GemClub
Reference Manual

Version 1.0

GEMPLUS

December 1997

At press time, this document is as thorough and correct as possible. The information contained herein may, however, have been updated after this date. GEMPLUS reserves the right to change the functions and specifications of its products at any time without prior notice.

This document was prepared by GEMPLUS for both its clients and for its own internal use. The information contained herein is the sole property of GEMPLUS and shall not under any circumstances be reproduced without prior consent of the company.

Smart Cards and Smart Card readers are patent protected by INNOVATRON and Bull CP8 and are produced by GEMPLUS under license.

Printed in France.

GEMPLUS, B.P. 100, 13881 GEMENOS CEDEX, FRANCE.
Tel: +33 (0)4.42.36.50.00 Fax: +33 (0)4.42.36.50.90

Document Reference: E5217803/DPD05864A00

ABOUT THIS MANUAL

This manual provides reference information about GemClub cards. It provides detailed descriptions of:

- The card memory structure and functions
- How the card protects data
- How to protect and verify transmissions with the card
- How to send low-level commands to the card

It is aimed at people who want:

- To understand how the operating system works
- To access reference information about the operating system
- To develop applications for GemClub cards

Audience

This manual assumes that you are familiar with the following subjects:

- Cryptography
- Smart card technology

You should also have access to the following documentation:

- ISO/IEC 7816-3—Identification Cards, Part 3 : Electronic signals and Transmission Protocols
- ISO/IEC 7816-4—Identification Cards, Part 4: Inter-Industry Commands for Interchange
- EMV—IC Card Specifications for Payment Systems, parts 1,2,3

How to Use This Manual

You should read the chapters up to and including *Gemclub Command Format* in this Reference Manual completely before you start to work with GemClub, and then refer to the rest of the Reference Manual as required to carry out your responsibilities.

Overview

Describes GemClub cards, their data structure, security system, and commands.

GemClub Files

Describes GemClub file types and structures.

GemClub Initial Status

Provides details of the card initialization process in GemClub.

GemClub Security Architecture

Describes how GemClub cards protect data stored in files using secret codes and secret keys.

GemClub Encryption Algorithms Describes the GemClub cryptographic security features and Message Authentication Codes (MACs) specified for commands used in GemClub.

Other GemClub Features Describes the Backup and Point Conversion mechanisms that have been created for use in GemClub cards.

GemClub Command Format Provides details of the APDU message structure used by GemClub

GemClub Commands Describes the GemClub command set, their codes and how to use them.

Appendix A: GemClub Exchange Structure Describes the various exchange features and protocols used in GemClub to facilitate the transfer of data between GemClub cards and Card Readers.

Appendix B: Answer-to-Reset Provides details on both the Answer-to-Reset (ATR) procedure.

Appendix C: Customer Card Shipment Process Provides details on the procedures involved in customer card shipment.

Glossary Gives definitions of the GemClub–specific terms.

Abbreviations And Acronyms Gives the definitions of the Abbreviations and Acronyms used in this document.

Notation

The following notation conventions are used throughout this document.

By default, a numeric value is expressed in decimal notation.

Whenever a value is expressed in binary, it will be followed by the b character. For example the decimal value 13 expressed in binary becomes **1101b**.

An hexadecimal number is followed by the h character. For example the decimal value 13 expressed in hexadecimal becomes **0Dh**.

The value 00h is assigned to each 'RFU (Reserved for Future Use)' byte.

Bit numbering

A **byte B** consists of 8 bits $b_7b_6b_5b_4b_3b_2b_1b_0$: b_7 is the most significant bit and b_0 the least significant bit:

One Byte	b7	b6	b5	b4	b3	b2	b1	b0

A **string of bytes** consists of n concatenated bytes $B_nB_{n-1}....B_2B_1$: B_n is the most significant byte and B_1 the least significant byte :

A string of n bytes	Bn	Bn-1	Bn-2	—	B5	B4	B3	B2	B1

RFU Reserved for future use. When RFU bytes are included in a command, the value 0 (or 00h) should be used.

3DES This notation is used to indicate that 3DES algorithm is being used.

CONTENTS

OVERVIEW

Range Presentation

GemClub is an easy-to-use microprocessor card, with advanced loyalty functions, which has been developed for loyalty card applications. GemClub cards can also be used in many other types of application, such as coupon programs, stored value cards, private electronic purses, metering (e.g., measuring consumption of gas or electricity), customer identification etc.

GemClub consists of:

- **GemClub 1K** cards which consist of 1kilobyte of EEPROM and
- **GemClub-EMV** cards—cards which are EMV compatible.

Data Structure

GemClub has a flat file structure. All data objects needed for the loyalty application are stored at the same level in the memory.

- **System file** Used for the global security management of the card.
- **Loyalty files** Used for the storage of points and rules.
- **Record files** —Used for data storage.
- **Security files** —Used for the storage of **Secret Codes** and **Secret Keys**.

Command Set

The GemClub operating system includes the following:

- *Administrative* commands—**Create Object, Delete Object** and **Select Communication Speed.**
- *Application* commands—For example, **Award**, **Redeem**, **Use Rule, Update Parameter** and **Read Parameter**.

Data Access Management & Security

Access conditions are used to protect data files or data elements. These conditions also define the level of protection granted to a file. In general, individual files stored in GemClub cards are **Secret Key** or **Secret Code** protected.

Specific access conditions can be defined for each data file or data element and for each command (or group of commands).

A single access condition is defined on one byte.

The secret codes are 8 bytes long and are stored in secret code files (only one secret code per file).

In addition to the main access conditions (secret code or secret key), PIN (Personal Identification Number) protection may be required. A PIN is a secret code, unique to the card holder, whose file is specified in the system file.

Secure messaging, (based on secret keys), is used to ensure authenticity and integrity of data exchanged between the terminal and the card. It uses the triple DES algorithm. It is also used in cross-authentication, that is the terminal and card can authenticate each other.

A Message Authentication Code (MAC_IN) is sent by the terminal with the command to be protected. The card will verify this MAC_IN before performing the command (the right verification ensures that the terminal has the rights to perform this action and ensures data transfer integrity). After performing the command, the card computes a MAC_OUT which will be verified by the terminal (ensuring the authenticity of the card and data transfer integrity).

For security, when updating the secret key, the new value can be ciphered before being transmitted to the card.

A transaction proof can be computed by the card after an application function. It is based on secret keys and uses the 3DES algorithm. This transaction proof is verified by the operator to validate the transaction (non repudiation).

Communication

GemClub cards send and receive data under the **T=0** communication protocol in accordance with the **ISO 7816-3** standard.

GemClub also offers high speed communication—at either 9,600 baud or 115,200 baud.

GEMCLUB FILES

Overview

This chapter discusses GemClub file structures. GemClub cards have a flat file structure. This means that all the files in GemClub are at the same hierarchy level as follows:

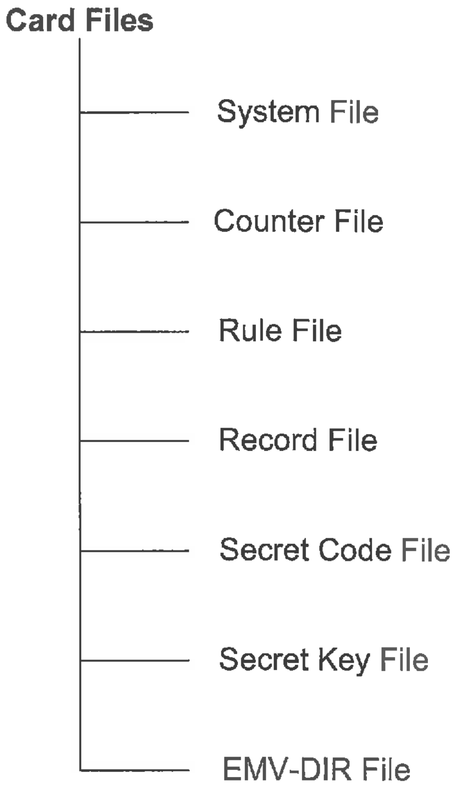


Figure 1 File Structure

GemClub File Organization

GemClub supports the following types of files:

- **System file** There is only one system file per card. The system file is used for global security management of the card.
- **Loyalty files**—Used for the storage of points, customer profiles and other information in the **Counter files** and rules in the **Rule files.**
- **Security files—Secret Key files** and **Secret Code files.**
- **Record files**—Used to store data. Record files also conform with **ISO/IEC 7816-4** record management standards. The files are named and accessed in accordance with these standards.

You must use the following file selection methods:

- **File type**—Identifies the purpose of the data file. See '*GemClub File Types*'.

  and

- **Short File Identifier** (SFI)—A number on five bits between **01h–1Eh** which is allocated when the file is created.

  The value zero is not managed by the GemClub operating system and will thus be rejected.

GemClub File Types

The following file types are supported by the GemClub operating system.

File Type	Data Object Type	Description
System file	01h	Data elements internally managed by the OS.
Record file	02h	Records managed by the terminal.
Counter	03h	Stores points that have been awarded to the cardholder.
Rule	04h	Macro-instructions to be run on counters.
Secret code	05h	Secret code.
Secret key	06h	Secret value used by cryptography algorithms.

Note: Only one system file can be created for each card.

*The SFI for a system file is always **01h**.*

No two files of the same type can have the same SFI. No more than 16 Rule files can be created on a GemClub card. For all other file types, with the exception of the System file, no more than 30 files can be created.

Personalization Data

This information is stored in files and is used for the files' own management. It may consist of:

- Access conditions
- PIN code and EMV directory file references
- A key reference, for transaction proof
- A rule version
- Ratification counters for the maximum number of allowed attempts and number of attempts remaining for the verification of secret codes and secret keys

This information must be updated after the file is created (using the **Update Parameter** command - see *GemClub Commands*). When using this command, you specify the data to be updated by using a specific tag. However, in order to simplify this task, there is a tag value (**the personalization tag**) which you can use to update all the personalization data at the same time.

For the file descriptions in this chapter, the personalization data elements are shown in gray in the tables of tag values.

The System File

The **System** file can store up to 12 bytes of information, including a card transaction counter (CTC), internal operating system data and personalization data.

File Descriptor

Although the system file is predefined on all GemClub cards, you must specify the properties of each system file for each GemClub card to operate correctly.

Note: When a system file is first created, there are no other files on the card. These files can be created at a later stage.

System file ID, type and size are as follows:

File ID	01h
File Type (Data object type)	System (01h)
File Size	12 bytes

The system file contains the following details:

Tag	Size	Contains	Details
21h	1 byte	**Access condition for Update/Delete**	Specifies right to **Update** or **Delete** the system file.
22h	1 byte	**Access condition for Read**	Specifies right to **Read** the system file.
23h	1 byte	**Access condition for Create**	Specifies right to **Create** a file.
24h	1 byte	**PIN Code File Reference information**	Identifier of the secret code file used as PIN for cardholder identification.
26h	1 byte	**EMV-DIR File Reference information**	Identifier of the file used for EMV-DIR simulation.
27h	2 bytes	**Card Transaction Counter**	Used for authentication operations. The CTC is used for secure messaging computation.

Note: A tag is a unique number which is one byte long. This tag is used to define the data to be used but is not stored in the card itself.

The personalization tag is 20h. This enables you to read or update the file details corresponding to tags 21h, 22h, 23h, 24h and 26h simultaneously.

For more details on Access Conditions coding , see Rules for Defining Access Conditions .

Loyalty Files

Loyalty files consist of **Counter files** and **Rule files** as follows:

Counter Files

A **Counter** file is a set of data elements used for points '**Award**' and/or '**Redeem**' schemes. A Counter file can be accessed through Rule files.

Up to 30 different counter files can be registered on a single GemClub card.

Counter File Body Structure

GemClub **counters** vary in structure depending on the file attributes you specify for them.

You can specify the type of counter structure you require—bytes stored in the counter structure—using the Structure byte field, as follows:

Bit 0	Cumulative Balance
Bit 1	Visit Counter
Bit 2	Validity Date—Award
Bit 3	Validity Date—Redeem
Bit 4	Rules Specified for this Counter
Bit 5	Label
Bit 6	Reserved for Future Use
Bit 7	Reserved for Future Use

Note: *Set the bit to 1 to ensure that the relevant data object type is included in the counter structure. Set it to 0 to ensure that the relevant data object type is not included in the counter structure.*

Counter file ID, type and size are as follows:

File ID	01h–1Eh.
File Type (Data Object Type)	Counter (03h).
File Size	Counter files can vary in size from 16 bytes to 40 bytes—according to the properties you specify for them.

A counter file must contain at least the following details:

Tag	Size	Contains	Details
61h	1 byte	**Access condition for Update/Delete**	Specifies right to **Update** or **Delete** the counter file.
62h	1 byte	**Access condition for Read**	Specifies right to **Read** the counter file
63h	1 byte	**Key reference for Transaction Proof**	SFI of the key file that is used for transaction proof computation. It ensures that a transaction proof is requested.
64h	1 byte	**Access condition for Award**	Specifies right to make a points **Award**.
65h	1 byte	**Access condition for Redeem**	Specifies right to **Redeem** points.
70h	3 bytes	**Balance**	Shows the number of points currently stored on the counter.

The personalization tag is 60h. This enables you to read or update the file details corresponding to tags 61h, 62h, 63h, 64h and 65h simultaneously.

Optional features that you can add to each counter that you create on your GemClub card include the following:

Note: *These optional features are defined in accordance with the counter file body structure.*

Tag	Size	Contains	Details
71h	3 bytes	**Cumulative Balance**	Displays the number of points awarded to the counter since its creation.
72h	2 bytes	**Visit Counter**	Displays the number of award operations carried out on the counter since its creation.
73h	6 bytes	**Award Validity Date**	Specifies validity dates for an **Award** operation—For example, if you are running a special sales promotion you may wish to give points during a limited period.
74h	6 bytes	**Redeem Validity Date**	Specifies validity dates for a **Redeem** operation—For example, if you are running a special sales promotion you may wish to offer a special redemption period.
75h	2 bytes	**Rules allowed for this counter**	Specifies rules that can be run on this counter. If this field is absent, the '**Use Rule**' operation cannot modify the details on the counter. Otherwise, each bit corresponds to a rule identifier. For more information see: '*Use Rule Command*'.
76h	8 bytes	**Label**	Displays an alphanumeric value which can be used to identify the name and the version of the counter. This can be reused during the lifetime of the card.

Validity Dates

An Award/Redeem operation is rejected if the current date does not fall within the range specified by the relevant validity dates.

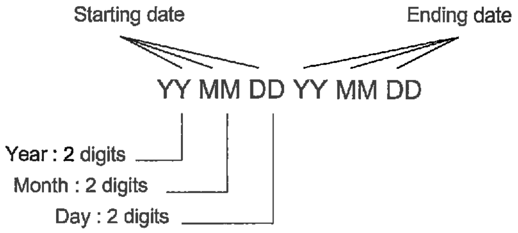
The date coding method must be written with a view to ensuring that the following comparison can be performed by the card:

Start date ≤ Current date ≤ End date

Note: *The operating system does not perform any format control on the date values.*

Example of how GemClub deals with year 2000 date coding problem.

Dates are stored in a BCD format—as follows:



Note: *To avoid the possibility of the year 2000 being interpreted as the year 1900 (because BCD uses the last two digits of the year), the terminal converts year dates before sending them to the card as follows:*

Year	***Converted to***
1997	*00*
1998	*01*
1999	*02*
2000	*03*
2001	*04*

Rule Files

A **Rule file** is a set of macro instructions that can be run during a '**Use rule**' operation. You can use rules to link up to four different counters. Up to 16 rule files can be stored in a single GemClub card.

The structure of a **Rule** file depends on the features and properties that you specify for the file.

Rules file ID, type and size are as follows:

File ID	01h–10h
File Type (Data Object Type)	Rule (04h)
File Size	*(12+8n)* bytes (Including O/S information)—where n = the number of macro instructions.

A **Rule** file must contain the following details:

Tag	Size	Contains	Details
81h	1 byte	**Access condition for Update/Delete**	Specifies right to **Update** or **Delete** the Rule file.
82h	1 byte	**Access condition for Read**	Specifies right to **Read** the Rule file.
83h	1 byte	**Key reference for Transaction Proof**	SFI of the key file that is used for transaction proof computation. It ensures that a transaction proof is requested.
84h	1 byte	**Access condition for Use Rule**	Specifies right to run the rule by performing a **Use Rule** command.
85h	1 byte	**Version**	A byte used to show the version of the rule.
90h	8 bytes	**Macro Instruction 1**	Specifies, and describes, the action that is to be carried out on the specified counter. *Note*: At least one macro must be specified for each rule you create for the **counter.** For more information on macro instructions see '*GemClub Rule Mechanism Point Conversion*'.

The personalization tag is 80h. This enables you to read or update the file details corresponding to tags 81h, 82h, 83h, 84h and 85h simultaneously.

You can also add a range of other macro instructions that you create, as follows:

Tag	Size	Contains	Details
91h	8 bytes	**Macro Instruction 2**	Specifies, and describes, the instruction to be used on this counter.
—			
93h	8 bytes	**Macro Instruction 4**	Specifies, and describes, the instruction to be used on this counter.

A maximum of four macro instructions can be stored in a rule file.

About Macro Instructions

A Macro Instruction is a logical description of an operation—either an **Award** or a **Redeem** that is specified for a given counter. A macro instruction is made up of the following information:

Field	Size	What it is
Function	1 byte	Operation to be applied to the specified counter. It consists of: **00h**—Bypass the macro instruction. **01h**—Specifies that the instruction is for an **Award** operation. **02h**—Specifies that the instruction is for a **Redeem** operation. Other values are reserved for future use (RFU).
Counter Reference	1 byte	Short File Identifier of the counter on which the macro instruction is to be run.
Parameter A	3 bytes	First conversion parameter. This parameter represents the number of points that are to be added to or subtracted from the counter each time that the value specified in Parameter B is attained. See '*GemClub Rule Mechanism Point Conversion*'.
Parameter B	3 bytes	Second conversion parameter. This parameter is used to specify the value that must be attained prior to points being added to or subtracted from the counter that has been specified for the card. See '*GemClub Rule Mechanism Point Conversion*'.

Note: *A macro instruction can only be run if the rule is allowed in the Rules Allowed field for the corresponding counter.*

Security Files

Security files consist of **Secret Code** files and **Secret Key** files as follows:

Secret Code Files

Secret codes are used for file protection. A secret code file may only contain one secret code.

You can also use a **PIN** which is a secret code whose file is referenced in the System file.

To protect a file using a secret code or a PIN, the code or PIN must be referenced, in that file's access condition.

Secret code file ID, type and size are as follows:

File ID	01h–1Eh
File Type (Data Object Type)	Secret Code (05h)
File Size	(Including O/S information) 16 bytes

A **Secret Code** file must contain the following details:

Tag	Size	Contains	Details
A1h	1 byte	**Access condition for Update/Delete**	Specifies right to Update/Delete the secret code file.
A3h	1 byte	**Maximum Attempts Allowed/ Attempts Remaining**	Specifies the number of attempts allowed and the number of attempts remaining to log on before the card is rejected and the secret code is blocked.
B0h	8 bytes	**Secret Code Value**	Specifies the value assigned to the secret code.

The personalization tag is A0h. This enables you to update the file details corresponding to tags A1h and A3h simultaneously.

Note: *Up to 15 attempts can be specified in the file for the Maximum Attempts Allowed parameter.*

Up to 15 attempts can be specified in the file for the Attempts Remaining parameter. This value is reset to Maximum Attempts Allowed following a correct presentation of the secret code.

No file details can be read.

Secret Key Files

Secret Keys are used in GemClub to perform secure messaging and compute transaction proofs (see *GemClub Security Architecture* chapter).

To use a secret key for secure messaging, its SFI must be referenced in the access condition of the file that is to be protected. The access condition must also specify that secure messaging is to be used (see *Rules for Defining Access Conditions* in the *GemClub Security Architecture* chapter).

To use a secret key for transaction proof computation, its SFI must be referenced in the specified rule file or counter file (see *Counter Files* and *Rule Files* in the *GemClub Files* chapter).

A **Secret Key** file may only contain one secret key.

Secret Key file ID, type and size are as follows:

File ID	01h–1Eh
File Type (Data Object Type)	Secret Key (06h)
File Size	(Including O/S information) 24 bytes

Secret Key files must contain the following details:

Tag	Size	Contains	Details
C1h	1 byte	**Access condition for Update/Delete**	Specifies right to **Update/Delete** the secret key file.
C3h	1 byte	**Maximum Attempts Allowed/ Attempts Remaining**	Specifies the number of attempts allowed and the number of attempts remaining to log on before the card is rejected and the secret key is blocked.
D0h	8 bytes	**Secret Key Value**	Specifies half of the value allocated to the **3DES** key which is used to conform with **3DES** encryption standards. The two halves of the Secret Key value are accessed separately during '**Update**' operations.
D1h	8 bytes	**Secret Key Value**	Specifies half of the value allocated to the **3DES** key which is used to conform with **3DES** encryption standards. The two halves of the Secret Key value are accessed separately during '**Update**' operations.

The personalization tag is C0h. This enables you to update the file details corresponding to tags C1h and C3h simultaneously.

Note: *Up to 15 attempts can be specified in the file for the Maximum Attempts Allowed parameter.*

Up to 15 attempts can be specified in the file for the Attempts Remaining parameter. This value is reset to Maximum Attempts Allowed following a correct verification of the secret key.

No file details can be read.

Record Files

A **Record** file is used to store data in the GemClub card. GemClub supports linear files with variable sized records.

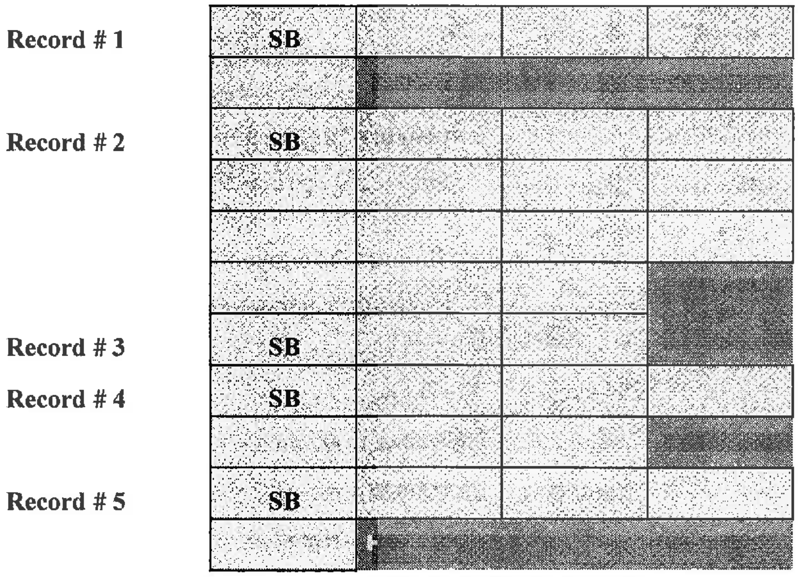


Figure 2 Linear File with Variable Sized Records

where:

SB	is the system byte
░	is space taken up by the record
▓	is bytes which must be added to round up to the next multiple of 4

Linear files with variable sized records—Records of varying lengths. The first record in this file is defined as record number 1. No more than 254 records can be stored in a linear variable record file. The maximum size of a record is 255 bytes (in the case of secure messaging protection, the maximum size of a record is 24 bytes).

The structure of a **Record** file depends on the features and properties that you specify for the file.

Record file ID, type and size are as follows:

File ID	01h–1Eh
File Type (Data Object Type)	Record (02h)
File Size	Variable (see below)

Record files must contain the following details:

Tag	Size	Contains	Details
41h	1 byte	**Access condition for Update/Delete**	Specifies right to **Update** or **Delete** the record file.
42h	1 byte	**Access condition for Read**	Specifies right to **Read** the record file.
	L_1	**Record 1**	Data.
	L_2	**Record 2**	Data.
	L_n	**Record N**	Data.

The personalization tag is 40h. This enables you to read or update the file details corresponding to tags 41h and 42h simultaneously.

Length of a Record File

This is equal to the length of the file descriptor and access conditions (8 bytes) plus the length of each record.

Length of a record in a record file:

Firstly, add one byte for the system byte to the length of the usable part of the record. Then round up this number to the next multiple of 4 bytes.

Thus, for the example in figure 2, the length of the file is:

8 bytes (file descriptor)

+ 8 bytes (record 1 = 1 byte for SB + 4 usable bytes + 3 rounding bytes).

+ 16 bytes (record 2 = 1 byte for SB + 14 usable bytes + 1 rounding byte).

+ 4 bytes (record 3 = 1 byte for SB + 2 usable bytes + 1 rounding byte).

+ 8 bytes (record 4 = 1 byte for SB + 6 usable bytes + 1 rounding byte).

+ 8 bytes (record 5 = 1 byte for SB + 4 usable bytes + 3 rounding bytes).

= 52 bytes total.

Referencing by Name EMV Simulation

EMV features are only usable with GemClub-EMV cards. A simulation of the '**Select application file by name**' command, contained in part 3 of the EMV card specifications, has been created to ensure that GemClub-EMV cards are compatible with this standard.

GemClub supports directory selection using names that are in accordance with **ISO/IEC 7816-4** standards. Record File names accepted by GemClub must be stored in the EMV-DIR file.

Note: An application file name can be right truncated for file selection.

The Payment System Environment (PSE) can also be right truncated for file selection.

The scanning algorithm always searches through the records of the EMV-DIR file stored on the card.

The scanning algorithm always stops at the first occurrence which matches.

EMV-DIR Files

GemClub also supports files that are structured in accordance with EMV standards. An example is provided below of how to write an EMV-DIR compatible record file.

An **EMV-DIR** file is a record file that is referenced in the **System** file and used for EMV simulation features. The first record must be in accordance with the EMV structure for PSE **‘1PAY.SYS.DDF01** (the tag to be used is ‘9Dh’).

Each record in the EMV-DIR file is structured using a simple **Tag Length Value** (TLV) in accordance with EMV card specification standards. The beginning of the mandatory fields in an EMV-DIR file is as follows:

	Tag	Size	Contains	Details
Record 1	70h	1 byte	**Data Template.**	
= DDF	variable	1 byte	**Length.**	Data template.
	61h	1 byte	**Application template.**	
	variable	1 byte	**Length.**	Application template.
	9Dh	1 byte	**Tag.**	Directory Definition File (DDF) (Tag = 9Dh) name
	5-16	1 byte	**Length.**	Application Identifier.
	—	5-16 bytes	**Application Identifier.**	The name of the application that is recognized by the record file.
	50h	1 byte	**Tag.**	Application Label.
	1-16	1 byte	**Length.**	Application Label.
	—	1-16 bytes	**Application Label.**	Alphanumeric value.
Record 2	70h	1 byte	**Data Template.**	
= ADF	variable	1 byte	**Length.**	Data template.
	61h	1 byte	**Application template.**	
	variable	1 byte	**Length.**	Application template.
	4Fh	1 byte	**Tag.**	Application Definition File (ADF) (Tag = 4Fh)
	5-16	1 byte	**Length.**	Application Identifier.
	—	5-16 bytes	**Application Identifier.**	The name of the application that is recognized by the record file.
	50h	1 byte	**Tag.**	Application Label.
	1-16	1 byte	**Length.**	Application Label.
	—	1-16 bytes	**Application Label.**	Alphanumeric value.
Record 3 = ADF	...	...	...	

GEMCLUB INITIAL STATUS

Initialization Process

During the manufacturing process, the microprocessor cards must be initialized. Card initialization involves writing basic information onto the card such as:

- Card Serial Number and Issuer Reference
- System file Structure
- Protection keys
- Filter creation—this only applies to GemClub-EMV cards

There are two types of card initialization processes:

- Customer card process—for GemClub card production issues only. The cards are protected by diversified keys. For more information, see '*Appendix C: Customer Card Shipment Process*'.
- Sample Card Process—The cards are protected by a common key. This is used for the sample cards that are delivered with each kit.

Initial File Structure

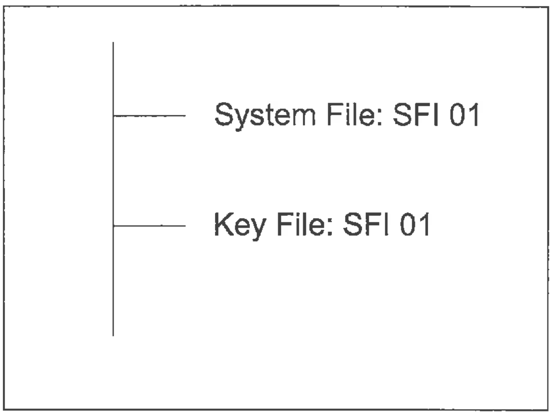


Figure 3 Initial File Structure

System File

Global access conditions for the card are defined in system file.

File Parameters are as follows:

Action Access Conditions and Parameters	**Sample Card and Customer Card Processes**
Update/Delete	Key 01
Read	Free
Create	Key 01
Pin Code File Reference	00h
EMV DIR file Reference	00h
Card Transaction Counter	0005h

Key File SFI: 01h

This file contains key 01. This key is needed to carry out most of the personalization operations by the card application issuer. Its value depends on the card's initial status:

- GemClub cards initialized with the sample process, the value of this key is the same for all the cards : ASCII string **TESTKEY1TESTKEY2.**
- GemClub application cards initialized with the customer card process, **the key is diversified** by Gemplus using the Mother Batch Key (derived from a Customer Key that is stored inside a Customer Card). The Customer Card is delivered to the card issuer. For more information, see '*Appendix C: Customer Card Shipment Process*'.

File parameters are as follows:

Identifier	**Body size**
01h	24 bytes

Security Access Conditions are as follows:

Action	**Access Condition**
Update / Delete	Secure Messaging with Key 01

The Maximum Attempts Allowed and Attempts Remaining counters are set to 3.

Card Serial Number

The Card Serial Number is unique for each GemClub card and it cannot be modified. It is coded on 8 bytes (2 words of 32 bits) as follows:

<table>
<tr><th colspan="32">Bits</th></tr>
<tr><td>31</td><td>30</td><td>29</td><td>28</td><td>27</td><td>26</td><td>25</td><td>24</td><td>23</td><td>22</td><td>21</td><td>20</td><td>19</td><td>18</td><td>17</td><td>16</td><td>15</td><td>14</td><td>13</td><td>12</td><td>11</td><td>10</td><td>09</td><td>08</td><td>07</td><td>06</td><td>05</td><td>04</td><td>03</td><td>02</td><td>01</td><td>00</td></tr>
<tr><td colspan="10">[1 10 bits]</td><td colspan="22">[2 22 bits]</td></tr>
<tr><td colspan="10">[3 10 bits]</td><td colspan="6">[4 6 bits]</td><td colspan="8">[5 8 bits]</td><td colspan="8">[6 8 bits]</td></tr>
</table>

1:	**Batch number**	(10 bits)
2:	Serial number (part)	(22 bits)
3:	Serial number (part)	(10 bits)
4:	**Week number**	(06 bits)
5:	Serial number (part)	(08 bits)
6:	**Year of manufacture**	(08 bits)

Fields 2, 3 and 5 make up the serial number in a batch.

Note: This is a read-only parameter.

Issuer Reference Number

The Issuer Reference Number is coded over 4 bytes (3 bytes plus a checksum). This number is provided by Gemplus as a unique reference for each customer and cannot be updated. We recommend that you use the issuer reference number to identify your card.

In sample cards, the Issuer Reference Number is set to 00 FF 00 10h.

Note: This is a read-only parameter.

IO Buffer Size

It is possible to update a record of up to 255 bytes without secure messaging.

***Warning**: This value may be less than 255 bytes due to the limitation of the reader or terminal you are using.*

When secure messaging is used, data is limited to 24 bytes.

EEPROM Size

The following EEPROM size is available to the user after the card has been initialized:

Product	EEPROM available prior to creation of the system and key files	EEPROM available after the system and key files have been created
GemClub 1K	952 bytes	916 bytes
GemClub EMV	748 bytes	712 bytes

GEMCLUB SECURITY ARCHITECTURE

Overview

GemClub security features can be divided into five separate sections as follows:

- **Security Statuses.**
- **Security Attributes**—Security specifications that authorize actions which can be performed on files stored on a card.
- **Security Mechanisms**—Procedures used to ensure the security of files stored on cards. These security mechanisms are also used to ensure that data communications between GemClub cards and terminals are not corrupted.
- **Transaction Proofs**—Procedure used to prove the authenticity of transactions to the operator.
- **Access Condition Check Algorithm** Used to find whether a user or merchant has the right to run any actions on the card

Security Statuses

During a card session or during the performing of a command, it may be necessary to register that a code presentation or secure messaging has been successfully completed. These registers are called security statuses. There are two types of security statuses:

- Command–specific security status
- Global Security Status

Command–specific security status:

This registers key verifications. It also registers secret code verifications when they are included in the data field of the command. This status is stored in RAM and only lasts during the execution of the command. These verifications have no impact on the global security status.

Global Security Status:

This register is used when the secret code has to be presented - using the Verify Secret Code command - before performing the main command.

This is necessary for PIN verification and for the ISO 7816-4 commands (where the secret code is not transmitted in the command field). The global security status can only record proof of the PIN code and of the last secret code that was presented. A secret code which was presented without using the Verify Secret Code command is not registered. This status is stored in RAM and is erased at the end of the card session.

Security Attributes

Security attributes are predefined security requirements which users of GemClub cards must provide prior to carrying out any actions associated with a file held on a GemClub card.

You can specify access conditions after you create the various files that will be used in the card. An access condition is specified for a specific function or group of functions. It references the secret code(s) and the secret key which protects the relevant data or element.

Functions Submitted for Access Conditions

The following functions are submitted for access conditions:

Functions	Description	Data Objects
Create	Create a data object.	All
Update/Delete	Update a data element or delete a data object. Update/Append a record.	All
Read	Read a Data Element/Record.	All (except for security files)
Award	Run an Award command.	Counters
Redeem	Run a Redeem command.	Counters
Use Rule	Run a Use Rule command.	Rules

Note: Both the Update and Delete functions are protected by the same access condition.

Rules for Defining Access Conditions

An access condition is defined over one byte as follows:

b7	b6	b5	b4	b3	b2	b1	b0
PIN	Condition		Short File Identifier				

Field	What is specified
b7	**1** If **PIN** code is also required. **0** If **PIN** code is not required.
b6 b5	**00**—Neither **Secret Code Protection** nor **Secure Messaging** is required. **01**—Specifies **Secret Code Protection**. **10** Specifies **Secure Messaging**. **11** Locks Access.
b4 b0	The file identifier of the Secret Code file or the Secret Key file used for the condition specified with the **b6 b5** condition.

Note: The secret code to be used as the PIN is referenced in the System file.

The following rules apply to codes which are specified for **Access Conditions** in GemClub:

What is Required	What to Do
Neither **Secret code protection** nor **Secure Messaging**.	Use the value x**00xxxxxb**.
Create **Open Access** to a specified function.	Use the value **000xxxxxb**.
Ensure that a specified function is **Locked**.	Use the value **x11xxxxxb**.
PIN code protection only.	Use the value **100xxxxxb**.
Secure Messaging protection only	Use the value **010xxxxxb**.
Secret Code protection only	Use the value **001xxxxxb**.
Secure Messaging and **PIN** protection	Use the value **110xxxxxb**.
Secret Code and **PIN** protection	Use the value **101xxxxxb**.

*Note: **Secret codes** protect access to the files for each access condition group.*

*If the tag **00h** is specified as the **PIN** code reference to be used in the **System file** of a GemClub card, **PIN** code verification will be bypassed in access condition verification.*

*Secure messaging protection is not supported for **Read** operations. If secure messaging protection is specified for **Read** operations, the operating system will interpret it as a locked access condition.*

File Access Conditions

In general, each file that you create on a GemClub card has access conditions (AC) specified for it. Each AC is defined for a group of commands or functions which are pertinent to the file.

Implicit Access Conditions

For certain data elements in GemClub, the access condition for 'Read' operation is internally known to the operating system.

The following data elements have an implicit access condition for 'Read' operations:

Data Object	Data Elements	Access Condition for Read
Secret Code file	All data elements	Locked
Secret Key file	All data elements	Locked

Security Mechanisms

Security Mechanisms are procedures that are used to protect the security of files stored on cards. These security mechanisms are also used to prevent the use of the GemClub card or one of the loyalty functions on a non-authorized terminal.

How Security Mechanisms Work

The card compares the secret code received from the terminal with the relevant secret code stored in the card. The reference to the internal secret code with which the OS makes the comparison can be:

- Transmitted in the command—for example, **Verify Secret Code** command.
- Implicitly known by the card—for example, **Verify PIN Code** command.
- Internally stored as an access condition in the data object involved in the command—for example, **Award, Use Rule**.

Entity Authentication with Secret Code

GemClub uses a **Secret Code Ratification Counter** which is specified in each secret code. It is used to prevent repeated attacks on the **Secret Code** value. The **Secret Code Ratification Counter** consists of the following elements:

- **Maximum Attempts Allowed**—Used to specify the number of incorrect consecutive attempts allowed to access the file. No more than 15 attempts can be specified in the file for the Maximum Attempts Allowed parameter.
- **Attempts Remaining** —Specifies the number of attempts outstanding before access to the file is blocked. No more than 15 attempts can be specified in the file for the Attempts Remaining parameter. After a successful presentation, the counter is returned to the number of Maximum Attempts Allowed specified.

The value of the Attempts Remaining parameter is always less than or equal to the value of the Maximum Attempts Allowed parameter.

Secret Code Administration Rules

The following rules apply to secret code administration:

- If the secret code is blocked, authentication fails. This is because the secret code presentations limit has been reached.
- The number of allowed presentations is decreased in EEPROM *prior* to a comparison being made between the secret code that has been presented, and the secret code stored on the card.
- Presentation of a secret code that does not match with the code stored in the card does not affect the global security status *except* where the wrong presentation causes the secret code to be blocked. In this situation, the operating system (OS) automatically cancels all information stored in the global security status.
- If a secret code is blocked, it can be unblocked by updating the value of the Attempts Remaining field. In order to do this, the Update / Delete access condition must be fulfilled (this is only possible if the secret code file is not protected by itself).

Data Authentication Secure Messaging

This mechanism ensures the authenticity and integrity of data exchanged between the terminal and the GemClub card. It also ensures confidentiality for key updating.

A MAC_IN is sent by the terminal with the command to be protected. The card will verify this MAC_IN before performing the command (the right verification ensures that the terminal has the rights to perform this action and ensures data transfer integrity). After performing the command, the card computes a MAC_OUT which will be verified by the terminal (ensuring the authenticity of the card and data transfer integrity).

Confidentiality—Confidentiality of key updating operations is assured.

Example

An example is provided of a data authentication procedure involving an '**Update Parameter**' command.

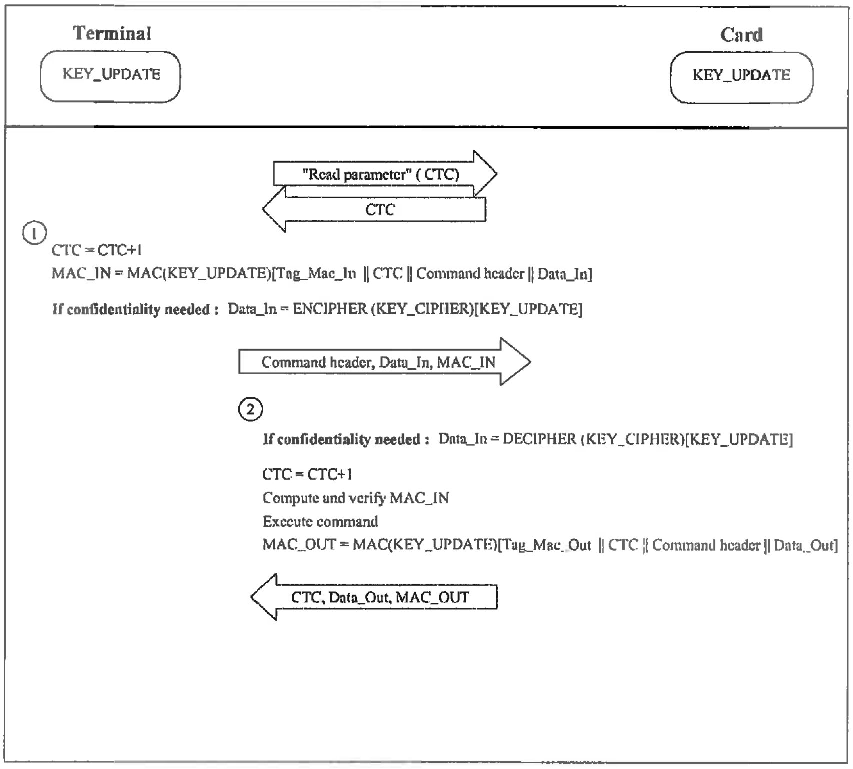

Terminal
KEY_UPDATE
Card
KEY_UPDATE
"Read parameter" (CTC)
CTC
①
CTC = CTC+1
MAC_IN = MAC(KEY_UPDATE)[Tag_Mac_In || CTC || Command header || Data_In]
If confidentiality needed : Data_In = ENCIPHER (KEY_CIPHER)[KEY_UPDATE]
Command header, Data_In, MAC_IN
②
If confidentiality needed : Data_In = DECIPHER (KEY_CIPHER)[KEY_UPDATE]
CTC = CTC+1
Compute and verify MAC_IN
Execute command
MAC_OUT = MAC(KEY_UPDATE)[Tag_Mac_Out || CTC || Command header || Data_Out]
CTC, Data_Out, MAC_OUT

Figure 4 Secure Messaging Update Parameter

Step 1

The terminal computes a MAC_IN using the secret key that has been predefined for the relevant function (KEY_UPDATE). This MAC must include the following details:

- Tag_Mac_In
- CTC
- Command Header
- Message to be transmitted to the card

If confidentiality is required (for a key value update) the terminal will then encrypt the message using the **3DES** encryption algorithm.

For more information the **3DES** algorithm see: '*GemClub Encryption Algorithms*'.

Step 2

The card then decrypts the message—if confidentiality has been specified—and computes and verifies the MAC that it has received from the terminal.

At the end of the command, the card computes a MAC_OUT using the secret key that has been predefined for the relevant functions (the same key that was used to compute the MAC_IN). This MAC must include the following details:

- Tag_Mac_Out
- CTC
- Command Header
- Message to be transmitted to the terminal

If authentication is successful, the card deems that the terminal has rights of access for the relevant function. In this case, the ratification counter associated with the secure messaging key is reset.

If authentication is unsuccessful, the card decreases the number of attempts remaining and rejects the command.

Step 3

The terminal verifies the MAC_OUT.

About the Secret Key Ratification Counter

The **Secret Key Ratification Counter** is present in every secret key file. It is used to prevent repeated attacks on the secret key value.

The counter consists of the following items:

- **Maximum Attempts Allowed**—Specifies the number of incorrect consecutive attempts allowed to access the file. No more than 15 attempts can be specified in the file for the Maximum Attempts Allowed parameter.
- **Attempts Remaining** —Specifies the number of attempts outstanding before access to the file is blocked. No more than 15 attempts can be specified in the file for the Attempts Remaining parameter. After a successful verification, the counter is returned to the number of Maximum Attempts Allowed specified.

The value of the Attempts Remaining parameter is always less than or equal to the value of the Maximum Attempts Allowed parameter.

Note: Authentication fails if the secret key is blocked.

The number of allowed presentations is decreased in EEPROM prior to a comparison being made between the MAC that has been presented, and the MAC computed by the card.

Transaction Proofs

A transaction proof is a certificate that is computed at the end of certain transactions by GemClub cards. It is used to prove the authenticity of the transaction to the loyalty operator (non-repudiation of the transaction). For example, after an '**Award**' operation a transaction proof can be computed.

Example: The card computes a transaction proof which is verified by the loyalty operator. The transaction can be either an **Award**, **Redeem** or **Use Rule** operation.

In this example, data authentication (secure messaging) is performed. However a transaction proof can be computed without data authentication.

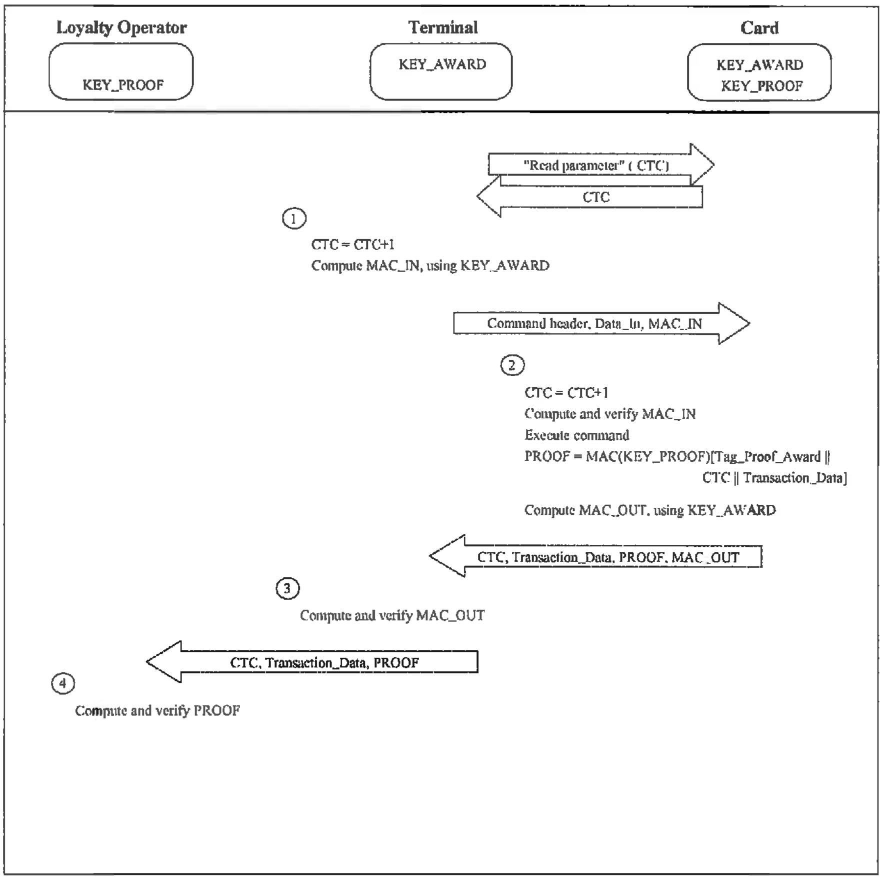

Loyalty Operator
KEY_PROOF
Terminal
KEY_AWARD
Card
KEY_AWARD
KEY_PROOF
"Read parameter" (CTC)
CTC
1
CTC = CTC+1
Compute MAC_IN, using KEY_AWARD
Command header, Data_In, MAC_IN
2
CTC = CTC+1
Compute and verify MAC_IN
Execute command
PROOF = MAC(KEY_PROOF)[Tag_Proof_Award ||
CTC || Transaction_Data]
Compute MAC_OUT, using KEY_AWARD
CTC, Transaction_Data, PROOF, MAC_OUT
3
Compute and verify MAC_OUT
CTC, Transaction_Data, PROOF
4
Compute and verify PROOF

Figure 5 Transaction Proof

Step 1

The terminal computes a MAC_IN using the secret key that has been predefined for the relevant function (KEY_AWARD). This MAC must include the following details:

- Tag_Mac_Award_In
- CTC
- Command Header
- Message to be transmitted to the card

Step 2

The card verifies the MAC_IN.

If the transaction is successful, it computes a transaction proof using the secret key that has been specified for transaction proof (KEY_PROOF). This proof includes the following details:

- Tag_Proof_Award
- Data involved in the transaction: Counter or rule identifier, terminal data, current data, transaction amount, status of rule execution (for a Use Rule operation)
- Card Transaction Counter, already increased at the beginning of the command

At the end of the command, the card computes a MAC_OUT using the secret key that has been predefined for the relevant functions. This MAC will include the following details:

- Tag_Mac_Award_Out
- CTC
- Command Header
- Message to be transmitted to the terminal.

Step 3

The terminal verifies the MAC_OUT.

- If the transaction is successful, the terminal transmits the transaction data and PROOF to the loyalty operator for verification and compensation.

Step 4

The loyalty operator then verifies the **PROOF**, and then gives approval for compensation in case of successful verification.

For more examples, see the Application note that comes with GemClub.

Rules for Transaction Proofs

The following rules are recommended for transaction proofs:

- A terminal is not able to compute the transaction proof.
- Secret keys for transaction proofs must not be used for any other purpose.
- Secret keys for transaction proofs are only known to GemClub cards and the loyalty operators.
- If the key reference for transaction proof specified in the counter file or rule file is **00h**, then the transaction proof requirement is disabled (the card returns a value of zero).

Access Condition Check Algorithm

GemClub cards use **Access Condition Checks** to find whether a user or merchant has the right to run any actions on the card. These **Access Condition Checks** are either implicit in the system—in certain operations the OS automatically carries out an access condition check—or they are specified for the file when it is being created.

Rules for Access Condition Checks

For **ISO Record Management Commands:**

- If the access condition check specified in the corresponding record file is a PIN code or a secret code, then the PIN or secret code must be presented to the card before running the '**ISO Record**' command.
- '**Verify Code**' command updates the global security status if code presentation is successful.
- If the access condition is **Secure Messaging**, then **Message Authentication Code** (MAC) will be included in the '**ISO Record**' command.

For other functions:

- The format of the APDU command does not depend on the access condition stored on the file. Accordingly if neither a MAC nor a secret code is requested, the authentication data will be replaced by zeroes.
- If the access condition stored in the file indicates that a secret code is required, the secret code value may be stored in the data field of the command.
- The Global Security Status will not be updated by this type of code presentation.
- If the access condition indicates that a PIN presentation is required, a '**Verify Code**' command will be carried out before the command is run. This command will update the Global Security Status of the card—for the PIN code only.

GEMCLUB ENCRYPTION ALGORITHMS

Overview

This chapter discusses secure messaging in GemClub as well as the message authentication codes specified for commands used in GemClub.

Secure Messaging in GemClub 3DES Algorithm

GemClub can guarantee secure messaging and encryption (for key value updates) through the use of the **Triple DES** Encryption/Decryption algorithm. GemClub can work with double length keys to perform a triple DES encryption/decryption with this algorithm, as follows:

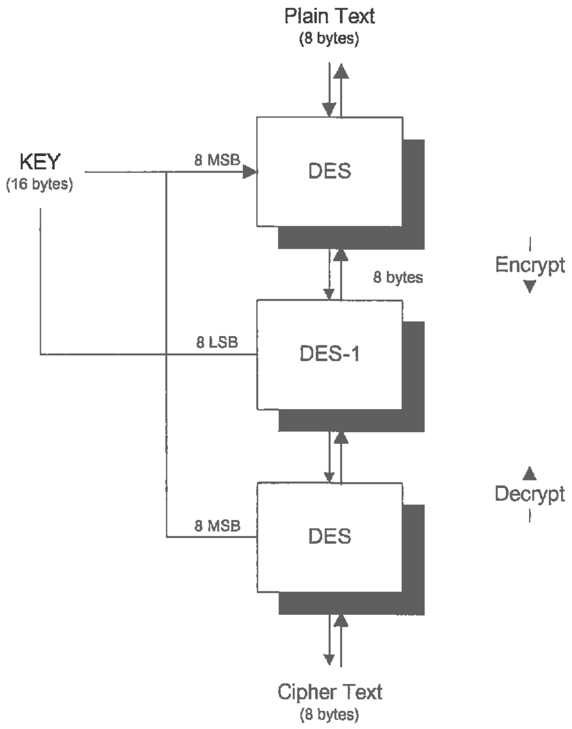


Figure 6 Triple DES Encryption/Decryption Algorithm

Note: MSB is the most significant byte.

LSB is the least significant byte.

MAC Calculation

In the GemClub card, a Message Authentication Code is computed in the last block of a CBC encryption algorithm in which the following conditions apply:

- DES compatible encryption is carried out on intermediary blocks using the 8 most significant bytes of the relevant key—in practice, the first half key.
- Triple DES encryption is carried out on the last block using all 16 bytes of the key.

MAC calculation is illustrated as follows:

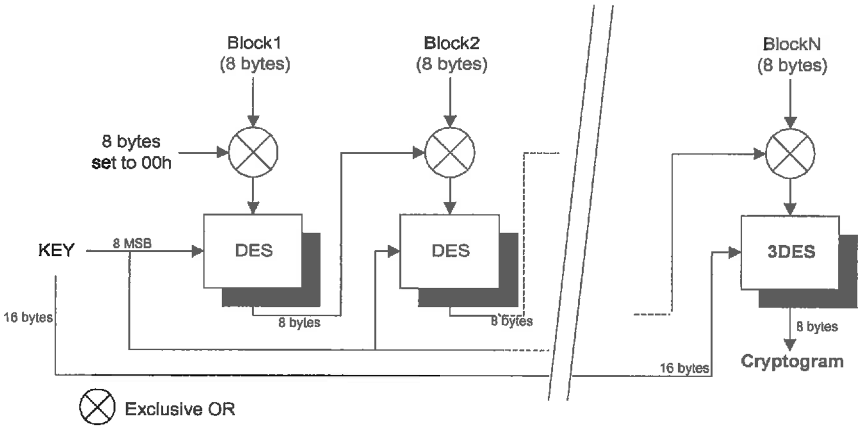


Figure 7 MAC Calculation

Cryptography and Commands in GemClub

GemClub uses a number of MACs in its operations. The content of the MAC data fields used by GemClub is as follows:

Note: A tag is a unique number one byte long that is included at the beginning of each data block involved in the MAC calculation.

Tags used in GemClub are defined later in this chapter.

For details of the contents of Incoming and Outgoing data, refer **to** *the corresponding command in the* GemClub Commands *chapter.*

Award Operations

MAC_IN

Name	Code
KEY (16 bytes)	**KEY_AWARD**—Key referenced in the relevant **Access Condition**.
TAG (1 byte)	**TAG_MAC_IN_AWARD**. See '*MAC Tag Administration Rules*'.
MAC data and padding	Block 1=[Tag$_{1\ \text{byte}}$ || CTC$_{2\ \text{bytes}}$ || Command header$_{5\ \text{bytes}}$] Block 2=[Tag$_{1\ \text{byte}}$ || Incoming data$_{7\ \text{bytes}}$] Block 3=[Tag$_{1\ \text{byte}}$ || Incoming data (continued)$_{7\ \text{bytes}}$] Block 4=[Tag$_{1\ \text{byte}}$ || 80h 00 00 00 00 00 00]

Transaction Proof

Name	Code
KEY (16 bytes)	**KEY_PROOF**—Key referenced in the relevant **Counter** for transaction proof calculation.
TAG (1 byte)	**TAG_PROOF_AWARD**. See '*MAC Tag Administration Rules*'.
MAC data and padding	Block 1=[Tag$_{1\ \text{byte}}$ || CTC$_{2\ \text{bytes}}$ || Counter identifier$_{1\ \text{byte}}$ || Terminal data$_{4\ \text{bytes}}$] Block 2=[Tag$_{1\ \text{byte}}$ || Terminal data (continued)$_{4\ \text{bytes}}$ || Current date$_{3\ \text{bytes}}$] Block 3=[Tag$_{1\ \text{byte}}$ || Transaction amount$_{3\ \text{bytes}}$ || 80h 00 00 00]

MAC_OUT

Name	Code
KEY (16 bytes)	**KEY_AWARD**—Key referenced in the relevant **Access Condition**.
TAG (1 byte)	**TAG_MAC_OUT_AWARD**. See '*MAC Tag Administration Rules*'.

Name	Code
MAC data and padding	Block 1=[Tag$_{1\ byte}$ || CTC$_{2\ bytes}$ || Command header$_{5\ bytes}$] Block 2=[Tag$_{1\ byte}$ || Outgoing data$_{7\ bytes}$] Block 3=[Tag$_{1\ byte}$ || Outgoing data (continued)$_{6\ bytes}$ || 80h]

Redeem Operations

MAC_IN

Name	Code
KEY (16 bytes)	**KEY_REDEEM**—Key referenced in the relevant **Access Condition**.
TAG (1 byte)	**TAG_MAC_IN_REDEEM**. See '*MAC Tag Administration Rules*'.
MAC data and padding	Block 1=[Tag$_{1\ byte}$ || CTC$_{2\ bytes}$ || Command header$_{5\ bytes}$] Block 2=[Tag$_{1\ byte}$ || Incoming data$_{7\ bytes}$] Block 3=[Tag$_{1\ byte}$ || Incoming data (continued)$_{7\ bytes}$] Block 4=[Tag $_{1\ byte}$ || 80h 00 00 00 00 00 00]

Transaction Proof

Name	Code
KEY (16 bytes)	**KEY_PROOF**—Key referenced in the relevant **Counter** for transaction proof calculation.
TAG (1 byte)	**TAG_PROOF_REDEEM**. See '*MAC Tag Administration Rules*'.
MAC data and padding	Block 1=[Tag$_{1\ byte}$ || CTC$_{2\ bytes}$ || Counter identifier$_{1\ byte}$ || Terminal data$_{4\ bytes}$] Block 2=[Tag$_{1\ byte}$ || Terminal data (continued)$_{4\ bytes}$ || Current date$_{3\ bytes}$] Block 3=[Tag$_{1\ byte}$ || Transaction amount$_{3\ bytes}$ || 80h 00 00 00]

MAC_OUT

Name	Code
KEY (16 bytes)	**KEY_REDEEM**—Key referenced in the relevant **Access Condition**.
TAG (1 byte)	**TAG_MAC_OUT_REDEEM**. See '*MAC Tag Administration Rules*'.
MAC data and padding	Block 1=[Tag$_{1\ byte}$ || CTC$_{2\ bytes}$ || Command header$_{5\ bytes}$] Block 2=[Tag$_{1\ byte}$ || Outgoing data$_{7\ bytes}$] Block 3=[Tag$_{1\ byte}$ || Outgoing data (continued)$_{6\ bytes}$ || 80h]

Use Rule Operations

MAC_IN

Name	Code
KEY (16 bytes)	**KEY_USE_RULE**—Key referenced in the relevant **Access Condition**.
TAG (1 byte)	**TAG_MAC_IN_USE_RULE**. See '*MAC Tag Administration Rules*'.
MAC data and padding	Block 1=[Tag$_{1\ \text{byte}}$ ‖ CTC$_{2\ \text{bytes}}$ ‖ Command header$_{5\ \text{bytes}}$] Block 2=[Tag$_{1\ \text{byte}}$ ‖ Incoming data$_{7\ \text{bytes}}$] Block 3=[Tag$_{1\ \text{byte}}$ ‖ Incoming data (continued)$_{7\ \text{bytes}}$] Block 4=[Tag$_{1\ \text{byte}}$ ‖ 80h 00 00 00 00 00 00]

Transaction Proof

Name	Code
KEY (16 bytes)	**KEY_PROOF**—Key referenced in the relevant **Counter** for transaction proof calculation.
TAG (1 byte)	**TAG_PROOF_USE_RULE**. See '*MAC Tag Administration Rules*'.
MAC data and padding	Block 1=[Tag$_{1\ \text{byte}}$ ‖ CTC$_{2\ \text{bytes}}$ ‖ Rule identifier$_{1\ \text{byte}}$ ‖ Terminal data$_{4\ \text{bytes}}$] Block 2=[Tag$_{1\ \text{byte}}$ ‖ Terminal data (continued)$_{4\ \text{bytes}}$ ‖ Current date$_{3\ \text{bytes}}$] Block 3=[Tag$_{1\ \text{byte}}$ ‖ Transaction amount$_{3\ \text{bytes}}$ ‖ Status of rule execution$_{1\ \text{byte}}$ ‖ 80h 00 00]

MAC_OUT

Name	Code
KEY (16 bytes)	**KEY_USE_RULE**(REDEEM)—Key referenced in the relevant **Access Condition**.
TAG (1 byte)	**TAG_MAC_OUT_USE_RULE**. See '*MAC Tag Administration Rules*'.
MAC data and padding	Block 1=[Tag$_{1\ \text{byte}}$ ‖ CTC$_{2\ \text{bytes}}$ ‖ Command header$_{5\ \text{bytes}}$] Block 2=[Tag$_{1\ \text{byte}}$ ‖ Outgoing data$_{7\ \text{bytes}}$] Block 3=[Tag$_{1\ \text{byte}}$ ‖ Outgoing data (continued)$_{5\ \text{bytes}}$ ‖ 80h 00]

Other Secure Messaging Operations

There are a number of other commands in GemClub which use MAC. These include the following:

- Update Parameter
- Create Object
- Delete Object
- Update Record
- Append Record

MAC_IN

<table>
<tr><th>Name</th><th>Code</th></tr>
<tr><td>KEY (16 bytes)</td><td><b>KEY_UPDATE_PARAM</b> —Key referenced in the relevant <b>Access Condition</b>.</td></tr>
<tr><td>TAG (1 byte)</td><td><b>TAG_MAC_IN_UPDATE_PARAM</b>. See ‘<i>MAC Tag Administration Rules</i>’.</td></tr>
<tr><td>MAC data and padding</td><td>Block 1=[Tag$_{1\ \text{byte}}$ || CTC$_{2\ \text{bytes}}$ || Command header$_{5\ \text{bytes}}$]
Block2 =[Tag$_{1\ \text{byte}}$ || Incoming data$_{\text{variable}}$ ...]
—
Block n=[[Tag$_{1\ \text{byte}}$ || Incoming data$_{\text{variable}}$ ||]... Padding]</td></tr>
</table>

Note: See Rules For Padding later in this chapter.

MAC_OUT

<table>
<tr><th>Name</th><th>Code</th></tr>
<tr><td>KEY (16 bytes)</td><td><b>KEY_UPDATE_PARAM</b>—Key referenced in the relevant <b>Access Condition</b>.</td></tr>
<tr><td>TAG (1 byte)</td><td><b>TAG_MAC_OUT_UPDATE_PARAM</b>. See ‘<i>MAC Tag Administration Rules</i>’.</td></tr>
<tr><td>MAC data and padding</td><td>Block 1=[Tag$_{1\ \text{byte}}$ || CTC$_{2\ \text{bytes}}$ || Command header$_{5\ \text{bytes}}$]
Block 2=[Tag$_{1\ \text{byte}}$ || Outgoing data$_{2\ \text{bytes}}$ || 80h 00 00 00 00]</td></tr>
</table>

MAC Tag Administration Rules

GemClub Commands have the following predefined tags:

Command	Tag
TAG_MAC_IN_AWARD	11h
TAG_MAC_OUT_AWARD	12h
TAG_PROOF_AWARD	13h
TAG_MAC_IN_REDEEM	21h
TAG_MAC_OUT_REDEEM	22h
TAG_PROOF_REDEEM	23h
TAG_MAC_IN_USE_RULE	31h
TAG_MAC_OUT_USE_RULE	32h
TAG_PROOF_USE_RULE	33h
TAG_MAC_IN_UPDATE_PARAM	51h
TAG_MAC_OUT_UPDATE_PARAM	52h
TAG_MAC_IN_CREATE	61h
TAG_MAC_OUT_CREATE	62h
TAG_MAC_IN_DELETE	71h
TAG_MAC_OUT_DELETE	72h
TAG_MAC_IN_UPDATE_REC	81h
TAG_MAC_OUT_UPDATE_REC	82h
TAG_MAC_IN_APPEND_REC	91h
TAG_MAC_OUT_APPEND_REC	92h

Rules for Padding

As all the cryptography features used in GemClub are based on the 3DES algorithm, the length of any block of data that is to be encrypted or decrypted must be a multiple of eight bytes. If the amount of data that is being transmitted or received does not completely fill these eight–byte blocks of data, then each unfilled block must be 'filled' with **Padding** data. Rules for padding are as follows:

- Data that is to be encrypted or decrypted must fulfill the rules specified in the 3DES algorithm—It must be sent in blocks which are multiples of eight bytes in length.
- In the MAC calculation process, at least one byte with the value '80h' will be added to the MAC data. If the MAC data is still not a multiple of eight bytes, the MAC data will be right–padded with zeroes.

Examples:

1. MAC data length is 13 bytes.

   In this situation the padding string will be—in hex:

   80 00 00h, and 16 bytes (two blocks) will be used in the MAC calculation process.

2. MAC data length is 15 bytes.

   In this situation the padding string will be—in hex:

   80h, and 16 bytes (two blocks) will be used in the MAC calculation process.

3. MAC data length is 16 bytes

   In this case, the padding string will be—in hex:

   [$\text{Tag}_{1\ \text{byte}}$ || **80 00 00 00 00 00 00h**], and 24 bytes (three blocks) will be involved in the MAC calculation process.

OTHER GEMCLUB FEATURES

Overview

This chapter introduces the GemClub **Backup Procedure** and the GemClub **Point Conversion Feature.**

GemClub Backup Procedure

To ensure that sensitive information is secure and cannot be altered or corrupted in transit or during a power failure, GemClub contains a '**file backup**' feature. GemClub uses this feature to 'backup' all information stored on the card *prior* to any failed attempt to run a command on the card.

This feature is activated in the event of a command being blocked or a file being accidentally corrupted. Should either of these events occur, GemClub will default to the information that has been automatically stored on the file backup prior to the command being run.

The file backup procedure is automatically activated each time that the following operations are used:

- **Create / Delete Object** operations
- **Update Parameter** operations—every data element involved.
- **Award / Redeem** operations—the Balance and Counters, where they exist.
- **Use Rule** operations—counters involved in macro processing.
- Each time system data is updated (checksum, file descriptor).

Running the Backup Procedure

The file backup procedure is automatically run in GemClub when the following actions are carried out:

- Updating a file of the GemClub application in EEPROM (**Update Parameter**, **Award**, **Redeem**, **Use Rule** commands).
- Updating internal information, including checksum, secret code and key ratification counter.

For a file update, the file backup procedure is only completed after the **Get Response** command has been successfully run.

Note: You cannot run the backup procedure when you are updating a record that is stored in an ***ISO 7816-4*** *compatible file.*

GemClub Rule Mechanism Point Conversion

You can use the Rule procedure provided in all GemClub cards to **Award** or **Redeem** points to or from **counters**.

Up to four macro instructions can be specified to be carried out during the running of a specific **rule**. The same **rule** can be specified to run on up to four different **counters**.

Each macro instruction must have the following information specified:

- ID of the counter to be used
- Type of function to run—Award and Redeem
- Parameter A
- Parameter B

Once you specify a **Point Conversion Use Rule** command, GemClub automatically converts any incoming **Use Rule** command—that is pertinent to this point conversion command—into a number of points to be awarded or redeemed as required.

Parameters A and B are used to define the point conversion rule that will be used on the counter.

Parameter A: The First Conversion parameter. This parameter represents the number of points that are to be added to or subtracted from the counter each time that the value specified in Parameter B is attained.

Parameter B: The Second Conversion parameter. This parameter is used to specify the value that must be attained prior to points being added to or subtracted from the counter that has been specified for the card. This value is expressed in the same units as the transaction amount that has been received in the command field of the '**Use Rule**', (for example, number of air miles, number of liters of petrol).

The point conversion equation used in GemClub is as follows:

$$Points = ParameterA \times \mathrm{E}\left(\frac{Amount}{ParameterB}\right)$$

Note: *In this equation '* E *' represents the integer division.*

It follows that if **50** points are to be awarded to counter **0Dh** for every 1000 air miles flown

Parameter **A** = **50**

Parameter **B** = **1000**

And if the cardholder has flown **4,990** air miles

{**Amount** = **4990**}

Then:

$$Points = 50 \times \mathrm{E}\left(\frac{4990}{1000}\right)$$

The GemClub cardholder will receive **200** points.

Note: If Parameter B is zero, then the number of points awarded equals Parameter A regardless of the amount received.

GEMCLUB COMMAND FORMAT

Overview

This chapter discusses the **Application Protocol Data Unit** (**APDU**) message structure used by GemClub.

GemClub cards accept commands and responses in compliance with the Application Data Protocol (APDU) format defined by the ISO 7816-4 standard. This enables GemClub commands to be compatible with the Gemplus GCR Interface Driver Library. You should bear in mind, however that the GemClub transport layer protocol complies with the ISO 7816-3 T=0 standard, which converts APDUs into Transport Protocol Data Units (TPDU)s. The reader sends TPDU commands to the card, and the card returns TPDU responses to the reader.

GemClub cards accept commands in any of the following cases:

Case 1-No command or response data. This is sent as a T=0 ISO IN TPDU with length = 0.

Case 2-Short format: no command data, but with between 1 and 256 bytes response data. This is sent as a T=0 ISO OUT TPDU.

Case 3-Short format: command data between 1 and 255 bytes and no response data. This is sent as a T=0 ISO IN TPDU.

Case 4-Short format: command data between 1 and 255 bytes, response data between 1 and 256 bytes. The command is sent as a T=0 ISO IN TPDU. It must be followed by a **Get Response** command sent as a T=0 ISO OUT TPDU. The **Get Response** mechanism is compliant with the ISO 7816-4 standard.

Command Format

In their default configuration (T=0), GemClub cards accept commands in the following format:

Header	Body		
CLA INS P1 P2	Lc	Parameters/data	Le

The fields are described below:

Header Fields

Header fields are mandatory and are used as follows:

Field	Length	What it is
CLA	**1**	Class of Command—type of command
INS	**1**	Command Code—code that has been specified for the command
P1	**1**	Command Parameter
P2	**1**	Command Parameter