate_left (*a, s) +b;
}

uint4 G(uint4 x, uint4 y, uint4 z)
{
 return (x & z) | (y & ~z);
}

void GG(uint4 *a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac)
{
        *a += G(b, c, d) + x + ac;
        *a = rotate_left (*a, s) +b;
}

uint4 H(uint4 x, uint4 y, uint4 z)
```

```
{
 return x ^ y ^ z;
}
void HH(uint4 *a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac)
{
        *a += H(b, c, d) + x + ac;
        *a = rotate_left (*a, s) +b;
}

uint4 I(uint4 x, uint4 y, uint4 z)
{
 return y ^ (x | ~z);
}

void II(uint4 *a, uint4 b, uint4 c, uint4 d, uint4 x, uint4 s, uint4 ac)
{
        *a += I(b, c, d) + x + ac;
        *a = rotate_left (*a, s) +b;
}

void transform (uint1 block[64],uint4 *pstate,uint1 *pfinalized){

 uint4 a = pstate[0], b = pstate[1], c = pstate[2], d = pstate[3], x[16];
 decode (x, block, 64);

 /* Round 1 */
 FF(&a, b, c, d, x[ 0], S11, 0xd76aa478); /* 1 */
 FF(&d, a, b, c, x[ 1], S12, 0xe8c7b756); /* 2 */
 FF(&c, d, a, b, x[ 2], S13, 0x242070db); /* 3 */
 FF(&b, c, d, a, x[ 3], S14, 0xc1bdceee); /* 4 */
 FF(&a, b, c, d, x[ 4], S11, 0xf57c0faf); /* 5 */
 FF(&d, a, b, c, x[ 5], S12, 0x4787c62a); /* 6 */
 FF(&c, d, a, b, x[ 6], S13, 0xa8304613); /* 7 */
 FF(&b, c, d, a, x[ 7], S14, 0xfd469501); /* 8 */
 FF(&a, b, c, d, x[ 8], S11, 0x698098d8); /* 9 */
 FF(&d, a, b, c, x[ 9], S12, 0x8b44f7af); /* 10 */
 FF(&c, d, a, b, x[10], S13, 0xffff5bb1); /* 11 */
 FF(&b, c, d, a, x[11], S14, 0x895cd7be); /* 12 */
 FF(&a, b, c, d, x[12], S11, 0x6b901122); /* 13 */
 FF(&d, a, b, c, x[13], S12, 0xfd987193); /* 14 */
 FF(&c, d, a, b, x[14], S13, 0xa679438e); /* 15 */
 FF(&b, c, d, a, x[15], S14, 0x49b40821); /* 16 */

 /* Round 2 */
 GG(&a, b, c, d, x[ 1], S21, 0xf61e2562); /* 17 */
 GG(&d, a, b, c, x[ 6], S22, 0xc040b340); /* 18 */
 GG(&c, d, a, b, x[11], S23, 0x265e5a51); /* 19 */
 GG(&b, c, d, a, x[ 0], S24, 0xe9b6c7aa); /* 20 */
 GG(&a, b, c, d, x[ 5], S21, 0xd62f105d); /* 21 */
 GG(&d, a, b, c, x[10], S22, 0x2441453); /* 22 */
 GG(&c, d, a, b, x[15], S23, 0xd8a1e681); /* 23 */
 GG(&b, c, d, a, x[ 4], S24, 0xe7d3fbc8); /* 24 */
 GG(&a, b, c, d, x[ 9], S21, 0x21e1cde6); /* 25 */
 GG(&d, a, b, c, x[14], S22, 0xc33707d6); /* 26 */
 GG(&c, d, a, b, x[ 3], S23, 0xf4d50d87); /* 27 */
 GG(&b, c, d, a, x[ 8], S24, 0x455a14ed); /* 28 */
```

```
GG(&a, b, c, d, x[13], S21, 0xa9e3e905); /* 29 */
GG(&d, a, b, c, x[ 2], S22, 0xfcefa3f8); /* 30 */
GG(&c, d, a, b, x[ 7], S23, 0x676f02d9); /* 31 */
GG(&b, c, d, a, x[12], S24, 0x8d2a4c8a); /* 32 */

/* Round 3 */
HH(&a, b, c, d, x[ 5], S31, 0xfffa3942); /* 33 */
HH(&d, a, b, c, x[ 8], S32, 0x8771f681); /* 34 */
HH(&c, d, a, b, x[11], S33, 0x6d9d6122); /* 35 */
HH(&b, c, d, a, x[14], S34, 0xfde5380c); /* 36 */
HH(&a, b, c, d, x[ 1], S31, 0xa4beea44); /* 37 */
HH(&d, a, b, c, x[ 4], S32, 0x4bdecfa9); /* 38 */
HH(&c, d, a, b, x[ 7], S33, 0xf6bb4b60); /* 39 */
HH(&b, c, d, a, x[10], S34, 0xbebfbc70); /* 40 */
HH(&a, b, c, d, x[13], S31, 0x289b7ec6); /* 41 */
HH(&d, a, b, c, x[ 0], S32, 0xeaa127fa); /* 42 */
HH(&c, d, a, b, x[ 3], S33, 0xd4ef3085); /* 43 */
HH(&b, c, d, a, x[ 6], S34, 0x4881d05); /* 44 */
HH(&a, b, c, d, x[ 9], S31, 0xd9d4d039); /* 45 */
HH(&d, a, b, c, x[12], S32, 0xe6db99e5); /* 46 */
HH(&c, d, a, b, x[15], S33, 0x1fa27cf8); /* 47 */
HH(&b, c, d, a, x[ 2], S34, 0xc4ac5665); /* 48 */

/* Round 4 */
II(&a, b, c, d, x[ 0], S41, 0xf4292244); /* 49 */
II(&d, a, b, c, x[ 7], S42, 0x432aff97); /* 50 */
II(&c, d, a, b, x[14], S43, 0xab9423a7); /* 51 */
II(&b, c, d, a, x[ 5], S44, 0xfc93a039); /* 52 */
II(&a, b, c, d, x[12], S41, 0x655b59c3); /* 53 */
II(&d, a, b, c, x[ 3], S42, 0x8f0ccc92); /* 54 */
II(&c, d, a, b, x[10], S43, 0xffeff47d); /* 55 */
II(&b, c, d, a, x[ 1], S44, 0x85845dd1); /* 56 */
II(&a, b, c, d, x[ 8], S41, 0x6fa87e4f); /* 57 */
II(&d, a, b, c, x[15], S42, 0xfe2ce6e0); /* 58 */
II(&c, d, a, b, x[ 6], S43, 0xa3014314); /* 59 */
II(&b, c, d, a, x[13], S44, 0x4e0811a1); /* 60 */
II(&a, b, c, d, x[ 4], S41, 0xf7537e82); /* 61 */
II(&d, a, b, c, x[11], S42, 0xbd3af235); /* 62 */
II(&c, d, a, b, x[ 2], S43, 0x2ad7d2bb); /* 63 */
II(&b, c, d, a, x[ 9], S44, 0xeb86d391); /* 64 */

pstate[0] += a;
pstate[1] += b;
pstate[2] += c;
pstate[3] += d;
memset ( (uint1 *) x, 0, sizeof(x));
}
void update (uint1 *input, uint4 input_length,uint4 *pstate,uint1 *pfinalized,uint4 *pcount,uint1
*pbuffer) {

uint4 input_index, buffer_index;
uint4 buffer_space;

if (*pfinalized)
return;
buffer_index = (uint4)((pcount[0] >> 3) & 0x3F);
```

```
 if ( (pcount[0] += ((uint4) input_length << 3))<((uint4) input_length << 3) )
 pcount[1]++;

 pcount[1] += ((uint4)input_length >> 29);

 buffer_space = 64 - buffer_index;

 if (input_length >= buffer_space)
 {
 memcpy (pbuffer + buffer_index, input, buffer_space);
 transform (pbuffer,pstate,pfinalized);

 for (input_index = buffer_space; input_index + 63 < input_length; input_index += 64)
  transform (input+input_index,pstate,pfinalized);

 buffer_index = 0;
 }
 else
 input_index=0;

 memcpy(pbuffer+buffer_index, input+input_index, input_length-input_index);
}

void finalize(uint4 *pstate,uint1 *pfinalized,uint4 *pcount,uint1 *pbuffer,uint1 *pOutPut)
{
 unsigned char bits[8];
 uint4 index, padLen;
 uint1 PADDING[64]=
 {
 0x80, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
 };

 if (*pfinalized)
 return;

 encode (bits, pcount, 8);
 index = (uint4) ((pcount[0] >> 3) & 0x3f);
 padLen = (index < 56) ? (56 - index): (120 - index);
 update (PADDING, padLen,pstate,pfinalized,pcount,pbuffer);
 update (bits, 8,pstate,pfinalized,pcount,pbuffer);
 encode (pOutPut, pstate, 16);
 memset (pbuffer, 0, sizeof(*pbuffer));
 *pfinalized=1;
}

unsigned char TFCA_MD5(unsigned char *pInPut,uint4 dwInput_Length,unsigned char
*pOutPut)
{
        uint4 state[4];
        uint4 count[2];
        uint1 buffer[64];
        uint1 finalized;
```

```
        finalized=0;
        count[0] = 0;
        count[1] = 0;

        state[0] = 0x67452301;
        state[1] = 0xefcdab89;
        state[2] = 0x98badcfe;
        state[3] = 0x10325476;

        update(pInPut, dwInput_Length,state,&finalized,count,buffer);
        finalize(state,&finalized,count,buffer,pOutPut);
        return 0x00;
}
```

Appendix 6 3DES Codes

1 d3des.h file

```
#define EN0     0       /* MODE == encrypt */
#define DE1     1       /* MODE == decrypt */

#ifdef __cplusplus
extern "C"{
#endif
/* A useful alias on 68000-ish machines, but NOT USED HERE. */

typedef union {
        unsigned long blok[2];
        unsigned short word[4];
        unsigned char byte[8];
        } M68K;

extern void deskey(unsigned char *, short);
/*                  hexkey[8]  MODE
 * Sets the internal key register according to the hexadecimal
 * key contained in the 8 bytes of hexkey, according to the DES,
 * for encryption or decryption according to MODE.
 */

extern void usekey(unsigned long *);
/*                  cookedkey[32]
 * Loads the internal key register with the data in cookedkey.
 */

extern void cpkey(unsigned long *);
/*                  cookedkey[32]
 * Copies the contents of the internal key register into the storage
 * located at &cookedkey[0].
 */

extern void des(unsigned char *, unsigned char *);
/*                  from[8]  to[8]
 * Encrypts/Decrypts (according to the key currently loaded in the
 * internal key register) one block of eight bytes at address 'from'
 * into the block at address 'to'. They can be the same.
 */

extern void des2key(unsigned char *, short);
/*                  hexkey[16]  MODE
 * Sets the internal key registerS according to the hexadecimal
 * keyS contained in the 16 bytes of hexkey, according to the DES,
 * for DOUBLE encryption or decryption according to MODE.
 * NOTE: this clobbers all three key registers!
 */
```

```
extern void Ddes(unsigned char *, unsigned char *);
/*                    from[8]  to[8]
 * Encrypts/Decrypts (according to the keyS currently loaded in the
 * internal key registerS) one block of eight bytes at address 'from'
 * into the block at address 'to'. They can be the same.
 */
bool TFCA_3DES(bool bEnspot, unsigned char* pbyKey,int nLength, unsigned char* pbySource,
unsigned char* pbyTarget)
bool TFCA_DES(bool bEnspot, unsigned char* pbyKey,int nLength, unsigned char* pbySource,
unsigned char* pbyTarget)
/*
TFCA_3DES performs 3DES encryption, and TFCA_DES performs DES encryption. Parameter
meaning as follows:
bEnspot: truemeans encryption, falsemeans decryption
pbyKey: point to key for encryption
nLength: length of encryption data, with unit of bytes, should be multiples of 8
pbySource: point to data first pointer for encryption
pbyTarget: point to first pointer of returned encrypted data
*/

#ifdef __cplusplus
}
#endif
```

2 d3des.c

```
#include "d3des.h"

static void scrunch(unsigned char *, unsigned long *);
static void unscrun(unsigned long *, unsigned char *);
static void desfunc(unsigned long *, unsigned long *);
static void cookey(unsigned long *);

static unsigned long KnL[32] = { 0L };
static unsigned long KnR[32] = { 0L };
static unsigned long Kn3[32] = { 0L };
static unsigned char Df_Key[24] = {
        0x01,0x23,0x45,0x67,0x89,0xab,0xcd,0xef,
        0xfe,0xdc,0xba,0x98,0x76,0x54,0x32,0x10,
        0x89,0xab,0xcd,0xef,0x01,0x23,0x45,0x67 };

static unsigned short bytebit[8]    = {
        0200, 0100, 040, 020, 010, 04, 02, 01 };

static unsigned long bigbyte[24] = {
        0x800000L,      0x400000L,      0x200000L,      0x100000L,
        0x80000L,       0x40000L,       0x20000L,       0x10000L,
        0x8000L,        0x4000L,        0x2000L,        0x1000L,
        0x800L,         0x400L,         0x200L,         0x100L,
        0x80L,          0x40L,          0x20L,          0x10L,
        0x8L,           0x4L,           0x2L,           0x1L    };

/* Use the key schedule specified in the Standard (ANSI X3.92-1981). */

static unsigned char pc1[56] = {
        56, 48, 40, 32, 24, 16, 8,   0, 57, 49, 41, 33, 25, 17,
         9, 1, 58, 50, 42, 34, 26,  18, 10, 2, 59, 51, 43, 35,
        62, 54, 46, 38, 30, 22, 14, 6, 61, 53, 45, 37, 29, 21,
        13, 5, 60, 52, 44, 36, 28,  20, 12, 4, 27, 19, 11, 3 };

static unsigned char totrot[16] = {
        1,2,4,6,8,10,12,14,15,17,19,21,23,25,27,28 };

static unsigned char pc2[48] = {
        13, 16, 10, 23, 0, 4, 2, 27, 14, 5, 20, 9,
        22, 18, 11, 3, 25, 7, 15, 6, 26, 19, 12, 1,
        40, 51, 30, 36, 46, 54, 29, 39, 50, 44, 32, 47,
        43, 48, 38, 55, 33, 52, 45, 41, 49, 35, 28, 31 };

void deskey(key, edf)       /* Thanks to James Gillogly & Phil Karn! */
unsigned char *key;
short edf;
{
        register int i, j, l, m, n;
        unsigned char pc1m[56], pcr[56];
        unsigned long kn[32];

        for ( j = 0; j < 56; j++ ) {
                l = pc1[j];
```

```
                m = l & 07;
                pc1m[j] = (key[l >> 3] & bytebit[m]) ? 1: 0;
                }
        for( i = 0; i < 16; i++ ) {
                if( edf == DE1 ) m = (15 - i) << 1;
                else m = i << 1;
                n = m + 1;
                kn[m] = kn[n] = 0L;
                for( j = 0; j < 28; j++ ) {
                        l = j + totrot[i];
                        if( l < 28 ) pcr[j] = pc1m[l];
                        else pcr[j] = pc1m[l - 28];
                        }
                for( j = 28; j < 56; j++ ) {
                 l = j + totrot[i];
                 if( l < 56 ) pcr[j] = pc1m[l];
                 else pcr[j] = pc1m[l - 28];
                 }
                for( j = 0; j < 24; j++ ) {
                        if( pcr[pc2[j]] ) kn[m] |= bigbyte[j];
                        if( pcr[pc2[j+24]] ) kn[n] |= bigbyte[j];
                        }
                }
        cookey(kn);
        return;
        }

static void cookey(raw1)
register unsigned long *raw1;
{
        register unsigned long *cook, *raw0;
        unsigned long dough[32];
        register int i;

        cook = dough;
        for( i = 0; i < 16; i++, raw1++ ) {
                raw0 = raw1++;
                *cook    = (*raw0 & 0x00fc0000L) << 6;
                *cook    |= (*raw0 & 0x00000fc0L) << 10;
                *cook    |= (*raw1 & 0x00fc0000L) >> 10;
                *cook++ |= (*raw1 & 0x00000fc0L) >> 6;
                *cook    = (*raw0 & 0x0003f000L) << 12;
                *cook    |= (*raw0 & 0x0000003fL) << 16;
                *cook    |= (*raw1 & 0x0003f000L) >> 4;
                *cook++ |= (*raw1 & 0x0000003fL);
                }
        usekey(dough);
        return;
        }

void cpkey(into)
register unsigned long *into;
{
        register unsigned long *from, *endp;

        from = KnL, endp = &KnL[32];
```

```
        while( from < endp ) *into++ = *from++;
        return;
        }

void usekey(from)
register unsigned long *from;
{
        register unsigned long *to, *endp;

        to = KnL, endp = &KnL[32];
        while( to < endp ) *to++ = *from++;
        return;
        }

void des(inblock, outblock)
unsigned char *inblock, *outblock;
{
        unsigned long work[2];

        scrunch(inblock, work);
        desfunc(work, KnL);
        unscrun(work, outblock);
        return;
        }

static void scrunch(outof, into)
register unsigned char *outof;
register unsigned long *into;
{
        *into     = (*outof++ & 0xffL) << 24;
        *into    |= (*outof++ & 0xffL) << 16;
        *into    |= (*outof++ & 0xffL) << 8;
        *into++ |= (*outof++ & 0xffL);
        *into     = (*outof++ & 0xffL) << 24;
        *into    |= (*outof++ & 0xffL) << 16;
        *into    |= (*outof++ & 0xffL) << 8;
        *into    |= (*outof & 0xffL);
        return;
        }

static void unscrun(outof, into)
register unsigned long *outof;
register unsigned char *into;
{
        *into++ = (*outof >> 24) & 0xffL;
        *into++ = (*outof >> 16) & 0xffL;
        *into++ = (*outof >> 8) & 0xffL;
        *into++ = *outof++       & 0xffL;
        *into++ = (*outof >> 24) & 0xffL;
        *into++ = (*outof >> 16) & 0xffL;
        *into++ = (*outof >> 8) & 0xffL;
        *into    = *outof        & 0xffL;
        return;
        }

static unsigned long SP1[64] = {
```

```
    0x01010400L, 0x00000000L, 0x00010000L, 0x01010404L,
    0x01010004L, 0x00010404L, 0x00000004L, 0x00010000L,
    0x00000400L, 0x01010400L, 0x01010404L, 0x00000400L,
    0x01000404L, 0x01010004L, 0x01000000L, 0x00000004L,
    0x00000404L, 0x01000400L, 0x01000400L, 0x00010400L,
    0x00010400L, 0x01010000L, 0x01010000L, 0x01000404L,
    0x00010004L, 0x01000004L, 0x01000004L, 0x00010004L,
    0x00000000L, 0x00000404L, 0x00010404L, 0x01000000L,
    0x00010000L, 0x01010404L, 0x00000004L, 0x01010000L,
    0x01010400L, 0x01000000L, 0x01000000L, 0x00000400L,
    0x01010004L, 0x00010000L, 0x00010400L, 0x01000004L,
    0x00000400L, 0x00000004L, 0x01000404L, 0x00010404L,
    0x01010404L, 0x00010004L, 0x01010000L, 0x01000404L,
    0x01000004L, 0x00000404L, 0x00010404L, 0x01010400L,
    0x00000404L, 0x01000400L, 0x01000400L, 0x00000000L,
    0x00010004L, 0x00010400L, 0x00000000L, 0x01010004L };

static unsigned long SP2[64] = {
    0x80108020L, 0x80008000L, 0x00008000L, 0x00108020L,
    0x00100000L, 0x00000020L, 0x80100020L, 0x80008020L,
    0x80000020L, 0x80108020L, 0x80108000L, 0x80000000L,
    0x80008000L, 0x00100000L, 0x00000020L, 0x80100020L,
    0x00108000L, 0x00100020L, 0x80008020L, 0x00000000L,
    0x80000000L, 0x00008000L, 0x00108020L, 0x80100000L,
    0x00100020L, 0x80000020L, 0x00000000L, 0x00108000L,
    0x00008020L, 0x80108000L, 0x80100000L, 0x00008020L,
    0x00000000L, 0x00108020L, 0x80100020L, 0x00100000L,
    0x80008020L, 0x80100000L, 0x80108000L, 0x00008000L,
    0x80100000L, 0x80008000L, 0x00000020L, 0x80108020L,
    0x00108020L, 0x00000020L, 0x00008000L, 0x80000000L,
    0x00008020L, 0x80108000L, 0x00100000L, 0x80000020L,
    0x00100020L, 0x80008020L, 0x80000020L, 0x00100020L,
    0x00108000L, 0x00000000L, 0x80008000L, 0x00008020L,
    0x80000000L, 0x80100020L, 0x80108020L, 0x00108000L };

static unsigned long SP3[64] = {
    0x00000208L, 0x08020200L, 0x00000000L, 0x08020008L,
    0x08000200L, 0x00000000L, 0x00020208L, 0x08000200L,
    0x00020008L, 0x08000008L, 0x08000008L, 0x00020000L,
    0x08020208L, 0x00020008L, 0x08020000L, 0x00000208L,
    0x08000000L, 0x00000008L, 0x08020200L, 0x00000200L,
    0x00020200L, 0x08020000L, 0x08020008L, 0x00020208L,
    0x08000208L, 0x00020200L, 0x00020000L, 0x08000208L,
    0x00000008L, 0x08020208L, 0x00000200L, 0x08000000L,
    0x08020200L, 0x08000000L, 0x00020008L, 0x00000208L,
    0x00020000L, 0x08020200L, 0x08000200L, 0x00000000L,
    0x00000200L, 0x00020008L, 0x08020208L, 0x08000200L,
    0x08000008L, 0x00000200L, 0x00000000L, 0x08020008L,
    0x08000208L, 0x00020000L, 0x08000000L, 0x08020208L,
    0x00000008L, 0x00020208L, 0x00020200L, 0x08000008L,
    0x08020000L, 0x08000208L, 0x00000208L, 0x08020000L,
    0x00020208L, 0x00000008L, 0x08020008L, 0x00020200L };

static unsigned long SP4[64] = {
    0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
    0x00802080L, 0x00800081L, 0x00800001L, 0x00002001L,
```

```
        0x00000000L, 0x00802000L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00800080L, 0x00800001L,
        0x00000001L, 0x00002000L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002001L, 0x00002080L,
        0x00800081L, 0x00000001L, 0x00002080L, 0x00800080L,
        0x00002000L, 0x00802080L, 0x00802081L, 0x00000081L,
        0x00800080L, 0x00800001L, 0x00802000L, 0x00802081L,
        0x00000081L, 0x00000000L, 0x00000000L, 0x00802000L,
        0x00002080L, 0x00800080L, 0x00800081L, 0x00000001L,
        0x00802001L, 0x00002081L, 0x00002081L, 0x00000080L,
        0x00802081L, 0x00000081L, 0x00000001L, 0x00002000L,
        0x00800001L, 0x00002001L, 0x00802080L, 0x00800081L,
        0x00002001L, 0x00002080L, 0x00800000L, 0x00802001L,
        0x00000080L, 0x00800000L, 0x00002000L, 0x00802080L };

static unsigned long SP5[64] = {
        0x00000100L, 0x02080100L, 0x02080000L, 0x42000100L,
        0x00080000L, 0x00000100L, 0x40000000L, 0x02080000L,
        0x40080100L, 0x00080000L, 0x02000100L, 0x40080100L,
        0x42000100L, 0x42080000L, 0x00080100L, 0x40000000L,
        0x02000000L, 0x40080000L, 0x40080000L, 0x00000000L,
        0x40000100L, 0x42080100L, 0x42080100L, 0x02000100L,
        0x42080000L, 0x40000100L, 0x00000000L, 0x42000000L,
        0x02080100L, 0x02000000L, 0x42000000L, 0x00080100L,
        0x00080000L, 0x42000100L, 0x00000100L, 0x02000000L,
        0x40000000L, 0x02080000L, 0x42000100L, 0x40080100L,
        0x02000100L, 0x40000000L, 0x42080000L, 0x02080100L,
        0x40080100L, 0x00000100L, 0x02000000L, 0x42080000L,
        0x42080100L, 0x00080100L, 0x42000000L, 0x42080100L,
        0x02080000L, 0x00000000L, 0x40080000L, 0x42000000L,
        0x00080100L, 0x02000100L, 0x40000100L, 0x00080000L,
        0x00000000L, 0x40080000L, 0x02080100L, 0x40000100L };

static unsigned long SP6[64] = {
        0x20000010L, 0x20400000L, 0x00004000L, 0x20404010L,
        0x20400000L, 0x00000010L, 0x20404010L, 0x00400000L,
        0x20004000L, 0x00404010L, 0x00400000L, 0x20000010L,
        0x00400010L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x00000000L, 0x00400010L, 0x20004010L, 0x00004000L,
        0x00404000L, 0x20004010L, 0x00000010L, 0x20400010L,
        0x20400010L, 0x00000000L, 0x00404010L, 0x20404000L,
        0x00004010L, 0x00404000L, 0x20404000L, 0x20000000L,
        0x20004000L, 0x00000010L, 0x20400010L, 0x00404000L,
        0x20404010L, 0x00400000L, 0x00004010L, 0x20000010L,
        0x00400000L, 0x20004000L, 0x20000000L, 0x00004010L,
        0x20000010L, 0x20404010L, 0x00404000L, 0x20400000L,
        0x00404010L, 0x20404000L, 0x00000000L, 0x20400010L,
        0x00000010L, 0x00004000L, 0x20400000L, 0x00404010L,
        0x00004000L, 0x00400010L, 0x20004010L, 0x00000000L,
        0x20404000L, 0x20000000L, 0x00400010L, 0x20004010L };

static unsigned long SP7[64] = {
        0x00200000L, 0x04200002L, 0x04000802L, 0x00000000L,
        0x00000800L, 0x04000802L, 0x00200802L, 0x04200800L,
        0x04200802L, 0x00200000L, 0x00000000L, 0x04000002L,
        0x00000002L, 0x04000000L, 0x04200002L, 0x00000802L,
```

```
        0x04000800L, 0x00200802L, 0x00200002L, 0x04000800L,
        0x04000002L, 0x04200000L, 0x04200800L, 0x00200002L,
        0x04200000L, 0x00000800L, 0x00000802L, 0x04200802L,
        0x00200800L, 0x00000002L, 0x04000000L, 0x00200800L,
        0x04000000L, 0x00200800L, 0x00200000L, 0x04000802L,
        0x04000802L, 0x04200002L, 0x04200002L, 0x00000002L,
        0x00200002L, 0x04000000L, 0x04000800L, 0x00200000L,
        0x04200800L, 0x00000802L, 0x00200802L, 0x04200800L,
        0x00000802L, 0x04000002L, 0x04200802L, 0x04200000L,
        0x00200800L, 0x00000000L, 0x00000002L, 0x04200802L,
        0x00000000L, 0x00200802L, 0x04200000L, 0x00000800L,
        0x04000002L, 0x04000800L, 0x00000800L, 0x00200002L };

static unsigned long SP8[64] = {
        0x10001040L, 0x00001000L, 0x00040000L, 0x10041040L,
        0x10000000L, 0x10001040L, 0x00000040L, 0x10000000L,
        0x00040040L, 0x10040000L, 0x10041040L, 0x00041000L,
        0x10041000L, 0x00041040L, 0x00001000L, 0x00000040L,
        0x10040000L, 0x10000040L, 0x10001000L, 0x00001040L,
        0x00041000L, 0x00040040L, 0x10040040L, 0x10041000L,
        0x00001040L, 0x00000000L, 0x00000000L, 0x10040040L,
        0x10000040L, 0x10001000L, 0x00041040L, 0x00040000L,
        0x00041040L, 0x00040000L, 0x10041000L, 0x00001000L,
        0x00000040L, 0x10040040L, 0x00001000L, 0x00041040L,
        0x10001000L, 0x00000040L, 0x10000040L, 0x10040000L,
        0x10040040L, 0x10000000L, 0x00040000L, 0x10001040L,
        0x00000000L, 0x10041040L, 0x00040040L, 0x10000040L,
        0x10040000L, 0x10001000L, 0x10001040L, 0x00000000L,
        0x10041040L, 0x00041000L, 0x00041000L, 0x00001040L,
        0x00001040L, 0x00040040L, 0x10000000L, 0x10041000L };

static void desfunc(block, keys)
register unsigned long *block, *keys;
{
        register unsigned long fval, work, right, leftt;
        register int round;

        leftt = block[0];
        right = block[1];
        work = ((leftt >> 4) ^ right) & 0x0f0f0f0fL;
        right ^= work;
        leftt ^= (work << 4);
        work = ((leftt >> 16) ^ right) & 0x0000ffffL;
        right ^= work;
        leftt ^= (work << 16);
        work = ((right >> 2) ^ leftt) & 0x33333333L;
        leftt ^= work;
        right ^= (work << 2);
        work = ((right >> 8) ^ leftt) & 0x00ff00ffL;
        leftt ^= work;
        right ^= (work << 8);
        right = ((right << 1) | ((right >> 31) & 1L)) & 0xffffffffL;
        work = (leftt ^ right) & 0xaaaaaaaaL;
        leftt ^= work;
        right ^= work;
        leftt = ((leftt << 1) | ((leftt >> 31) & 1L)) & 0xffffffffL;
```

```
        for( round = 0; round < 8; round++ ) {
                work = (right << 28) | (right >> 4);
                work ^= *keys++;
                fval = SP7[ work                & 0x3fL];
                fval |= SP5[(work >> 8) & 0x3fL];
                fval |= SP3[(work >> 16) & 0x3fL];
                fval |= SP1[(work >> 24) & 0x3fL];
                work = right ^ *keys++;
                fval |= SP8[ work               & 0x3fL];
                fval |= SP6[(work >> 8) & 0x3fL];
                fval |= SP4[(work >> 16) & 0x3fL];
                fval |= SP2[(work >> 24) & 0x3fL];
                leftt ^= fval;
                work = (leftt << 28) | (leftt >> 4);
                work ^= *keys++;
                fval = SP7[ work                & 0x3fL];
                fval |= SP5[(work >> 8) & 0x3fL];
                fval |= SP3[(work >> 16) & 0x3fL];
                fval |= SP1[(work >> 24) & 0x3fL];
                work = leftt ^ *keys++;
                fval |= SP8[ work               & 0x3fL];
                fval |= SP6[(work >> 8) & 0x3fL];
                fval |= SP4[(work >> 16) & 0x3fL];
                fval |= SP2[(work >> 24) & 0x3fL];
                right ^= fval;
                }

        right = (right << 31) | (right >> 1);
        work = (leftt ^ right) & 0xaaaaaaaaL;
        leftt ^= work;
        right ^= work;
        leftt = (leftt << 31) | (leftt >> 1);
        work = ((leftt >> 8) ^ right) & 0x00ff00ffL;
        right ^= work;
        leftt ^= (work << 8);
        work = ((leftt >> 2) ^ right) & 0x33333333L;
        right ^= work;
        leftt ^= (work << 2);
        work = ((right >> 16) ^ leftt) & 0x0000ffffL;
        leftt ^= work;
        right ^= (work << 16);
        work = ((right >> 4) ^ leftt) & 0x0f0f0f0fL;
        leftt ^= work;
        right ^= (work << 4);
        *block++ = right;
        *block = leftt;
        return;
        }

void des2key(hexkey, mode)              /* stomps on Kn3 too */
unsigned char *hexkey;                  /* unsigned char[16] */
short mode;
{
        short revmod;
```

```
        revmod = (mode == EN0) ? DE1: EN0;
        deskey(&hexkey[8], revmod);
        cpkey(KnR);
        deskey(hexkey, mode);
        cpkey(Kn3);                                     /* Kn3 = KnL */
        return;
        }

void Ddes(from, into)
unsigned char *from, *into;             /* unsigned char[8] */
{
        unsigned long work[2];

        scrunch(from, work);
        desfunc(work, KnL);
        desfunc(work, KnR);
        desfunc(work, Kn3);
        unscrun(work, into);
        return;
        }

bool TFCA_3DES(bool bEnspot, unsigned char* pbyKey,int nLength, unsigned char* pbySource,
unsigned char* pbyTarget)
{
        if(nLength==0 || nLength%8)
                return false;
        int section = nLength/8;
        des2key(pbyKey,bEnspot?0:1);
        for(int i = 0 ;i<section;i++)
                Ddes(pbySource+8*i,pbyTarget+8*i);
        return true;
}

bool TFCA_DES(bool bEnspot, unsigned char* pbyKey,int nLength, unsigned char* pbySource,
unsigned char* pbyTarget)
{
        if(nLength==0 || nLength%8)
                return false;
        int section = nLength/8;
        deskey(pbyKey,bEnspot?0:1);
        for(int i = 0 ;i<section;i++)
                des(pbySource+8*i,pbyTarget+8*i);;
        return true;
}

/* Validation sets:
 *
 * Single-length key, single-length plaintext -
 * Key    : 0123 4567 89ab cdef
 * Plain: 0123 4567 89ab cde7
 * Cipher: c957 4425 6a5e d31d
 *
 * Double-length key, single-length plaintext -
 * Key    : 0123 4567 89ab cdef fedc ba98 7654 3210
 * Plain: 0123 4567 89ab cde7
 * Cipher: 7f1d 0a77 826b 8aff
```

```
* d3des V5.0a rwo 9208.07 18:44 Graven Imagery
**********************************************************************/
```