GLOBALPLATFORM

THE STANDARD FOR SMART CARD INFRASTRUCTURE

GlobalPlatform

Card Specification

Version 2.2

March 2006

Recipients of this document are invited to submit, with their comments, notification of any relevant patent rights or other intellectual property rights of which they may be aware which might be infringed by the implementation of the specification set forth in this document, and to provide supporting documentation.

Table of Contents

Table of Figures and Tables

Part I
Introduction

1 Introduction

GlobalPlatform is an organization that has been established by leading companies from the payments and communications industries, the government sector and the vendor community, and is the first to promote a global infrastructure for smart card implementation across multiple industries. Its goal is to reduce barriers hindering the growth of cross-industry, multiple Application smart cards. The smart card issuers will continue to have the freedom to choose from a variety of cards, terminals and back-end systems.

For smart cards to reach their true potential, consumers need to be able to use them for a wide variety of functions. For example, the cards can be used with mobile phones to make purchases over the Internet as well as to securely access a PC. Smart cards should also be cost effective and easily multifunctional.

Beginning in the mid-1990s, a number of very significant breakthroughs occurred in the chip card industry with the introduction of open systems specifications for Application development. The leading technologies in this area are Java Card™ and MULTOS™. These technology specifications provide an important contribution to the solution towards the multi-Application chip card vision, such as common programming standards allowing Application portability between different card specific implementations.

Then in 2001 a PKI-based card content management framework was defined by NICSS (the Next generation Ic Card System Study group) especially for the governmental sector market. The concept of NICSS is to separate the application provider from card issuer in a trusted manner whereby applications providers can 'rent' card memory space from the card issuer. This creates a new business model for each stakeholder. Additionally, in the mobile telecommunication sector ETSI have been addressing card content management 'over the air' for the secure management of the SIM and third generation UICC.

Through the Open Platform initiative, first Visa International and now GlobalPlatform have been working with the chip card industry to deliver a missing and critically important chip card standard — a hardware-neutral, vendor-neutral, Application-independent card management specification. This new specification provides a common security and card management architecture that protects the most important aspect of a chip card system investment — the infrastructure.

GlobalPlatform defines a flexible and powerful specification for Card Issuers to create single- and multi-Application chip card systems to meet the evolution of their business needs. The specification allows them to choose the card technology that is right for them today while also ensuring that they can migrate, if necessary, to a different card technology in the future without significant impact to their infrastructure.

This specification describes the GlobalPlatform Specifications that shall be implemented on GlobalPlatform smart cards.

The following meanings apply to SHALL, SHOULD, and MAY in this document:

- SHALL indicates that the statement containing the SHALL must be implemented as defined in this Specification. It does not mandate the implementation of the statement;
- SHOULD indicates a recommendation. It is strongly recommended to implement the statement as defined in this Specification;
- MAY indicates an option.

1.1 Audience

This specification is intended primarily for card manufacturers and application developers developing GlobalPlatform card implementations. Although this specification defines card components, command interfaces, transaction sequences, and interfaces that can be common across many different industries, it does not detail the implementation of the lower layers security, which may vary from one industry to the other.

This specification is also intended for a more general audience as it describes the generic security concepts and the various actors involved in a multi-Application Card Management System.

1.2 Normative References

Standard / Specification	Description
ANSI X9.52	Triple Data Encryption Algorithm Modes of Operation, draft, 1996
CWA 14890-1	CEN Workshop Agreement - Application Interface for smart cards used as Secure Signature Creation Devices - Part 1: Basic requirements, March 2004
ETSI TS 102 221 (Release 6)	Smart cards; UICC - Terminal interface; Physical and logical characteristics, European Telecommunications Standards Institute Project Smart Card Platform (EP SCP), 2004
ETSI TS 102 225 (Release 6)	Smart cards; Secured packet structure for UICC based applications, European Telecommunications Standards Institute Project Smart Card Platform (EP SCP), 2004
ETSI TS 102 226 (Release 6)	Smart cards; Remote APDU structure for UICC based applications, European Telecommunications Standards Institute Project Smart Card Platform (EP SCP), 2004
ETSI TS 102 241 (Release 6)	Smart cards; UICC Application Programming Interface (UICC API) for Java Card™, European Telecommunications Standards Institute Project Smart Card Platform (EP SCP), 2004
FIPS PUB 180-2	Federal Information Processing Standards Publication 180-2, 2002: Specifications for the Secure Hash Standard: U.S. Department Of Commerce, Technology Administration, National Institute Of Standards And Technology
FIPS PUB 197	Federal Information Processing Standards Publication 197, 2001: Specification for the Advanced Encryption Standard (AES): U.S. Department Of Commerce, Technology Administration, National Institute Of Standards And Technology
FIPS PUB 198	Federal Information Processing Standards Publication 198, 2002: Standard for the Keyed-Hash Message Authentication Code (HMAC): U.S. Department Of Commerce, Technology Administration, National Institute Of Standards And Technology
GlobalPlatform KMS	GlobalPlatform Key Management System Functional Requirements, version 1.0
GlobalPlatform MS	GlobalPlatform Messaging Specification, version 1.0
GlobalPlatform SCMS	GlobalPlatform Smart Card Management System Functional Requirements, version 4.0
GlobalPlatform Systems Scripting Specification	GlobalPlatform Systems Scripting Specification, version 1.1
ISO/IEC 7816-3:1997	Identification cards - Integrated circuit(s) cards with contacts - Part 3: Electronic signals and transmission protocols
ISO/IEC 7816-4:2005	Identification cards – Integrated circuit cards - Part 4: Organization, security and commands for interchange

Standard / Specification	Description
ISO/IEC 7816-6:2004	Identification cards - Integrated circuit(s) cards with contacts- Part 6: Interindustry data elements
ISO/IEC 7816-15:2004	Identification cards - Integrated circuit cards with contacts - Part 15: Cryptographic information application
ISO 8731-1:1987	[IS8731-1] Banking - Approved algorithms for message authentication – Part 1: DEA
ISO/IEC 8825-1:2002 | ITU-T Recommendation X.690 (2002)	Information technology – ASN.1 encoding rules: Specification of Basic Encoding Rules (BER), Canonical Encoding Rules (CER) and Distinguished Encoding Rules (DER)
ISO/IEC 9594-8 | ITU-T Recommendation X.509 (2000)	Information Technology - Open Systems Interconnection - The Directory: Public-Key and Attribute Certificate Frameworks
ISO/IEC 9796-2:2002	Information technology - Security techniques - Digital signature schemes giving message recovery - Part 2: Integer factorization based mechanisms
ISO/IEC 9797:1994	[IS9797] Information technology – Security techniques - Data integrity mechanism using a cryptographic check function employing a block cipher algorithm
ISO/IEC 9899:1999	Programming languages - C
ISO/IEC 10116: 1997	[IS10116] Information technology - Modes of operation of an n-bit block cipher algorithm
ISO/IEC 10118-3: 1998	[IS10118-3] Information technology – Security techniques - Hash functions –Part 3: Dedicated hash functions
ISO/IEC 14443-3:2001	Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 3: Initialization and anticollision
ISO/IEC 14443-4:2001	Identification cards - Contactless integrated circuit(s) cards - Proximity cards - Part 4: Transmission protocol
ISO/IEC 18033-3:2005	Information Technology - Security techniques - Encryption algorithms - Part 3: Block ciphers
Java Card™	Go to the following website for Java Card™ documentation: http://java.sun.com/products/javacard
MAO-DOC-REF-009	MULTOS Guide to Generating Application Load Units, version 2.51
MULTOS™	Go to the following website for MULTOS™ documentation: http://www.multos.com
NICSS Framework (NICSS-F)	NICSS Prerequisites, First Edition, version 1.20, April 24, 2001, The Next generation IC Card System Study group
PKCS#1 (RFC 2437)	PKCS #1 v2.0: RSA Cryptography Standard, RSA Laboratories, October 1998 (RFC 2437).

Table 1-1: Normative References

1.3 Terminology and Definitions

Table 1-2 defines the expressions used within this Specification that use an upper case first letter in each word of the expression. Expressions within this document that use a lower case first letter in each word take the common sense meaning. (Tagged data elements are also given an upper case first letter in each word of their names.)

Term	Definition
Application	Instance of an Executable Module after it has been installed and made selectable
Application Management System	An off-card application-specific system required to successfully implement an Application Provider's service to a cardholder
Application Protocol Data Unit (APDU)	Standard communication messaging protocol between a card accepting device and a smart card
Application Provider	Entity that owns an application and is responsible for the application's behavior
Application Session	The link between the Application and the external world on a logical channel starting with the selection of the Application and ending when another Application selection occurs on the logical channel, the logical channel is closed or the Card Session terminates
Asymmetric Cryptography	A cryptographic technique that uses two related transformations, a public transformation (defined by the Public Key component) and a private transformation (defined by the Private Key component); these two key components have a property so that it is computationally infeasible to discover the Private Key, even if given the Public Key
Basic Logical Channel	The permanently available interface between the card and an external entity. The Basic Logical Channel is numbered zero
Card Content	Code and Application information (but not Application data) contained in the card that is under the responsibility of the OPEN e.g. Executable Load Files, Application instances, etc
Card Image Number (CIN)	An identifier for a specific GlobalPlatform card
Card Issuer	Entity that owns the card and is ultimately responsible for the behavior of the card
Card Management System	An off-card system providing functions to manage various card types and their associated application(s)and specific configurations for cardholders
Card Manager	Generic term for the card management entities of a GlobalPlatform card i.e. the OPEN, Issuer Security Domain and a Cardholder Verification Method services provider
Card Recognition Data	Information that tells an external system, in particular a Smart Card Management System (SCMS), how to work with the card (including indicating that this is a GlobalPlatform card)
Card Session	The link between the card and the external world starting at card reset (contact cards), activation (contactless cards) or power on of the card and ending with a subsequent reset (contact cards), deactivation (contactless cards) or power off of the card
Card Unique Data	Data that uniquely identifies a card being the concatenation of the Issuer Identification Number and Card Image Number
Cardholder	The end user of a card
Cardholder Verification Method (CVM)	A method to ensure that the person presenting the card is the person to whom the card was issued

Term	Definition
Certificate	In this Specification, a Certificate refers to a key certificate: the public key and identity of an entity together with some other information, rendered unforgeable by signing with the private key of the certification authority which issued that Certificate.
Controlling Authority	A Controlling Authority has the privilege to keep the control over the Card Content through the mandating of DAP Verification
Current Security Level	A level of security that is to be applied to the current command-response pair in a Secure Channel Protocol using secure messaging. It is set for an individual command (APDU pair): the current incoming command APDU and the next response.
DAP Block	Part of the Load File used for ensuring Load File Data Block verification
DAP Verification	A mechanism used by a Security Domain to verify that a Load File Data Block is authentic
Delegated Management	Pre-authorized Card Content changes performed by an approved Application Provider
Digital Signature	A cryptographic transformation of data that allows the recipient of the data to prove the origin and integrity of the data; it protects the sender and the recipient of the data against forgery by third parties; it also protects the sender against forgery by the recipient
Executable Load File	Actual on-card container of one or more application's executable code (Executable Modules). It may reside in Immutable Persistent Memory or may be created in Mutable Persistent Memory as the resulting image of a Load File Data Block
Executable Module	Contains the on-card executable code of a single application present within an Executable Load File
GlobalPlatform Registry	A container of information related to Card Content management
Host	A logical term used to represent the back end systems that support the GlobalPlatform system; hosts perform functions such as authorization and authentication, administration, Post-Issuance application code and data downloading, and transactional processing
Immutable Persistent Memory	Memory that can only be read
Issuer Security Domain	The primary on-card entity providing support for the control, security, and communication requirements of the card administrator (typically the Card Issuer)
Life Cycle	The existence of Card Content on a GlobalPlatform card and the various stages of this existence where applicable; or the stages in the life of the card itself
Life Cycle State	A specific state within the Life Cycle of the card or of Card Content
Load File	A file transferred to a GlobalPlatform card that contains a Load File Data Block and possibly one or more DAP Blocks
Load File Data Block	Part of the Load File that contains one or more application(s) or libraries and support information for the application(s) as required by the specific platform
Load File Data Block Hash	A value providing integrity for the Load File Data Block

Term	Definition
Load File Data Block Signature	A value encompassing the Load File Data Block Hash and providing both integrity and authenticity of the Load File Data Block
Message Authentication Code (MAC)	A symmetric cryptographic transformation of data that provides data origin authentication and data integrity
Mutable Persistent Memory	Memory that can be modified
OPEN	The central on-card administrator that owns the GlobalPlatform Registry
Post-Issuance	Phase following the card being issued to the Cardholder
Pre-Issuance	Phase prior to the card being issued to the Cardholder
Private Key	The private component of the asymmetric key pair
Public Key	The public component of the asymmetric key pair
Receipt	An cryptographic value provided by the card (if so required by the Card Issuer) as proof that a Delegated Management operation has occurred
Retry Counter	A counter, used in conjunction with the Retry Limit, to determine when attempts to present a CVM value shall be prohibited
Retry Limit	The maximum number of times an invalid CVM value can be presented prior to the CVM prohibiting further attempts to present a CVM value
Runtime Environment	Functionality on a card which provides a secure environment for multiple applications to operate. Its role is complementary to that of the GlobalPlatform Card Manager
Secure Channel	A communication mechanism between an off-card entity and a card that provides a level of assurance, to one or both entities
Secure Channel Protocol	A secure communication protocol and set of security services
Secure Channel Session	A session, during an Application Session, starting with the Secure Channel initiation and ending with a Secure Channel termination or termination of either the Application Session or Card Session
Security Domain	On-card entity providing support for the control, security, and communication requirements of an off-card entity (e.g. the Card Issuer, an Application Provider or a Controlling Authority)
Session Security Level	A mandatory minimum level of security to be applied to protected commands in a Secure Channel Protocol using secure messaging. It is established during the initialization of the Secure Channel Session, either explicitly or implicitly.
Supplementary Logical Channel	Up to 19 additional interfaces (other than the permanently available Basic Logical Channel) between the card and an external entity. Each Supplementary Logical Channel is numbered from 1 up to 19
Supplementary Security Domain	A Security Domain other than the Issuer Security Domain
Symmetric Cryptography	A cryptographic technique that uses the same secret key for both the originator’s and the recipient’s transformation
Token	A cryptographic value provided by a Card Issuer as proof that a Delegated Management operation has been authorized
Trust Point	An authority whose public key is trusted by a Security Domain or Off-Card Entity through some undefined mechanism such as a secure process that delivers the public key in a self-signed certificate. A Trust Point's public key is typically the 'highest' public key known to the entity.

Term	Definition
UICC	The ICC as defined by ETSI Project Smart Card Platform (EP SCP)

Table 1-2: Terminology and Definitions

1.4 Abbreviations and Notations

Abbreviation	Meaning
AES	Advanced Encryption Standard
AID	Application Identifier
APDU	Application Protocol Data Unit
API	Application Programming Interface
ASCII	American Standard Code for Information Interchange
ATR	Answer-to-Reset
ATQ	Answer-to-Request (for contactless cards)
BCD	Binary Coded Decimal
BER	Basic Encoding Rules
CAT	Card Application Toolkit; or Cryptographic Authorization Template
CBC	Cipher Block Chaining
CCT	Control Reference Template for Cryptographic Checksum
CIN	Card Image Number / Card Identification Number
CLA	Class byte of the command message
CRT	Control Reference Template
CT	Control Reference Template for Confidentiality
CVM	Cardholder Verification Method
DAP	Data Authentication Pattern
DEK	Data Encryption Key
DER	Distinguished Encoding Rules
DES	Data Encryption Standard
DST	Control Reference Template for Digital Signature
ECB	Electronic Code Book
EMV	Europay, MasterCard, and Visa; used to refer to the ICC Specifications for Payment Systems
ENC	Encryption
FCI	File Control Information
HEX	Hexadecimal
HMAC	Keyed-Hash Message Authentication Code
ICC	Integrated Circuit Card
ICV	Initial Chaining Vector
IIN	Issuer Identification Number
INS	Instruction byte of the command message
ISO	International Organization for Standardization
Lc	Exact length of data in a case 3 or case 4 command
Le	Maximum length of data expected in response to a case 2 or case 4 command
LV	Length Value
MAC	Message Authentication Code

Abbreviation	Meaning
MEL	MULTOS Executable Language. The instruction set of the MULTOS™ runtime environment
OID	Object Identifier
P1	Reference control parameter 1
P2	Reference control parameter 2
PIN	Personal Identification Number
PKI	Public Key Infrastructure
RAM	Random Access Memory
RFU	Reserved for Future Use
RID	Registered Application Provider Identifier
ROM	Read-only Memory
RSA	Rivest / Shamir / Adleman asymmetric algorithm
SCP	Secure Channel Protocol; or (ETSI) Smart Card Platform
SW	Status Word
SW1	Status Word One
SW2	Status Word Two
TLV	Tag Length Value
TP	Trust Point
'xx'	Hexadecimal values are expressed as hexadecimal digits between single quotation marks
'X'	A value in a cell of a table whose purpose is described in the 'meaning' column of the table
'-'	A value (0 or 1) in a cell of a table that does not affect the 'meaning' given for that row of the table

Table 1-3: Abbreviations and Notations

1.5 Revisions History

1.5.1 Open Platform Card Specification v2.0 to Open Platform Card Specification v2.0.1

This section provides a brief summary of the revisions made to the Open Platform Card Specification 2.0 Card Specification in the Open Platform Card Specification 2.0.1'.

Wording and formatting of the specification had been improved.

Anything relating to a specific implementation of Open Platform had been removed from the main body of the specification and was detailed in the appendices.

Anything specific relating to the personalization of Open Platform or Applications had been removed.

The changes relating specifically to the Java Card™ implementation of Open Platform were listed in the beginning of appendix A - *GlobalPlatform API*.

All the issues identified in the FAQ document dated April-June 1999 and the FAQ documents dated October-November 1999 and relating strictly to Open Platform, had been included in this version. The one caveat to this was the point 3.1.20 of the FAQ document dated October-November 1999 i.e. it is now required that the Security Domain associated with an Application be the same Security Domain used to perform Delegated Management functions for this Application.

The inclusion of Part V described a specific use of security and key management that was not present in Open Platform 2.0.

1.5.2 Major Adjustments in GlobalPlatform Card Specification V2.1

The following major adjustments are the modifications decided by GlobalPlatform Members. All of these modifications are intended to make the GlobalPlatform more usable for a wider number of entities while maintaining backwards compatibility. The minor editing changes and rewording for readability are not listed.

1.5.2.1 Enhancement to DAP Verification Scheme

In the process of implementing GlobalPlatform cards that supported Delegated Management and DAP Verification, it was determined that the method defined in the previous version of Open Platform necessitated the verification of multiple signatures on large blocks of data (Load File and Load File Data Block) that differed only slightly. When multiple DAP Verifications and possibly Delegated Management was being performed simultaneously, multiple hash generations were required to run concurrently. This negatively impacted the performance of the load process.

A new method has been defined in which instead of signing large blocks of data, a hash of the critical large block of data (Load File Data Block) may be generated and this hash signed. In this new method signatures are required on very much smaller blocks of information. Only one hash generation and check is required on the Load File Data Block.

1.5.2.2 Application Reception of Data from Security Domains

The Secure Channel mechanism previously provided by Open Platform only allowed an Application to request services from its associated Security Domain.

A new service is now provided that may be initiated by a Security Domain. It allows data to be passed by the Security Domain associated with an Application to the Application for further processing. The main purpose of this service is to facilitate the personalization of Applications through the Applications' associated Security Domain. A new INSTALL command has been defined to identify the Application to be personalized.

1.5.2.3 Security Domain Association and Extradition

In the previous Open Platform Card Specification an Executable Load File was associated with a Security Domain and all Applications instantiated from an Executable Load file, when installed, were also associated with the same Security Domain. This method was restrictive.

The new scheme provides Application Extradition. Application Extradition allows an Application that is already associated with a Security Domain to be extradited and associated with another Security Domain.

Another benefit provided by this enhancement is that in addition to Executable Load Files and Applications, now applications within Executable Load Files become visible in the GlobalPlatform Registry at the time that the Executable Load File is registered.

In order to avoid confusion between selectable Applications and the applications within an Executable Load File a new term has been introduced i.e. Executable Module. The term Executable Module is intended to identify the one or more applications present within an Executable Load File.

1.5.2.4 Executable Modules

In the previous Open Platform Card Specification, an off-card entity could only retrieve information relating to the Executable Load Files and selectable Applications present on the card. In order to enhance the information returned by the GET STATUS command, an additional set of information will be stored in the GlobalPlatform Registry and returned in the response to the GET STATUS command. This information relates to the Executable Module and it is

now possible to also retrieve information relating to application code within an Executable Load File that is available for installation.

1.5.2.5 Card Recognition Data

A concerted effort was made to ensure that there would be a uniform method to determine basic information about a card. The Card Recognition Data and the method for retrieving this data have been included in this version of the GlobalPlatform Card Specification. Information such as: this is a GlobalPlatform card, implemented in a particular way, and with a particular version number is now available.

1.5.2.6 Support of New Secure Channel Protocols

In order to be more accommodating, the method for including additional Secure Channel Protocols has been formalized. While this was allowed in previous versions of the Open Platform Card Specification, only one Secure Channel Protocol was defined. As part of the efforts of GlobalPlatform a formal process for including Secure Channel Protocols is defined. This version of the specification details two Secure Channel Protocols and their selection process. To provide clarity a slight re-organization of the GlobalPlatform Card specification has been done. Part V of the Open Platform 2.0.1' specification has been removed. Part of its content is incorporated into Part III and the rest is moved into the appendices.

1.5.2.7 Cardholder Verification Method Services

Additional services and features regarding the optional CVM have been included. In the previous versions of the Open Platform Card Specification, the CVM provided the possibility for a single PIN to be common across multiple Applications. When the PIN was changed by one Application it became visible to all Applications. However an Application could not check whether the PIN had previously been correctly presented to another Application during the same Card Session. The new Card Verification Method (CVM) services provide the ability to do this check.

1.5.2.8 Card Manager Separation

Historically the Card Manager has been viewed and defined as the major on-card component with no distinction or separation between the various responsibilities encompassed within as well as not clearly defining where the runtime environment ends and the Card Manager begins. A decision has been made to separate the Card Manager into 3 distinct entities and clearly identifying what runtime environment functionality must be within each. While the term Card Manager is still present, it now encompasses the OPEN, the Issuer Security Domain and the Cardholder Verification Method Services provider. This new structure clarifies the responsibilities of the Card Manager and takes into account both Java Card and Windows Powered Smart Card.

1.5.2.9 Windows Powered Smart Card API

The GlobalPlatform API for Windows Powered Smart Card is now included.

1.5.2.10 Java Card API

Taking into account the various new features of the GlobalPlatform a new API providing support for these new, and all relevant existing, features has been specified for Java Card. The previous Open Platform API for Java Card defined in version 2.0.1' is deprecated and remains in this version for backward compatibility.

1.5.2.11 Appendices

The body of the GlobalPlatform Card Specification only contains generic GlobalPlatform descriptions. All information specific to a particular implementation has been positioned in a set of appendices.

1.5.3 Revisions in GlobalPlatform Card Specification V2.1.1

The following modifications correct issues in the previous version and synchronize with recent evolutions of the underlying runtime environment specifications while maintaining full backwards compatibility. The minor editing changes and rewording for readability are not listed.

1.5.3.1 Errata

All intermediate published errata have been incorporated into this version. There is also a small set of errata and precisions that would have been published at around the same time this version is released, and these have also been incorporated directly into this version.

1.5.3.2 Content Removal

A new optional Card Content removal feature has been added that allows an Executable Load File and all its related Applications to be deleted in the same operation.

1.5.3.3 Logical Channels

To meet the emerging needs of some industries and recent evolutions of the underlying runtime environment specifications, logical channel functionality is added to this version of the specification as an optional feature.

1.5.3.4 Additional Secure Channel Protocol Implementation Options

An enhanced mechanism of generating a C-MAC has been added to both Secure Channel Protocol '01' and Secure Channel Protocol '02'. One new implementation option has been added for Secure Channel Protocol '01' and four new implementation options have been added for Secure Channel Protocol '02'. It is recommended that these new implementation options be used.

1.5.4 Major Adjustments in GlobalPlatform Card Specification V2.2

The following major adjustments are modifications decided by GlobalPlatform Members. All of these modifications are intended to make the GlobalPlatform more usable for a wider number of entities while maintaining full backward compatibility with previous versions.

The body of the document has been reorganized to reflect these changes, to arrange it by functions and components as well as to remove excessive duplication.

1.5.4.1 PKI functionality and Card Content Management

Card Content Management may now be performed by relying exclusively on asymmetric cryptography and a Public Key Infrastructure. This has resulted in the need to formalize the process of authentication and establishing ownership, so that the card can apply the relevant security and authorization rules for different off-card entities.

A delete token is added as an option, so that the Card Issuer may have a PK based policy to authorize card content removal.

1.5.4.2 Over-The Air Functionality and Inter-Application Communication

A mechanism for inter-application communication is formalized in terms of a general framework: Trusted Framework, which encompasses both the GlobalPlatform mechanism whereby an Application could receive its personalization data from its Security Domain as well as the SIM Toolkit and CAT frameworks defined for UICC cards by ETSI Project Smart Card Platform specifications. A new privilege - Trusted Path - between an on-card Receiving Entity and an on-card target Application is introduced.

1.5.4.3 Contactless Cards, Implicit Selection and Logical Channels

Support for contactless cards and dual interface cards (contact and contactless) is made more explicit in this version.

With the introduction of a new installation parameter, different Applications can now be implicitly selected on different card I/O interfaces: contact or contactless (and potentially others), and different logical channels.

The Default Selected privilege is redefined as the Card Reset privilege to modify the historical bytes. An Application is able to refuse explicit selection, e.g. because it does not support the current card I/O interface, and allow the (partial) selection process by OPEN to continue. To provide backward compatibility, the privilege confers implicit selectability if it has not been awarded to another Application.

As a further option, support of logical channels is expanded up to 19 supplementary logical channels as defined by the latest version of ISO/IEC 7816-4.

1.5.4.4 Secure Channel Protocols

A new Secure Channel Protocol based on asymmetric cryptography and a Public Key Infrastructure is introduced as Secure Channel Protocol '10'. Secure Channel Protocol '10' is compatible with CEN Workshop Agreement specification CWA 14890-1. It also fulfills the requirements of NICSS Framework Scheme defined by the Next Generation IC Card System Study Group (NICSS).

This version references the Secure Channel Protocol defined by ETSI Project Smart Card Platform TS 102 226 specification as GlobalPlatform Secure Channel Protocol '80'.

The number of recommended options for Secure Channel Protocol '02' is reduced to the most commonly used ones. The option indicator for Secure Channel Protocol '02' is defined as a bit-map and allows the support of any other option that was described in version 2.1.1 or Amendment A.

Secure Channel Protocol '01' is now deprecated.

The use of Security Levels associated with a secure channel session and an individual command-response pair have been made more explicit in this version. This version provides improved API support for Secure Channel Protocols. For example, an Application has the ability to increase the security level required for a (sequence of) individual command-response pair(s) using the GlobalPlatform API.

1.5.4.5 Global Services and CVMs

A general framework for dynamic management of card-wide services is introduced whereby one or more Global Services Applications may be present on a card to provide services to other Applications. The Global Services Applications are distinguished by having the Global Service privilege. A new installation parameter allows the on-card registration of service parameters. Service parameters are standardized by GlobalPlatform and can be registered as unique on the card by Global Services Applications using the GlobalPlatform API. GlobalPlatform API extensions allow Applications to request and, if authorized, use the services offered by Global Services Applications.

CVM functions are devolved from the OPEN to separate CVM Applications as Global Services Applications. Each CVM Application can handle one or multiple CVMs. GlobalPlatform API extensions allow CVM Applications to establish whether Applications have the authority to use and/or modify CVMs. Backward compatibility of the GlobalPlatform API is ensured for existing Applications using CVM services.

1.5.4.6 Security Domains, Privileges and Hierarchies

The privileges associated with Security Domains, in particular the Issuer Security Domain, are formalized so that access rights to Card Content Management functionality are more explicit. The Issuer Security Domain now has an explicit set of privileges, including new privileges such as Authorized Management or Token Verification. Other

new privileges are introduced to formalize the access rights awarded to Security Domains and Applications such as Global Registry Access, Global Lock and Global Delete.

Security Domain and Application privileges can now be modified dynamically on the card during their lifetime. CVM identifiers are standardized by GlobalPlatform.

Security Domains and OPEN itself can now have their Card Content Management functionality restricted dynamically during their Life Cycle through the use of a new INSTALL command.

Security Domains can now be associated with other Security Domains and so create a hierarchy of Security Domains. Association with Security Domains is made more systematic, so that a hierarchy of control can be established through association and extradition. Multiple hierarchies of Security Domains can be established, including by extraditing a Security Domain to itself and so making it the root of a new hierarchy. Access rights of Security Domains to Card Content Management functionality are expressed in regard to their association and hierarchy.

Card Content Management functionality has been extended to include explicit extradition of Executable Load Files.

The Token Verification privilege and the Receipt Generation privilege have been added.

1.5.4.7 On-Card APIs for Applications

A GlobalPlatform API expressed in the 'C' programming language for MULTOS™ cards is now included.

The GlobalPlatform API has been expanded and enhanced in order to support the new functionality introduced in this version.

References to Windows Powered Smart Card, and its specific API, have been deleted from this version.

The deprecated Open Platform Java Card API has also been removed from this version.

1.5.4.8 Key Management

New optional attributes for cryptographic keys are added to allow for more control, and for the partitioning of keys with different purposes. The new attributes are key usage and key access condition.

New key types are also added to support new cryptographic algorithms.

1.5.4.9 Amendment A

Amendment A to version 2.1.1 has been incorporated into this version. It comprises a set of optional extensions in support of GlobalPlatform Scripting Specification, EMV Card Personalization Specification and ETSI Project Smart Card Platform TS 102.225 and TS 102 226 specifications.

1.5.4.10 Errata and Precisions

All errata and precisions published since the release of version 2.1.1 and Amendment A have been incorporated into this version.

1.5.4.11 Other Major Changes

New installation parameters are added to further card memory management and allow memory reservation.

The Get Data command can now be used to retrieve a list of Applications present on the card.

It is now possible to lock, and subsequently unlock, a Security Domain and all its associated Applications in a single command.

It is now possible to load, install and make selectable an Application in a single combined process.

Part II
Architecture

2 System Architecture

Deploying a large number of chip cards based on dynamic, multi-application technology is not unlike deploying a very large number of workstations in a vast, semi-connected network. These card-based workstations support several different applications at any one time as well as allow for the possibility of updating or deleting those applications and installing new applications at any point in time.

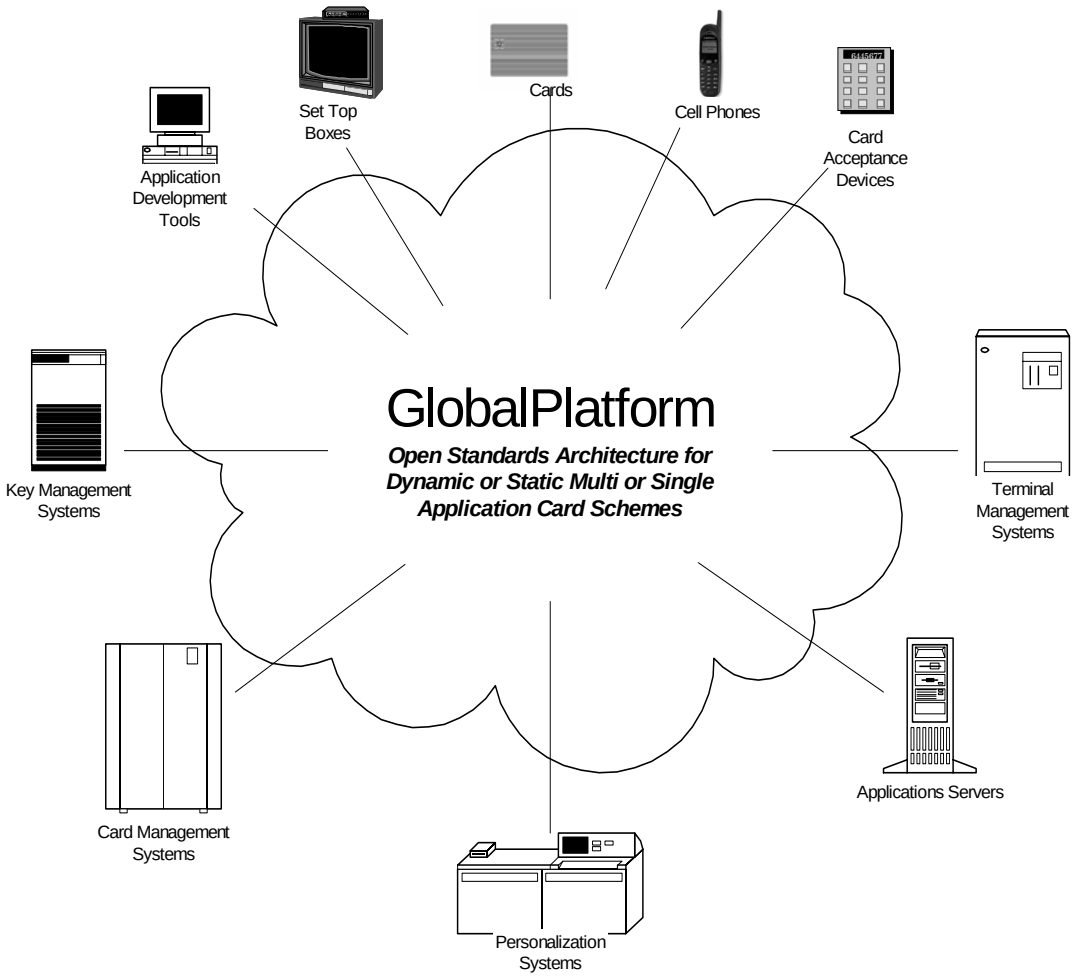

Figure 2-1: GlobalPlatform Architecture

The GlobalPlatform architecture is designed to provide Card Issuers with the system management architecture for managing these smart cards. Although GlobalPlatform is based on the paradigm that there is one single Card Issuer for a card, it offers to the Card Issuer the flexibility for managing an ever-changing array of business partners who may want to run applications on the Card Issuer's cards.

GlobalPlatform gives Card Issuers the power to manage their cards with the ultimate flexibility by enabling them to share control over part of their card with business partners. The ultimate control always rests with the Card Issuer, but through GlobalPlatform, the business partners of a Card Issuer can be allowed to manage their own Applications on the Card Issuer's cards as appropriate.

3 Card Architecture

The GlobalPlatform card architecture is comprised of a number of components that ensure hardware and vendor-neutral interfaces to Applications and off-card management systems. The following figure shows the components in a sample card configuration which includes one or more applications from the Card Issuer; one or more applications from one of the business partners of the Card Issuer, referred to as Application Providers; and one or more applications providing global services (e.g. CVM services) to other applications.

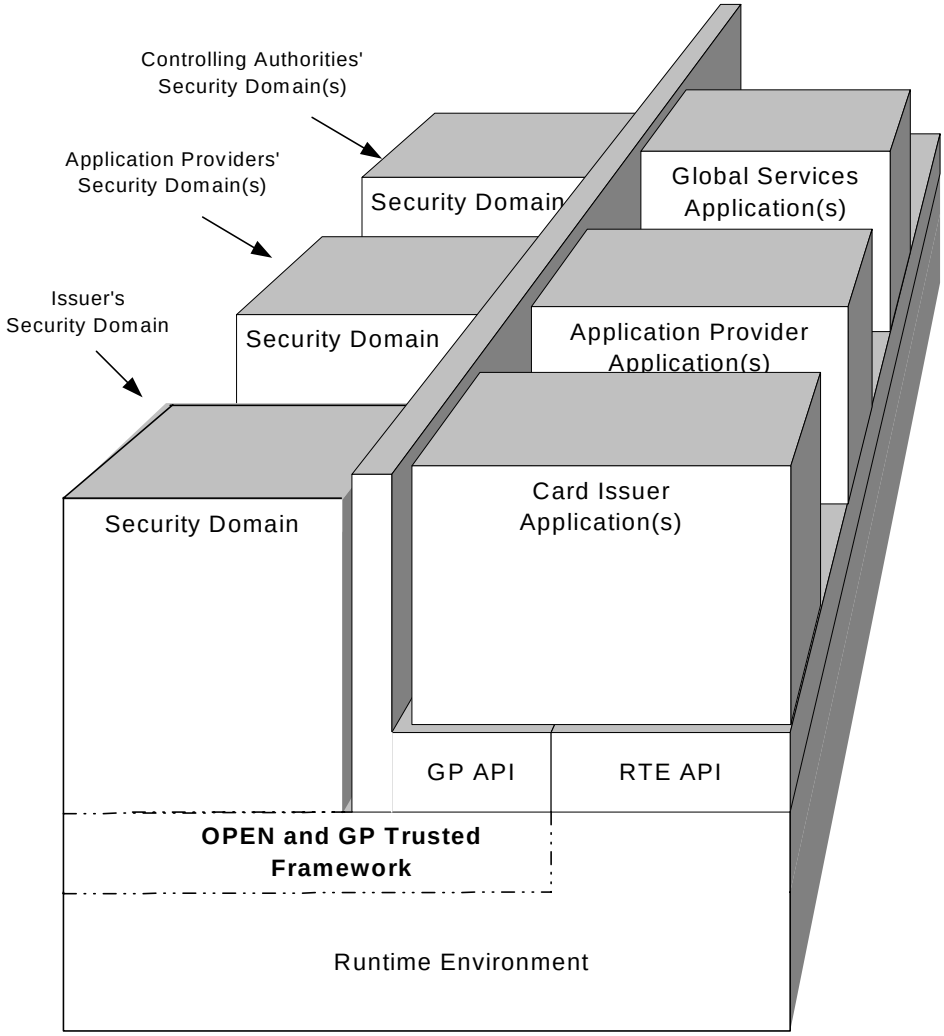

Figure 3-1: GlobalPlatform Card Architecture

All applications shall be implemented in a secure runtime environment that includes a hardware-neutral Application Programming Interface (API) to support application portability. GlobalPlatform does not mandate a specific runtime environment technology. The Card Manager is the primary GlobalPlatform card component that acts as the central administrator for a GlobalPlatform card. Special key and security management applications called Security Domains

are created to ensure complete separation of keys between the Card Issuer and multiple other Security Domain providers.

3.1 Security Domains

Security Domains act as the on-card representatives of off-card authorities. There are three main types of Security Domain, reflecting the three types of off-card authority recognized by a card:

- The Issuer Security Domain is the primary, mandatory on-card representative of the Card Administrator, typically the Card Issuer;
- Supplementary Security Domains are additional, optional on-card representatives of Application Providers or the Card Issuer; or their agents (e.g. service bureaus);
- Controlling Authority Security Domains are a special type of Supplementary Security Domain. A Controlling Authority may exist whose role is to enforce the security policy on all application code loaded to the card. If so, the Controlling Authority also uses this type of Security Domain as its on-card representative. There may be more than one such Security Domain.

In the main, all three types are referred to simply as Security Domains in this Specification;

Security Domains support security services such as key handling, encryption, decryption, digital signature generation and verification for their providers' (Card Issuer, Application Provider or Controlling Authority) applications.

Each Security Domain is established on behalf of a Card Issuer, an Application Provider or a Controlling Authority when these off-card entities require the use of keys that are completely isolated from each other.

3.2 Global Services Applications

One or more Global Services Applications may be present on the card to provide services to other Applications on the card. An example of such services are Cardholder Verification Method services.

3.3 Runtime Environment

The GlobalPlatform is intended to run on top of any secure, multi-application card runtime environment. This runtime environment is responsible for providing a hardware-neutral API for applications as well as a secure storage and execution space for applications to ensure that each application's code and data can remain separate and secure from other applications on the card. The card's runtime environment is also responsible for providing communication services between the card and off-card entities.

Cards should comply with appropriate standards: ISO/IEC 7816-3, ISO/IEC 7816-4, ISO/IEC 14443-3 and ISO/IEC 14443-4 in terms of announcing options supported in the ATR/ATQ such as the communications protocol, logical channels and command chaining.

3.4 Trusted Framework

GlobalPlatform cards may contain one or more Trusted Frameworks, which provide inter-application communication services between Applications. Trusted Frameworks are not Applications or Security Domains, but have a special status in that they are part of or extensions of the card's run-time environment. They should be assessed for security similarly to the runtime environment’s security assessment. See appendix G - *Trusted Framework Inter-Application Communication* for further details.

3.5 GlobalPlatform Environment (OPEN)

The main responsibilities of the GlobalPlatform Environment (OPEN) are to provide an API to applications, command dispatch, Application selection, (optional) logical channel management, and Card Content management. These functions shall be implemented by the OPEN if they are not provided by the runtime environment, or if provided by the runtime environment in a way not complying with this Specification.

The OPEN performs the application code loading and related Card Content management and memory management.

The OPEN also manages the installation of applications loaded to the card. The OPEN is responsible for enforcing security principles defined for Card Content management.

Another important function provided by the OPEN is APDU command dispatching and Application selection. When a SELECT command is successfully processed, the OPEN sets the Application referenced in the SELECT command to be the selected Application and subsequent Application commands shall be dispatched to the selected Application.

The availability of logical channels introduces an additional dimension to the card's architecture as multiple Applications may be selected concurrently. The OPEN shall rely on the runtime environment to control whether and when an individual Application can be selected concurrently with itself or another Application. When supporting logical channels, the OPEN shall allow for Applications that have no notion of logical channels as well as those that are multi-selectable. Support of logical channels is optional. Cards may support one or more (up to 19 according to ISO/IEC 7816-4) Supplementary Logical Channels.

The OPEN owns and uses an internal GlobalPlatform Registry as an information resource for Card Content management. The GlobalPlatform Registry contains information for managing the card, Executable Load Files, Applications, Security Domain associations, and privileges.

3.6 GlobalPlatform API

The GlobalPlatform API provides services to Applications (e.g. Cardholder verification, personalization, or security services). It also provides Card Content management services (e.g. card locking or Application Life Cycle State update) to Applications.

For the specification of the Application Programming Interface (API) on a Java Card™, see appendix A.1.

For the specification of the Application Programming Interface (API) on a MULTOS™ card, see appendix A.2.

3.7 Card Content

All Card Content, as defined in this specification, is first available on the card in the form of an Executable Load File. An Executable Load File can either exist in:

- Immutable Persistent Memory in which case it is loaded during the manufacturing stage and cannot be altered (except being disabled); or
- Mutable Persistent Memory in which case it can be loaded, or removed during Pre-Issuance or Post-Issuance.

Each Executable Load File may contain one or multiple Executable Modules, being application code. The installation of an Application creates an instance from an Executable Module plus possibly Application data within Mutable Persistent Memory. Any Application instance and its related data can be removed. A GlobalPlatform card is intended to support multiple Executable Load Files and multiple Executable Modules and as such multiple Applications may co-exist on an GlobalPlatform card. Note that the foregoing description assumes that Executable Modules will be present in the Executable Load File: however, their presence is optional and depends on the requirements of the Runtime Environment.

Figure 3-2 represents the relationship between an Executable Load File, an Executable Module (in the case where Executable Modules are present) and an Application.

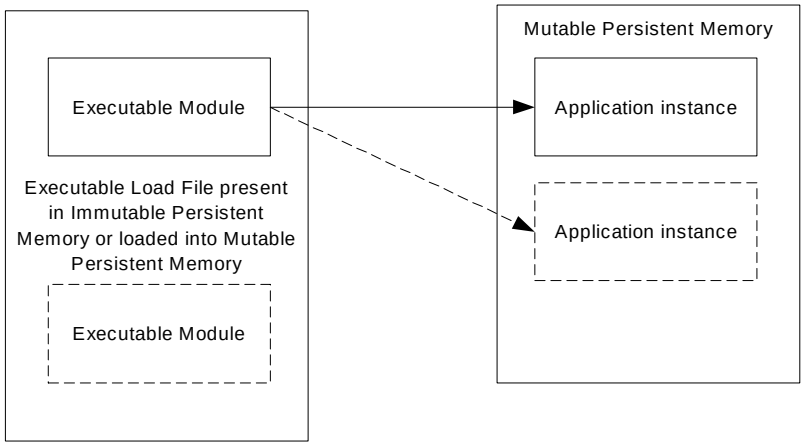

Figure 3-2 Card Content Relationships

3.8 Card Manager

The Card Manager, as the central administrator of the card, assumes multiple responsibilities.

The Card Manager can be viewed as three entities:

- The GlobalPlatform Environment (OPEN);
- The Issuer Security Domain; and
- Cardholder Verification Method Services.

4 Security Architecture

Well-designed security architectures are crucial to protecting the structure and function of cards within the GlobalPlatform system.

This chapter outlines:

- The security goals behind the architecture;
- The specific responsibilities of the Card Issuer as the owner of the card;
- The Application Providers as the owners of the Applications;
- The Controlling Authority;
- The security requirements for the on-card components;
- The cryptographic support provided by GlobalPlatform

4.1 Goals

The primary goal of the GlobalPlatform is to ensure the security and integrity of the card's components for the life of the card. These components are

- The runtime environment;
- The OPEN;
- The Issuer Security Domain;
- Supplementary Security Domains;
- The Applications.

To ensure card security and integrity, the GlobalPlatform is designed to support a range of secure mechanisms for:

- Data integrity;
- Resource availability;
- Confidentiality;
- Authentication.

The choice of security policy and cryptography is assumed to be industry and product specific.

Because the cards are only part of a larger card system involving multiple parties and off-card components, the GlobalPlatform also relies upon non-cryptographic, procedural means of protection, such as code testing and verification, physical security, and secure key handling. However, these aspects are out of scope for this card specification.

4.2 Security Responsibilities and Requirements

4.2.1 Card Issuer's Security Responsibilities

The Card Issuer is responsible for:

- Generating and loading the Issuer Security Domain keys;

- Enforcing standards and policies for Application Providers governing all aspects of Applications to be provided to the Card Issuer or operated on the Card Issuer's cards;
- Working with Application Providers to create and initialize Security Domains other than the Issuer Security Domain;
- Determining policy with regards to card and Application Life Cycle management, velocity checking levels, privileges, and other security parameters;
- Managing the application code loading and installing both on a Pre-Issuance and Post-Issuance basis, and
- Cryptographically authorizing load, install, and extradition to be performed by Application Providers.

4.2.2 Application Provider's Security Responsibilities

The Application Provider is responsible for:

- Generating the keys for its own Security Domains or obtaining Security Domain keys from a trusted third party;
- Working with the Card Issuer to load generated keys into the Application Provider's Security Domain;
- Providing applications that meet the Card Issuer's security standards and policies;
- Providing load file data block signatures according to its own security policy for integrity and source authenticity;
- Obtaining pre-authorization for load, install, and extradition from the Card Issuer;
- Returning receipts for load, install, delete, and extradition, according to the Card Issuer's policy.

4.2.3 Controlling Authority's Security Responsibilities

A Controlling Authority is responsible for:

- Generating the keys for its own Security Domain or obtaining Security Domain keys from a trusted third party;
- Working with the Card Issuer to load generated keys into the Controlling Authority's Security Domain;
- Providing load file data block signatures according to its own security policy for integrity and source authenticity.

4.2.4 On-Card Components' Security Requirements

4.2.4.1 Runtime Environment Security Requirements

The runtime environment is responsible for:

- Providing an interface to all Applications that ensures that the runtime environment security mechanisms cannot be bypassed, deactivated, corrupted or otherwise circumvented;
- Performing secure memory management to ensure that:
  - Each application's code and data (including transient session data) as well as the runtime environment itself and its data (including transient session data) is protected from unauthorized access from within the card;

  - When more than one logical channel is supported, each concurrently selected Application's code and data (including transient session data) as well as the runtime environment itself and its data (including transient session data) is protected from unauthorized access from within the card;
  - The previous contents of the memory is not accessible when that memory is reused;
  - The memory recovery process is secure and consistent in case of a loss of power or withdrawal of the card from the card reader while an operation is in progress;
- Providing communication services with off-card entities that ensures the proper transmission (according to the specific communication protocol rules) of unaltered command and response messages.

(See the appropriate runtime environment documentation for more details).

4.2.4.2 Trusted Framework Requirements

Each Trusted Framework present on the card shall:

- Check the application access rules of the inter-acting Applications according to their respective privileges;
- Enforce the Trusted Framework security rules for inter-application communication, including the rules; defined in appendix G;
- Ensure that incoming messages are properly routed unaltered to their intended destinations;
- Ensure that any response messages are properly returned unaltered (except for any cryptographic protection) to the original receiver of the incoming message.

4.2.4.3 OPEN Security Requirements

The OPEN shall:

- Provide an interface to all Applications that ensures that the GlobalPlatform security mechanism cannot be bypassed, deactivated, corrupted or otherwise circumvented;
- Check application access rules according to the Applications' privileges;
- Manage card and Application Life Cycle (see chapter 5 - *Life Cycle Models*);
- Ensure that the Card Content changes are authorized by the Card Issuer;
- Ensure that application code has been signed by the Controlling Authority represented on the card;
- Ensure that application code has been signed by Application Providers represented on the card, if required.

4.2.4.4 Security Domain Security Requirements

Security Domains enforce the security policies of their off-card Security Domain Provider.

When applicable a Security Domain shall:

- Communicate with off-card entities in accordance with its Security Domain Provider's security policy in Pre-Issuance and Post-Issuance;
- Manage on-card data securely;
- Provide cryptographic protection services for its own Applications during their personalization and optionally during their subsequent operation;
- Request the OPEN to load, install, extradite, and delete card content;
- Return to the off-card entity any receipt for load, install, extradition, and delete;

- Verify the authorization for Card Content changes initiated by an off-card authority;
- Generate receipts for load, install, extradition, and delete;
- Verify the load file data block signature when requested by the OPEN.

4.2.4.5 Global Services Application Security Requirements

A Global Services Application shall:

- Be able to provide services to other Applications, such as CVM services;
- Hold the Global Services application-related data securely;
- Perform internal security measures as required by the service.

4.2.4.6 Application Security Requirements

Applications should:

- Expose only data and resources that are necessary for proper application functionality and;
- Perform internal security measures required by the Application Provider.

4.2.5 Back-End System Security Requirements

Despite the best efforts of the card and the loading processes to provide a stable and secure environment, these components alone cannot ensure total security. The back-end systems (multiple back-end systems may exist for a single card), which communicate with the cards, perform the verifications, and manage the off-card key databases, also shall be trusted. Responsible personnel, secure operating systems, system security policies, and audit procedures are all essential components that secure the back-end systems. These requirements are beyond the scope of this Specification. Information on GlobalPlatform's off-card requirements relating to card management can be found in the GlobalPlatform Key Management System Functional Requirements, GlobalPlatform Smart Card Management System Functional Requirements and GlobalPlatform Messaging Specification.

4.3 Cryptographic support

One of the major requirements for a GlobalPlatform card is the ability to provide a minimum level of cryptographic functionality. This cryptography is, for example, used for the generation of signatures, and is available for use by the Applications present on the card.

The Issuer Security Domain shall implement one Secure Channel Protocol. A Security Domain other than the Issuer Security Domain shall implement [at least] one Secure Channel Protocol. A GlobalPlatform card should support symmetric cryptography such as the Data Encryption Standard (DES) algorithm. A GlobalPlatform card may also support asymmetric cryptography such as the Rivest / Shamir / Adleman (RSA) algorithm.

The following cryptographic services are described in this section:

- Integrity and authentication;
- Secure messaging.

When present, services to encrypt and decrypt any pattern of data using these algorithms shall be available to Applications.

It is the responsibility of the Card Issuer or the Controlling Authority to set up the appropriate off-card procedures to comply with the governmental restrictions upon cryptography. Features to disable or restrict cryptography usage by Applications on a card are beyond the scope of this Specification.

4.3.1 Secure Card Content Management

The concepts of integrity and authentication represent an additional value associated with a message or a block of data.

The purpose of this additional value is to provide a method of verifying the source and/or the integrity of particular block of code or data.

The following describes the different usages of integrity and authentication for Card Content management in this Specification.

4.3.1.1 Load File Data Block Hash

The Load File Data Block Hash is intended to verify the integrity of a complete Load File Data Block when loaded to a GlobalPlatform card.

The Load File Data Block Hash is used in the computation of:

- The Load File Data Block Signature (see section 4.3.1.2 - *Load File Data Block Signature*); and
- The Load Token (see section 4.3.1.3 - *Delegated Management Tokens*).

4.3.1.2 Load File Data Block Signature (DAP)

The Load File Data Block Signature is an authentication value generated by an off-card entity (an Application Provider or a Controlling Authority). This is the signature of the Load File Data Block Hash and is included in the DAP Block of the Load File. One or more DAP Blocks may be included in a Load File.

When present during the loading of a Load File to the card, each signature shall be verified by the appropriate Security Domain. The verification operation is referred to as Data Authentication Pattern (DAP) Verification.

4.3.1.3 Delegated Management Tokens

Delegated Management Tokens are signatures of one or more Delegated Management functions (loading, installing, extraditing and deleting) generated by the Card Issuer and used to provide the Card Issuer the control over these Card Content changes. Tokens shall be verified by the appropriate Security Domain.

4.3.1.4 Receipts

The appropriate Security Domain may generate Receipts during Delegated Management. A Receipt is proof that an Application Provider has modified the Card Content.

4.3.2 Secure Communication

A GlobalPlatform card may provide security services related to information exchanged between the card and an off-card entity. The security level of the communication with an off-card entity does not necessarily apply to each individual message being transmitted but can only apply to the environment and/or context in which messages are transmitted. The concept of the Life Cycle of the card (see section 5.1 - *Card Life Cycle*) may be used to determine the security level of the communication between the card and an off-card entity.

The choice of cryptographic algorithms for secure communication is assumed to be industry and product specific.

A GlobalPlatform card offers the following security services associated with messages and defined within a Secure Channel Protocol (see chapter 10 - *Secure Communication*):

- **Entity authentication** - in which the card or the off-card entity proves its authenticity to the other entity through a cryptographic exchange;

- **Integrity and authentication** - in which the receiving entity (the card or off-card entity) ensures that the data being received from the sending entity (respectively the off-card entity or card) actually came from an authenticated entity in the correct sequence and has not been altered;
- **Confidentiality** - in which data being transmitted from the sending entity (the off-card entity or card) to the receiving entity (respectively the card or off-card entity) is not viewable by an unauthenticated entity.

Authentication of the off-card entity combined with the card Life Cycle State allows the card to assume its environment and/or context.

Part III
Implementation

5 Life Cycle Models

The GlobalPlatform defines Life Cycle models to control the functionality and security of the following GlobalPlatform components:

- Card;
- Executable Load Files;
- Executable Modules;
- Security Domains;
- Applications.

The OPEN owns and maintains the Life Cycle information within the GlobalPlatform Registry and manages the requested state transitions.

The Life Cycle models of each component are presented in this chapter.

5.1 Card Life Cycle

The OPEN is responsible for maintaining the overall security and administration of the card and its content. As the OPEN plays this supervisory role over the entire card, its life cycle can be thought of as the life cycle of the card and is referred to as the card Life Cycle in the subsequent sections.

From a GlobalPlatform perspective, the card Life Cycle begins with the state OP_READY. Although a cards life includes activities prior to the initial card Life Cycle State, these activities are considered card implementation specific and are beyond the scope of this Specification.

The end of the card Life Cycle is the state TERMINATED.

The Issuer Security Domain inherits the card Life Cycle State.

5.1.1 Card Life Cycle States

The following card Life Cycle States shall apply:

- OP_READY
- INITIALIZED
- SECURED
- CARD_LOCKED
- TERMINATED

The card Life Cycle States OP_READY and INITIALIZED are intended for use during the Pre-Issuance phases of the card's life.

The states SECURED, CARD_LOCKED and TERMINATED are intended for use during the Post-Issuance phase of the card although it is possible to terminate the card at any point during its life.

5.1.1.1 Card Life Cycle State OP_READY

The state OP_READY indicates that the runtime environment shall be available and the Issuer Security Domain, acting as the selected Application, shall be ready to receive, execute and respond to APDU commands.

The following functionality shall be present when the card is in the state OP_READY:

- The runtime environment shall be ready for execution;
- The OPEN shall be ready for execution;
- The Issuer Security Domain shall be the implicitly selected Application for all card interfaces;
- Executable Load Files that were included in Immutable Persistent Memory shall be registered in the GlobalPlatform Registry;
- An initial key shall be available within the Issuer Security Domain.

The card shall be capable of Card Content changes, the loading of the Load Files containing applications not already present in the card may occur.

The installation, from Executable Load Files, of any Application may occur.

Additionally, if any personalization information is available at this stage, Applications may be personalized.

The OP_READY state may be used by an off-card entity to perform the following actions:

- Supplementary Security Domains may be loaded and/or installed;
- The Security Domain keys may be inserted in order to maintain a cryptographic key separation from the Issuer Security Domain keys.

5.1.1.2 Card Life Cycle State INITIALIZED

The state INITIALIZED is an administrative card production state. The state transition from OP_READY to INITIALIZED is irreversible. Its functionality is beyond the scope of this Specification. This state may be used to indicate that some initial data has been populated (e.g. Issuer Security Domain keys and/or data) but that the card is not yet ready to be issued to the Cardholder.

5.1.1.3 Card Life Cycle State SECURED

The state SECURED is the intended operating card Life Cycle State in Post-Issuance. This state may be used by Security Domains and Applications to enforce their respective security policies. The state transition from INITIALIZED to SECURED is irreversible.

The SECURED state should be used to indicate to off-card entities that the Issuer Security Domain contains all necessary keys and security elements for full functionality.

5.1.1.4 Card Life Cycle State CARD_LOCKED

The card Life Cycle state CARD_LOCKED is present to provide the capability to disable the selection of Security Domain and Applications. The card Life Cycle state transition from SECURED to CARD_LOCKED is reversible.

Setting the card to this state means that the card shall only allow selection of the application with the Final Application privilege.

Card Content changes including any type of data management (specifically Security Domain keys and data) are not allowed in this state.

Either the OPEN, or a Security Domain with Card Lock privilege, or an Application with Card Lock privilege (see section 6.6 - *Privileges*), may initiate the transition from the state SECURED to the state CARD_LOCKED.

5.1.1.5 Card Life Cycle State TERMINATED

The state TERMINATED signals the end of the card Life Cycle and the card. The state transition from any other state to TERMINATED is irreversible.

The state TERMINATED shall be used to permanently disable all card functionality with respect to any card content management and any life cycle changes. This card state is intended as a mechanism for an Application to logically 'destroy' the card for such reasons as the detection of a severe security threat or expiration of the card. If a Security Domain has the Final Application privilege only the GET DATA command shall be processed, all other commands defined in this specification shall be disabled and shall return an error. If an application has the Final Application privilege its command processing is subject to issuer policy.

The OPEN itself, or a Security Domain with Card Terminate privilege, or an Application with Card Terminate privilege (see section 6.6 - *Privileges*), may initiate the transition from any of the previous states to the state TERMINATED.

5.1.2 Card Life Cycle State Transitions

Figure 5-1 illustrates the state transition diagram for the card Life Cycle. This can typically be viewed as a sequential process with certain possibilities for reversing a state transition or skipping states.

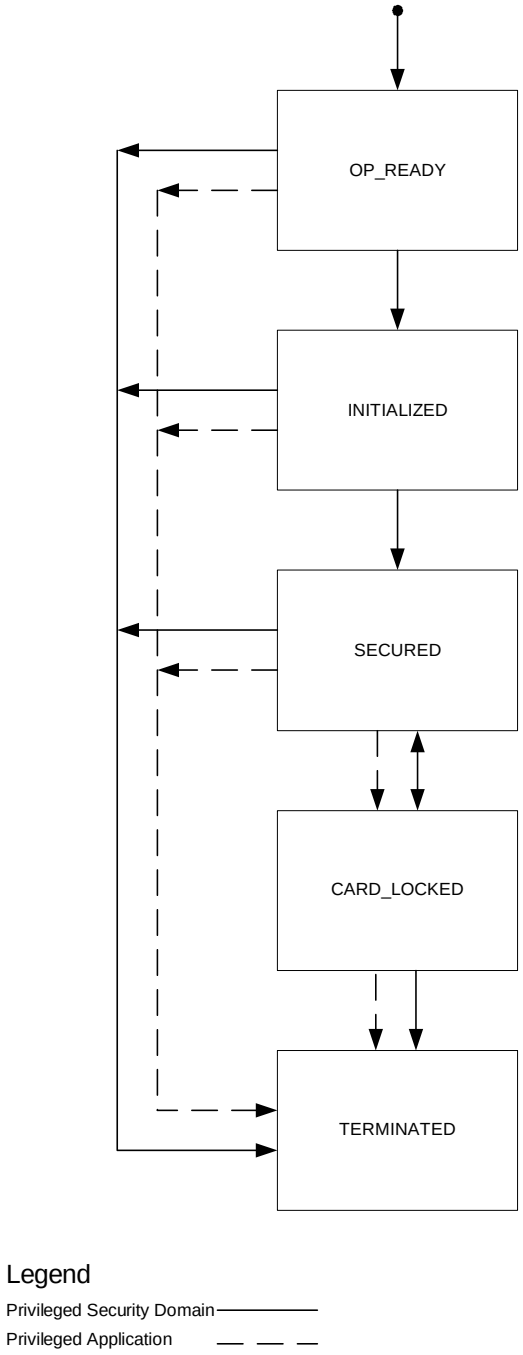

Figure 5-1: Card Life Cycle State Transitions

5.2 Executable Load File/ Executable Module Life Cycle

An Executable Load File is the actual on-card container of one or more application's executable code (Executable Modules). It may reside in Immutable Persistent Memory or may be created in Mutable Persistent Memory as the resulting image of a Load File Data Block. The format in which the Executable Load File is stored on the card is beyond the scope of this Specification.

The OPEN owns and maintains the Executable Load File Life Cycle information within the GlobalPlatform Registry.

5.2.1 Executable Load File Life Cycle

The Executable Load File Life Cycle can only have one state.

5.2.1.1 Executable Load Life Cycle LOADED

The OPEN shall consider all Executable Load Files present in the card in Immutable Persistent Memory or Mutable Persistent Memory to be in the state LOADED. An Executable Load File transferred to the card through a Load File shall become an entry in the GlobalPlatform Registry following the successful completion of the load process. Executable Load Files present in Immutable Persistent Memory shall automatically have entries within the GlobalPlatform Registry and initially be associated with the Issuer's Security Domain.

5.2.1.2 Executable Load File Deletion

The OPEN may receive a request to delete an Executable Load File. If the Executable Load File cannot be physically deleted (e.g., because it is stored in Immutable Persistent Memory), the following behavior shall apply except that the actual space cannot be reclaimed.

The space previously used to store a physically deleted Executable Load File is reclaimed and may be reused. The entries within the GlobalPlatform Registry of the Executable Load File and each Executable Module within the Executable Load File shall no longer be available, and the OPEN is not required to maintain a record of the deleted Executable Load File's or Executable Module's previous existence.

If the received request is also intended to delete each of the Applications instantiated from the Executable Modules within this Executable Load File, then for each of these Applications the behavior described in section 5.3.1.4 - *Application Deletion* or section 5.3.2.5 - *Security Domain Deletion* shall occur.

5.2.2 Executable Module Life Cycle

The Executable Module Life Cycle is linked to the Executable Load File Life Cycle.

5.3 Application and Security Domain Life Cycle

The Life Cycle of the Application or Security Domain begins when the application is instantiated from an Executable Module. The Life Cycle reflects states that are controlled by the OPEN and states that are controlled directly by the Application.

The Application becomes an entry in the GlobalPlatform Registry and the OPEN sets the Application Life Cycle State to the initial state of INSTALLED during the Application installation process. The OPEN is also responsible for making the Application available for selection by setting its Life Cycle State to SELECTABLE upon request during the Application installation process.

Once an Application or Security Domain is available for selection, it takes control of managing its own Life Cycle. The definition of these state transitions is Application or Security Domain dependent and not controlled by the OPEN.

At any point in the Application or Security Domain Life Cycle, the OPEN may take control for security protection by setting the Life Cycle State to LOCKED. The OPEN also controls the deletion of an Application from the card.

5.3.1 Application Life Cycle States

This Specification defines the following Application Life Cycle States:

INSTALLED

SELECTABLE

LOCKED

In addition to these Application Life Cycle States, the Application may define its own Application dependent states.

Once the Application reaches the SELECTABLE state, it is responsible for managing the next steps of its own Life Cycle. It may use any Application specific states as long as these do not conflict with the states already defined by GlobalPlatform. The OPEN may not perform these transitions without instruction from the Application and the Application is responsible for defining state transitions and ensuring that these transitioning rules are respected.

5.3.1.1 Application Life Cycle State INSTALLED

The state INSTALLED means that the Application executable code has been properly linked and that any necessary memory allocation has taken place. The Application becomes an entry in the GlobalPlatform Registry and this entry is accessible to off-card entities authenticated by the associated Security Domain. The Application is not yet selectable. The installation process is not intended to incorporate personalization of the Application, which may occur as a separate step.

5.3.1.2 Application Life Cycle State SELECTABLE

The state SELECTABLE means that the Application is able to receive commands from off-card entities. The state transition from INSTALLED to SELECTABLE is irreversible. The Application shall be properly installed and functional before it may be set to the state SELECTABLE. The transition to SELECTABLE may be combined with the Application installation process.

The behavior of the Application in the state SELECTABLE is beyond the scope of this Specification.

5.3.1.3 Application Life Cycle State LOCKED

The OPEN, the Application itself, the Application's associated Security Domain, an Application with the Global Lock privilege or a Security Domain with the Global Lock privilege uses the state LOCKED as a security management control to prevent the selection, and therefore the execution, of the Application.

If the OPEN detects a threat from within the card and determines that the threat is associated with a particular Application, that Application may be prevented from further selection by the OPEN setting the state to LOCKED.

Alternatively, the off-card entity may determine that a particular Application on the card needs to be locked for a business or security reason and may initiate the Application Life Cycle transition via the OPEN.

Once the state is LOCKED, only the Application's associated Security Domain, an Application with Global Lock privilege or a Security Domain with Global Lock privilege is allowed to unlock the Application. The OPEN shall ensure that the Application Life Cycle returns to its previous state.

5.3.1.4 Application Deletion

At any point in the Application Life Cycle, the OPEN may receive a request to delete an Application.

The space previously used to store a physically deleted Application is reclaimed and may be reused. The entry within the GlobalPlatform Registry shall no longer be available, and the OPEN is not required to maintain a record of the deleted Application's previous existence.

5.3.1.5 Application Specific Life Cycle States

These states are Application specific. The behavior of the Application, while in these states, is determined by the Application itself and is beyond the scope of this Specification. The OPEN does not enforce any control on Application specific Life Cycle State transitions.

5.3.1.6 Application Life Cycle State Transitions

Figure 5-2 illustrates the state transition diagram for the Application Life Cycle. This can typically be viewed as a sequential process with certain possibilities for reversing a state transition or skipping states.

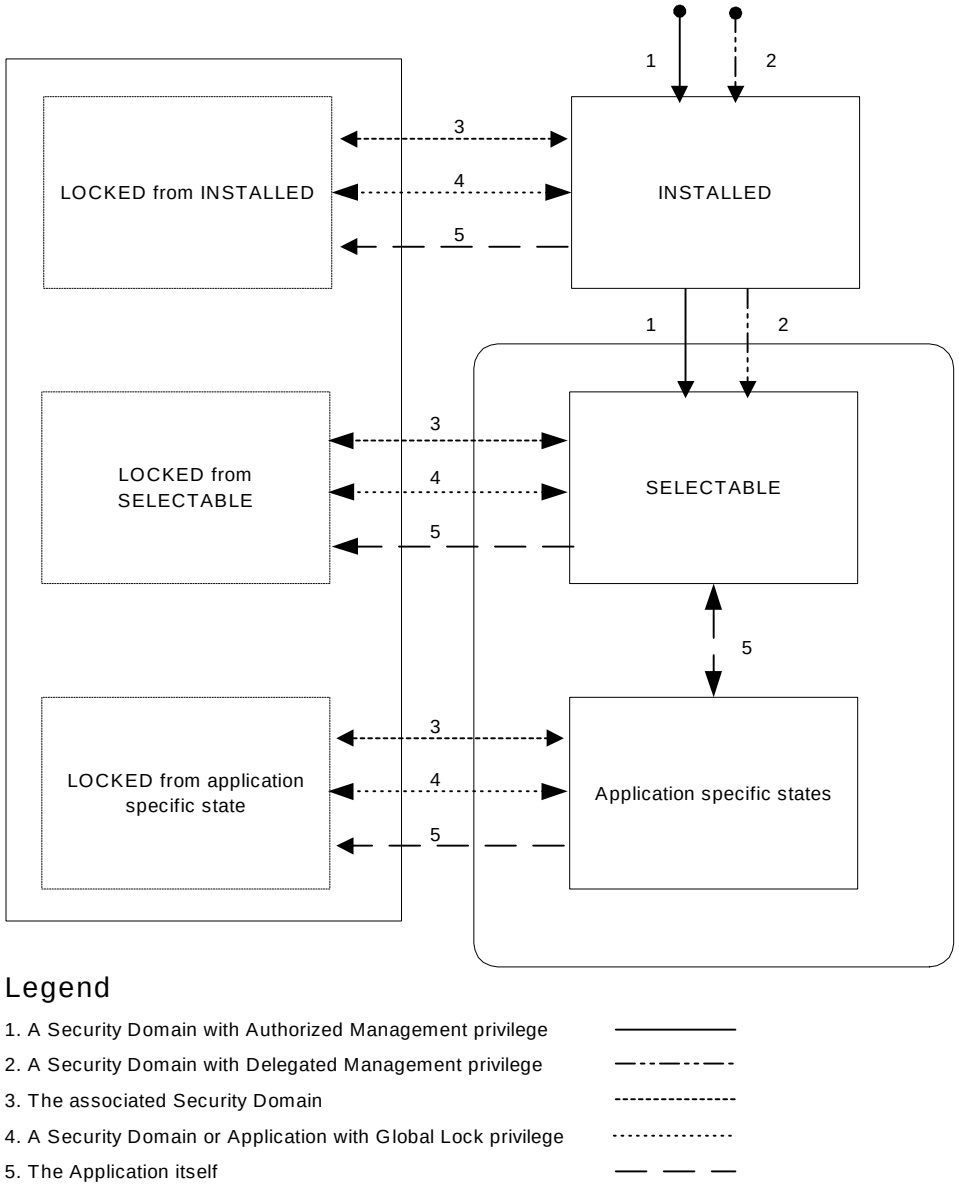

Figure 5-2: Application Life Cycle State Transitions

5.3.2 Security Domain Life Cycle States

This Specification defines the following states applicable to a Security Domain:

1. INSTALLED
2. SELECTABLE
3. PERSONALIZED

4. LOCKED

There are no proprietary Security Domain Life Cycle States.

5.3.2.1 Security Domain Life Cycle State INSTALLED

The state INSTALLED means that the Security Domain becomes an entry in the GlobalPlatform Registry and this entry is accessible to off-card entities authenticated by the associated Security Domain. The Security Domain is not yet available for selection. It cannot be associated with Executable Load Files or Applications yet and therefore its Security Domain services are not available to Applications.

5.3.2.2 Security Domain Life Cycle State SELECTABLE

The state SELECTABLE means that the Security Domain is able to receive commands (specifically personalization commands) from off-card entities. As they still do not have keys, the Security Domains cannot be associated with Executable Load Files or Applications and therefore their services are not available to Applications when they are in this state. The state transition from INSTALLED to SELECTABLE is irreversible. The transition to SELECTABLE may be combined with the Security Domain installation process.

5.3.2.3 Security Domain Life Cycle State PERSONALIZED

The definition of what is required for a Security Domain to transition to the state PERSONALIZED is Security Domain dependent but is intended to indicate that the Security Domain has all the necessary personalization data and keys for full runtime functionality (i.e. usable in its intended environment). The transition from SELECTABLE to PERSONALIZED (initiated by the Security Domain itself) is irreversible.

In the state PERSONALIZED, the Security Domain may be associated with Applications and its services become available to these associated Applications.

5.3.2.4 Security Domain Life Cycle State LOCKED

The OPEN, the Security Domain itself, the Security Domain's associated Security Domain (if any), an Application with the Global Lock privilege or a Security Domain with the Global Lock privilege uses the state LOCKED as a security management control to prevent the selection of the Security Domain.

If the OPEN detects a threat from within the card and determines that the threat is associated with a particular Security Domain, that Security Domain may be prevented from further selection by the OPEN setting the Security Domain's Life Cycle State to LOCKED.

Alternatively, the off-card entity may determine that a particular Security Domain on the card needs to be locked for a business or security reason and may initiate the state transition via the OPEN.

In this state, the Security Domain is prevented from being used for Delegated Management if applicable. Locking a Security Domain prevents this Security Domain from being associated with new Executable Load Files or Applications. In this state DAP verification, extradition and access to that Security Domain’s services shall fail.

Once the Life Cycle State is LOCKED, only the Security Domain's associated Security Domain (if any), an Application with Global Lock privilege or a Security Domain with Global Lock privilege is allowed to unlock the Security Domain. The OPEN shall ensure that the Security Domain's Life Cycle returns to its previous state.

5.3.2.5 Security Domain Deletion

At any point in the Security Domain Life Cycle, the OPEN may receive a request to delete a Security Domain.

The space previously used to store a physically deleted Security Domain is reclaimed and may be reused. The entry within the GlobalPlatform Registry shall no longer be available, and the OPEN is not required to maintain a record of the deleted Security Domain's previous existence.

5.3.2.6 Security Domain Life Cycle State Transitions

Figure 5-3 illustrates the state transition diagram for the Security Domain Life Cycle. This can typically be viewed as a sequential process with certain possibilities for reversing a state transition or skipping states.

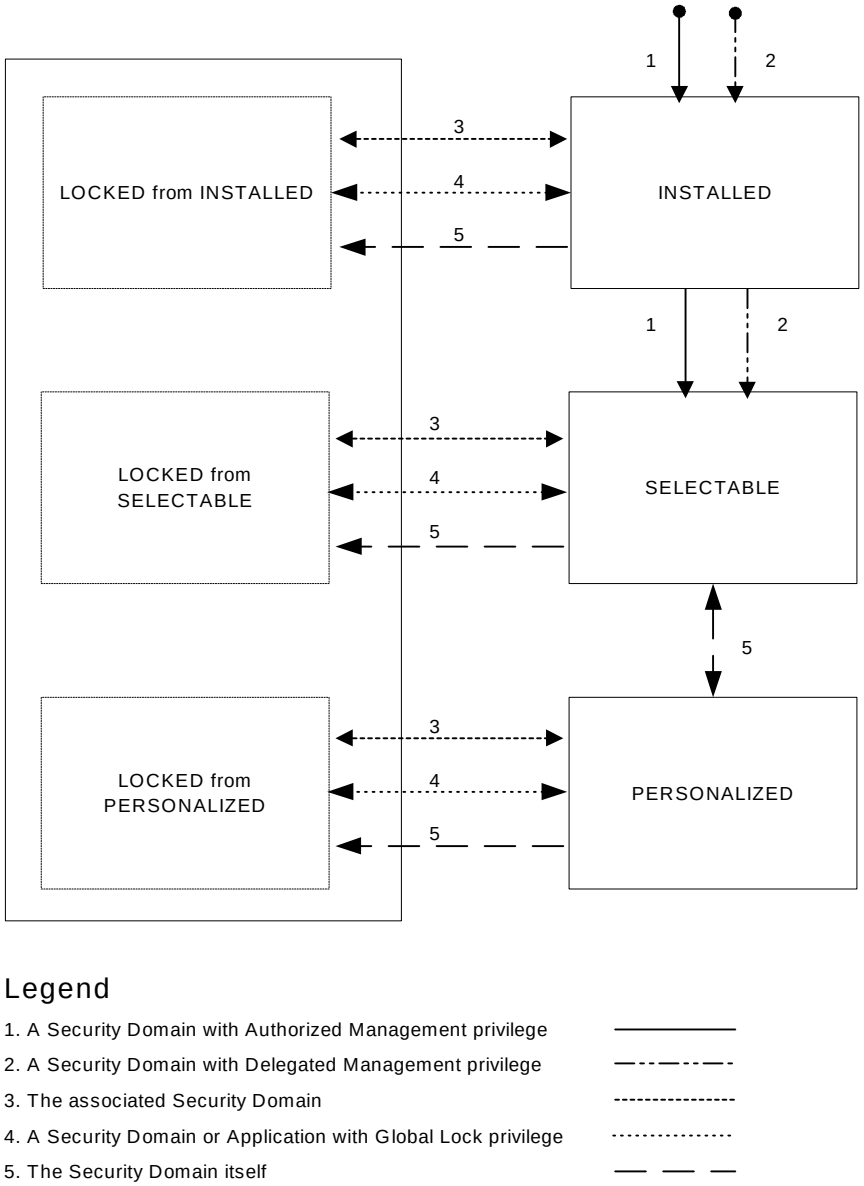

Figure 5-3: Security Domain Life Cycle State Transitions

5.4 Sample Life Cycle Illustration

This section provides a description of a sample GlobalPlatform card and its Life Cycle transitions from the card's creation to the time it is terminated. It also shows the status of several Executable Load Files, Executable Modules and Applications and their relationship with the card Life Cycle. Figure 5-4 illustrates these sample Life Cycle States:

Application A: Application code is present as an Executable Module within an Executable Load File in Mutable Persistent Memory when the card is manufactured. It is installed in an implementation specific manner. It is used throughout the card's life until the card is terminated and as long as the card is not in a CARD_LOCKED state.

Application B: Application code is present as an Executable Module within an Executable Load File in Immutable Persistent Memory when the chip is manufactured. It is installed prior to the card being initialized. Application B, along with its Executable Load File, is deleted some time during the card's life before the card is terminated. Because the Executable Load File for Application B is stored in Immutable Persistent Memory, it cannot be physically deleted from the card.

Application C: Application code is present as an Executable Module within an Executable Load File and is loaded in an implementation specific manner. The Application is installed in Post-Issuance while the card is in the Life Cycle State SECURED. The Application is used for some time and then deleted along with its Executable Load File before the card is terminated. Because the Application and its Executable Load File are stored in Mutable Persistent Memory, the Application and associated Executable Load File, along with all its Executable Modules, are purged from Mutable Persistent Memory and the memory space reclaimed for reuse.

Application D: Application code is present as an Executable Module within an Executable Load File and is loaded in an implementation specific manner. It is used during the full lifetime of the card until the card is terminated and as long as the card Life Cycle State is not CARD_LOCKED.

Application E: Application code is present as an Executable Module within an Executable Load File that is loaded and installed in Post-Issuance while in the card Life Cycle State SECURED. Application E is used until the card is terminated and as long as the card Life Cycle State is not CARD_LOCKED.

Application F: Application code is present as an Executable Module within the same Executable Load File as Application E. It is loaded and installed in Post-Issuance while in the card Life Cycle State SECURED. Application F is deleted some time during the card's lifetime.

Note that the following diagram is not intended to be a comprehensive description of what is permitted in each card Life Cycle State.

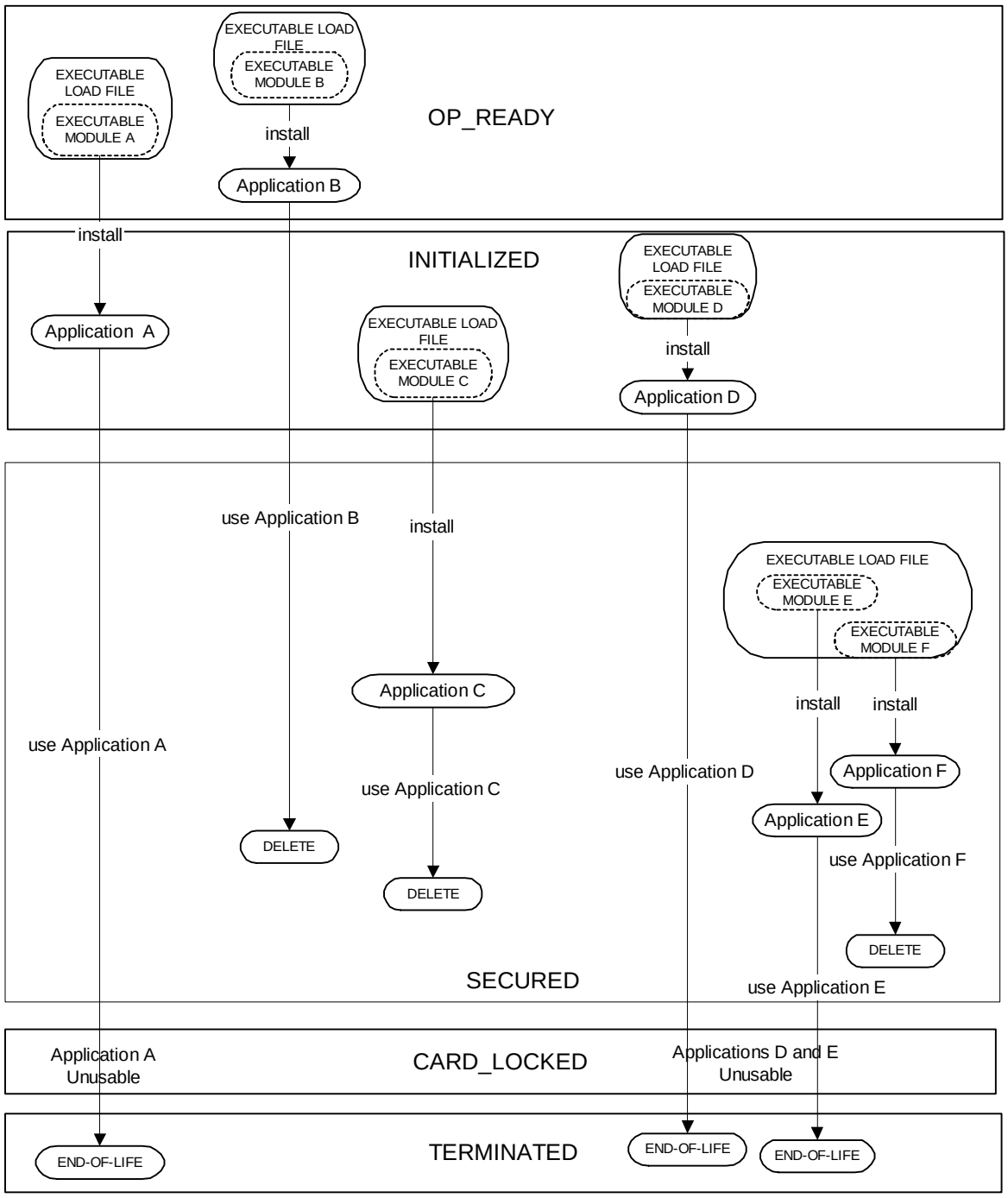

Figure 5-4: Sample Card Life Cycle and Application Life Cycles

6 GlobalPlatform Environment (OPEN)

6.1 Overview

The GlobalPlatform Environment (OPEN) supports the following functions if the underlying runtime environment does not support them:

- **Command Dispatch**

  Application and Security Domain selection

  (Optional) Logical channel management

  Command dispatching

- **Card Content Management**

  Content verification

  Content loading

  Content installation

  Content removal

  Access control rules for card content management

- **Security Management**

  Security Domain locking and unlocking

  Application locking and unlocking

  Card locking and unlocking

  Card termination

  Privilege usage

  Security Domain privilege usage

  Tracing and event logging

- **GlobalPlatform Trusted Framework**

  Secure inter-application communication

The OPEN architecture may be illustrated as a collection of system functions built upon a GlobalPlatform Registry. The GlobalPlatform Registry is a data store required to support the various system functions of the GlobalPlatform.

The following are some examples of the use of the GlobalPlatform Registry by the OPEN.

Command Dispatch:

- Determine if an Application or Security Domain is present and available to respond to a SELECT command;
- When supported, manage logical channels;
- Dispatch commands to the selected Application for command processing.

Card Content Management:

- Store state and management information about newly loaded Executable Load Files, Executable Modules, and installed Applications;
- Store the Security Domain to be associated with Executable Load Files being loaded;
- Store the privileges and associated Security Domains of the Applications being installed;
- Identify an Application's associated Security Domain to provide access for that Application to its Security Domain services.

Security Management:

- Allow the audit of Card Content by retrieving status information related to any Application present on the card;
- Verify the integrity and request verification of the authenticity for Executable Load Files;
- Verify that resource restrictions are respected during loading and installing of new content and during Application runtime execution;
- Verify an Application's or a Security Domain's accessibility to functionality that requires privileges.

6.2 OPEN Services

Appendix A.1 - *GlobalPlatform on a Java Card* provides the Java Card™ implementation of the following interfaces.

Appendix A.2 - *GlobalPlatform on MULTOS*™ provides the MULTOS™ implementation of the following interfaces.

Applications and Security Domains may access and/or modify some content known or managed by the OPEN. Depending on the relevant privileges of the requesting entity, the following services shall be provided by the OPEN:

- Retrieving the Application's own Life Cycle State stored by the OPEN in the GlobalPlatform Registry;
- Retrieving the card Life Cycle State;
- Obtaining access to the services of the Security Domain associated with the Application;
- Transitioning the card Life Cycle State to CARD_LOCKED;
- For contact cards according to ISO/IEC 7816-4 and for contactless cards Type A according to ISO/IEC 14443-3, setting the content of the historical bytes;
- Transitioning the Application's own Application Life Cycle State stored by the OPEN in the GlobalPlatform Registry;
- Transitioning the card Life Cycle State to TERMINATED;
- Obtaining access to GlobalPlatform Registry information;
- Obtaining access to Global Services Applications.

6.3 Command Dispatch

The commands received by a GlobalPlatform card shall be processed by the OPEN or dispatched to the selected Application for processing.

The SELECT [by name] command is processed by the OPEN.

One option of making the Issuer Security Domain the selected Application, is to specify its AID in a SELECT command with the [first or only occurrence] option set. As another option for making the Issuer Security Domain the selected Application, the SELECT command could contain no data in which case the AID of the Issuer Security Domain would be discovered by the off-card entity in the response to the SELECT command.

The processing of the MANAGE CHANNEL command is dependant on the capabilities of the card:

- If the card is aware of logical channels but only supports the Basic Logical Channel, the OPEN shall respond with the appropriate error;
- If the card is aware of logical channels and supports at least one Supplementary Logical Channel, the OPEN shall process the command; or
- If the card only supports the Basic Logical Channel and has no concept of logical channel support, the command shall be dispatched to the selected Application for processing.

Any other type of command received shall be dispatched to the currently selected Application.

Commands are either received on the Basic Logical Channel (logical channel number zero) or on a Supplementary Logical Channel (logical channel number other than zero). In compliance with ISO/IEC 7816-4, logical channel information shall be indicated in the class byte of the APDU command header. For all commands, if the command indicates a Supplementary Logical Channel that is not opened then:

- If the card only supports the Basic Logical Channel and has no concept of logical channel support, the command shall be dispatched to the selected Application for processing.
- If the card is aware of logical channels, the OPEN shall respond with the appropriate error. (This requirement may exclude the SELECT command, if the card supports opening of a logical channel using the SELECT command.)

For commands that are dispatched to an Application, it is the responsibility of the Application to correctly reject commands that it does not recognize, expect or cannot process.

The way in which an Application exhibits its multi-selection capabilities can vary according to the underlying platform and is beyond the scope of this Specification.

The Issuer Security Domain shall have no multi-selection restrictions on cards that support multiple logical channels i.e. it shall be capable of being selected across multiple logical channels.

6.4 Logical Channels and Application Selection

6.4.1 Implicit Selection Assignment

The following rules apply to the assignment of implicit selection to an Application:

- The Issuer Security Domain is by default the implicitly selectable Application on all logical channels of all card I/O interfaces supported by the card. Implicit selection on a specific logical channel of a specific card I/O interface may be assigned only if the Issuer Security Domain is the implicitly selectable Application on that logical channel of that card I/O interface and no other Locked Application is registered as implicitly selectable for the same logical channel and card I/O interface (see 9.6.2 - *Application Locking and Unlocking*;
- An Application installed or made selectable with a specific Implicit Selection parameter is registered in the GlobalPlatform Registry as the implicitly selectable Application on the logical channel(s) of the card I/O interface(s) indicated in the parameter if no other Application (other than the Issuer Security Domain) is already registered as implicitly selectable on that logical channel(s) of that card I/O interface(s);

- An Application installed or made selectable with the Card Reset privilege and no Implicit Selection parameter is registered in the GlobalPlatform Registry as the implicitly selectable Application on the Basic Logical Channel for all card I/O interfaces supported by the card if no other Application (other than the Issuer Security Domain) is already registered as implicitly selectable on the Basic Logical Channel of any card I/O interface;
- If an Application implicitly selectable on specific logical channel(s) of specific card I/O interface(s) is deleted, the Issuer Security Domain becomes the implicitly selectable Application on that logical channel(s) of that card I/O interface(s).

The OPEN shall use the implicit selection registration of each Application for controlling the following runtime behavioral requirements:

- Identifying the Application which is implicitly selectable on the Basic Logical Channel of the current card I/O interface during the card reset or activation sequence;
- Identifying the Application which is implicitly selectable when opening on the current card I/O interface a new Supplementary Logical Channel from the Basic Logical Channel.

6.4.2 Basic Logical Channel

The Basic Logical Channel is the permanently available interface to a GlobalPlatform card. This Basic Logical Channel shall be supported by the OPEN.

6.4.2.1 Application Selection on Basic Logical Channel

The OPEN shall support Application selection on the Basic Logical Channel via two processes:

- Implicit Selection following the card reset (see ISO/IEC 7816-3 for contact cards) or activation sequence (see ISO/IEC 14443-3 for contactless cards);
- Explicit Selection through the SELECT [by name] command.

The OPEN may also support additional selection processes.

Partial AID selection as defined in section 6.4.2.1.2 - *Explicit Selection*, shall be supported. (Partial AID selection does not require knowledge of the full AID by the off-card entity.) As multiple Applications on the card may have the same partial AID, it is required that a method exists to select all Applications matching the partial AID.

6.4.2.1.1 Implicit Selection on Basic Logical Channel

Once the card session has been established (for contact cards according to ISO/IEC 7816-4 after Answer-to-Reset, or after the activation sequence for contactless cards according to ISO/IEC 14443-3), and before the first command is issued to the card, the Application defined as implicitly selectable on the Basic Logical Channel and for that card I/O interface shall become the selected Application on the Basic Logical Channel for that card I/O interface.

Runtime Behavior

The following requirements apply for the OPEN for the implicit Application selection process:

- If the card is in the Life Cycle State CARD_LOCKED or TERMINATED, the Application with the Final Application privilege is the selected Application on the Basic Logical Channel and the OPEN shall not attempt to identify the implicitly selectable Application;
- In all other cases the OPEN shall search the GlobalPlatform Registry for an Application that is marked as implicitly selectable on the Basic Logical Channel for the current card I/O interface (e.g. contact or contactless), and if this Application is not in the Life Cycle State LOCKED, it shall become the selected Application on the Basic Logical Channel. If this is an Application in the Life Cycle State LOCKED, the

Application with the Final Application privilege shall become the selected Application on the Basic Logical Channel.

6.4.2.1.2 Explicit Selection on Basic Logical Channel

At any time during a Card Session the OPEN may receive a request to select an Application on the Basic Logical Channel (SELECT [by name] [first or only occurrence] command). The OPEN shall determine if the requested AID matches or partially matches an entry within the GlobalPlatform Registry and whether this entry is selectable.

At any time during a Card Session that has already contained a SELECT [by name] [first or only occurrence] command, the OPEN may receive a request to select a next Application (SELECT [by name] [next occurrence] command) on the Basic Logical Channel. The OPEN shall determine if the requested AID matches or partially matches another entry within the GlobalPlatform Registry and whether this entry is selectable.

For both the SELECT [by name] [first or only occurrence] command and the SELECT [by name] [next occurrence] command, an Application becomes the selected Application on the Basic Logical Channel if:

- The requested AID matches (fully or partially) the Application's AID;
- The Application being selected is in the correct Life Cycle State;
- The Application has no restrictions due to multi-selection, and supports the current card interface.

Runtime Behavior

The following requirements apply to the OPEN in the explicit Application selection (SELECT [by name]) process on the Basic Logical Channel (This behavior does not apply if the card Life Cycle State is TERMINATED):

- In the card Life Cycle State CARD_LOCKED:
  - If the Application being selected has the Final Application privilege, this Application is re-selected and a warning is returned to the off-card entity;
  - If any other Application is being selected, the Application with the Final Application privilege remains selected and an error is returned to the off- card entity;
- If a SELECT [by name] [first or only occurrence] or SELECT [by name] [next occurrence] is received and the data field of the command message is not present, the Issuer Security Domain shall become the currently selected Application and the SELECT command is dispatched to the Issuer Security Domain;
- If a SELECT [by name] [first or only occurrence] is received, the search always begins from the start of the GlobalPlatform Registry;
- If a SELECT [by name] [next occurrence] is received, the search always begins from the entry following the currently selected Application on the Basic Logical Channel in the GlobalPlatform Registry;
- If a full or partial match is found and this Application is in the Life Cycle State INSTALLED, continue searching through the GlobalPlatform Registry for a subsequent full or partial match. If no subsequent full or partial match is found, the OPEN shall return the appropriate error to the off-card entity;
- If a full or partial match is found and this Application is in the Life Cycle State LOCKED, continue searching through the GlobalPlatform Registry for a subsequent full or partial match. In the eventuality that this locked Application is already currently selected on the Basic Logical Channel, the OPEN shall terminate this Application Session. If no subsequent full or partial match is found, the OPEN shall return the appropriate error to the off-card entity;
- If a full or partial match is found and this Application cannot be selected due to a multi-selection restriction or because the Application refuses selection (e.g. because it does not support the current card interface), continue searching through the GlobalPlatform Registry for a subsequent full or partial match. If no subsequent full or partial match is found, the OPEN shall return the appropriate error to the off-card entity;

- If a full or partial match is found and this Application is selectable (i.e. in the correct Life Cycle State and has no multi-selection restrictions), then it shall become the currently selected Application on the Basic Logical Channel and the SELECT [by name] command, whether [first or only occurrence] or [next occurrence], shall be processed according to the requirements of the runtime environment (e.g. dispatched to the Application);
- If no full or partial match is found at all, the currently selected Application on the Basic Logical Channel shall remain the selected Application and
  - If the SELECT [by name] command has the [first or only occurrence] parameter set, the SELECT command is dispatched to the Application;
  - If the SELECT [by name] command has the [next occurrence] parameter set, the OPEN shall return the appropriate error to the off-card entity;
- In the eventuality that the current Application Session has been terminated and no subsequent full or partial match is found, the OPEN shall make an attempt to select the Application that is marked as implicitly selectable on the Basic Logical Channel for the current card interface.

6.4.2.2 Logical Channel Management on Basic Logical Channel

At any time during a Card Session the OPEN may receive a request on the Basic Logical Channel to either open or close a Supplementary Logical Channel.

If the card only supports the Basic Logical Channel and has no concept of logical channel support, the MANAGE CHANNEL command is dispatched to the currently selected Application. In this case, when a Security Domain is the currently selected Application, the command shall be rejected.

On cards that support logical channels, if a MANAGE CHANNEL [open] is received:

- If an Application is designated as implicitly selectable on the new Supplementary Logical Channel for the current card interface, that Application is implicitly selected on the newly opened Supplementary Logical Channel and runtime behavior requirements apply;
- Otherwise the Application designated as implicitly selectable on the Basic Logical Channel for this card interface is implicitly selected on the newly opened Supplementary Logical Channel and runtime behavior requirements apply.

On cards that support logical channels, if a MANAGE CHANNEL [close] is received, terminate the Application Session currently selected on the Supplementary Logical Channel indicated by the command and then close that logical channel. The Basic Logical Channel can never be closed.

Runtime Behavior

On receipt of a MANAGE CHANNEL [open] command, the following requirements apply:

- If the card is in the Life Cycle State CARD_LOCKED or TERMINATED, return the appropriate error.
- If the number of logical channels supported by the OPEN is not sufficient to open a new Supplementary Logical Channel, return the appropriate error.
- The OPEN shall search the GlobalPlatform Registry for the Application that supports the current card interface and that is marked as implicitly selectable on the new Supplementary Logical Channel (or failing that, on the Basic Logical Channel) and:
  - If this is an Application in the Life Cycle State LOCKED, the Application with the Final Application privilege shall become the selected Application on the Supplementary Logical Channel;
  - If this Application cannot be selected due to a multi-selection restriction, the new logical channel shall not be opened and the OPEN shall return the appropriate error;

- Otherwise, the Supplementary Logical Channel is opened and this Application shall become the selected Application on the Supplementary Logical Channel.

6.4.2.3 Application Command Dispatch on Basic Logical Channel

Once an Application becomes the selected Application on the Basic Logical Channel, the responsibility for subsequent command dispatching still rests with the OPEN.

Processing SELECT [by name] commands and runtime behavior requirements for OPEN are described in section 6.4.2.1.2 - *Explicit Selection*.

On cards that are aware of logical channels, the MANAGE CHANNEL commands are only processed by the OPEN and are not dispatched to an Application.

All other commands (including the MANAGE CHANNEL commands on cards that are not aware of logical channels or SELECT commands not described in section 6.4.2.1.2 - *Explicit Selection*) are immediately dispatched to the Application currently selected on the Basic Logical Channel. The processing of the command by the Application is beyond the scope of this Specification.

6.4.3 Supplementary Logical Channel

A Supplementary Logical Channel, if supported, allows an Application to be selected simultaneously to the Applications selected on other logical channels.

6.4.3.1 Application Selection on Supplementary Logical Channel

The OPEN shall support Application selection on an available Supplementary Logical Channel via two processes:

- Implicit Selection following a successful MANAGE CHANNEL [open] command;
- Explicit Selection through the SELECT [by name] command.

The OPEN may also support additional selection processes.

Partial AID selection as defined in section 6.4.3.1.2 - *Explicit Selection*, shall be supported on Supplementary Logical Channels.

6.4.3.1.1 Implicit Selection on Supplementary Logical Channel

Depending on whether a Supplementary Logical Channel is being opened from the Basic Logical Channel or from another Supplementary Logical Channel, the behavior of implicit selection differs.

Refer to section 6.4.2.2 - *Logical Channel Management* for the behavior of implicit selection initiated from the Basic Logical Channel.

Refer to section 6.4.3.2 - *Logical Channel Management* for the behavior of implicit selection initiated from a Supplementary Logical Channel.

6.4.3.1.2 Explicit Selection on Supplementary Logical Channel

At any time on an open Supplementary Logical Channel, the OPEN may receive a request to select an Application on this Supplementary Logical Channel (SELECT [by name] [first or only occurrence] command). The OPEN shall determine if the requested AID matches or partially matches an entry within the GlobalPlatform Registry and whether this entry is selectable.

At any time on an open Supplementary Logical Channel that has already contained a SELECT [by name] [first or only occurrence] command since the Supplementary Logical Channel was last opened, the OPEN may receive a

request to select a next Application (SELECT [by name] [next occurrence] command) on this Supplementary Logical Channel. The OPEN shall determine if the requested AID matches or partially matches another entry within the GlobalPlatform Registry and whether this entry is selectable.

For both the SELECT [by name] [first or only occurrence] command and the SELECT [by name] [next occurrence] command, an Application becomes the selected Application on the Supplementary Logical Channel if:

- The requested AID matches (fully or partially) the Application's AID;
- The Application being selected is in the correct Life Cycle State;
- The Application has no restrictions due to multi-selection, and supports the current card interface.

Runtime Behavior

The following requirements apply to the OPEN in the explicit Application selection (SELECT [by name]) process on a Supplementary Logical Channel:

- If the card is in the Life Cycle State CARD_LOCKED or TERMINATED:
  - Close the Supplementary Logical Channel, if currently open;
  - Return the appropriate error;
- If a SELECT [by name] [first or only occurrence] or SELECT [by name] [next occurrence] is received and the data field of the command message is not present, the Issuer Security Domain shall become the currently selected Application and the SELECT command is dispatched to the Issuer Security Domain;
- If a SELECT [by name] [first or only occurrence] is received, the search always begins from the start of the GlobalPlatform Registry;
- If a SELECT [by name] [next occurrence] is received, the search always begins from the entry following the currently selected Application on this Supplementary Logical Channel in the GlobalPlatform Registry;
- If a full or partial match is found and this Application is in the Life Cycle State INSTALLED, continue searching through the GlobalPlatform Registry for a subsequent full or partial match. If no subsequent full or partial match is found, the OPEN shall return the appropriate error to the off-card entity;
- If a full or partial match is found and this Application is in the Life Cycle State LOCKED, continue searching through the GlobalPlatform Registry for a subsequent full or partial match. In the eventuality that this locked Application is already currently selected on the same Supplementary Logical Channel, the OPEN shall terminate this Application Session. If no subsequent full or partial match is found, the OPEN shall return the appropriate error to the off-card entity;
- If a full or partial match is found and this Application cannot be selected due to a multi-selection restriction or because the Application refuses selection (e.g. because it does not support the current card interface), continue searching through the GlobalPlatform Registry for a subsequent full or partial match. If no subsequent full or partial match is found, the OPEN shall return the appropriate error to the off-card entity;
- If a full or partial match is found and this Application is selectable (i.e. in the correct Life Cycle State and has no multi-selection restrictions), then it shall become the currently selected Application on this Supplementary Logical Channel and the SELECT [by name] command, whether [first or only occurrence] or [next occurrence], shall be processed according to the requirements of the runtime environment (e.g. dispatched to the Application);
- If no full or partial match is found at all, the currently selected Application on the Supplementary Logical Channel shall remain the selected Application and
  - If the SELECT [by name] command has the [first or only occurrence] parameter set, the SELECT command is dispatched to the Application;

- If the SELECT [by name] command has the [next occurrence] parameter set, the OPEN shall return the appropriate error to the off-card entity.

6.4.3.2 Logical Channel Management on Supplementary Logical Channel

At any time on an open Supplementary Logical Channel the OPEN may receive a request to either open or close a Supplementary Logical Channel.

If a MANAGE CHANNEL [open] is received and the Application selected on the original Supplementary Logical Channel has no multi-selection restrictions, this Application becomes the selected Application on the newly opened Supplementary Logical Channel.

If a MANAGE CHANNEL [close] is received, terminate the Application Session currently selected on the Supplementary Logical Channel indicated by the command and then close that logical channel. The Basic Logical Channel can never be closed.

Runtime Behavior

On receipt of a MANAGE CHANNEL [open] command, the following requirements apply:

- If the card is in the Life Cycle State CARD_LOCKED or TERMINATED, return the appropriate error;
- If the number of logical channels supported by the OPEN is not sufficient to open a new Supplementary Logical Channel, return the appropriate error;
- If the Application currently selected on the original Supplementary Logical Channel cannot be selected on the new Supplementary Logical Channel due to a multi-selection restriction, the new logical channel shall not be opened and the OPEN shall return the appropriate error;
- Otherwise, the Supplementary Logical Channel indicated by the command is opened and the Application currently selected on the original Supplementary Logical Channel shall become the selected Application on the newly opened Supplementary Logical Channel.

6.4.3.3 Application Command Dispatch on Supplementary Logical Channel

Once an Application becomes the selected Application on a Supplementary Logical Channel, the responsibility for subsequent command dispatching still rests with the OPEN.

Processing SELECT [by name] commands and runtime behavior requirements for OPEN are described in section 6.4.3.1.2 - *Explicit Selection*.

The MANAGE CHANNEL commands are only processed by the OPEN and are not dispatched to an Application.

All other commands (including SELECT commands not described in section 6.4.3.1.2 - *Explicit Selection*) are immediately dispatched to the Application currently selected on the Supplementary Logical Channel. The processing of the command by the Application is beyond the scope of this Specification.

6.5 GlobalPlatform Registry

The GlobalPlatform Registry is used to:

- Store card management information;
- Store relevant application management information (e.g., AID, associated Security Domain and Privileges);
- Support card resource management data;
- Store Application Life Cycle information;

- Store card Life Cycle information;
- Track any counters associated with logs.

The contents of the GlobalPlatform Registry may be updated in response to:

- An internal OPEN invoked action;
- An authorized Application invoked action.

All Applications including all Security Domains, and all Executable Load Files, shall have an entry in the GlobalPlatform Registry.

There is no mandatory format for the storage of these data elements. However, format requirements do exist for the handling of the data elements via APDU commands and GlobalPlatform services available to Applications.

6.5.1 Application/Executable Load File/Executable Module Data Elements

6.5.1.1 Application/Executable Load File/Executable Module AID

Each Executable Load File or Executable Module is associated with an AID that shall be unique on the card.

An Application AID may be the same as that of an Executable Module but may not be the same as that of an Executable Load File or the same as another Application already present in the GlobalPlatform Registry.

This AID may be specified in a SELECT command to select the Application. It is not possible to select Executable Load Files or Executable Modules.

6.5.1.2 Application/Executable Load File/Executable Module Life Cycle

The Application Life Cycle State data element contains the current Life Cycle of the Application, Executable Load File or Executable Module.

6.5.1.3 Memory Resource Management Parameters

Resource management data elements contain information about the memory resources that are available to an Application. They are system-specific values and are used as a control mechanism by the OPEN to manage memory resource allocation as described in section 9.7 - *Memory Resource Management*.

When additional resources are requested by an Application, the OPEN shall validate the request against the value of this data element in the GlobalPlatform Registry.

6.5.1.4 Privileges

The Privileges data element indicates the privileges for each Application. Privileges are defined in section 6.6.

6.5.1.5 Implicit Selection Parameter

The implicit selection parameter indicates whether an Application is implicitly selectable on a specific logical channel of a specific card interface: see section 11.1.7 and Table 11-50.

6.5.1.6 Associated Security Domain AID

All Executable Load Files and Applications, including Security Domains, are associated with a Security Domain, whose AID is given in the registry. This AID is the AID of the Security Domain directly associated. A Security

Domain also has an associated Security Domain: either another Security Domain or itself, hence describing an association hierarchy of Executable Load Files / Applications and Security Domains.

6.5.1.7 Application Provider ID

The Registry holds the identifier of the Application Provider: the owner of the Application or Executable Load File when explicitly provided during the load or installation process.

An on-card entity may use this information to enforce security policies. Establishing whether an off-card entity is also the owner of the on-card entity being managed is addressed in more detail in section 10.4 - *Entity Authentication*.

6.5.1.8 Service Parameter

A Service Parameter identifies the service offered by the Global Services Application. The Global Services Application may register this service as a unique Global Service as described in section 8.1.1 - *Registering Global Services*. More than one service, hence more than one Service Parameter, may be registered for a Global Services Application.

6.5.2 Card-Wide Data

The card Life Cycle State is stored in the GlobalPlatform Registry similarly to Application information. Any restriction of the Card Content Management functionality of OPEN is stored in the GlobalPlatform Registry.

6.6 Privileges

6.6.1 Privilege Definition

The following table defines Security Domain and Application privileges:

Privilege Number	Privilege	Description	Notes
0	Security Domain	Application is a Security Domain.	Distinguishes a Security Domain from a 'normal' Application.
1	DAP Verification	Application is capable of verifying a DAP; Security Domain privilege shall also be set.	For details, see section 9.2.1.
2	Delegated Management	Application is capable of Delegated Card Content Management: Security Domain privilege shall also be set.	For details, see section 9.1.3.3.
3	Card Lock	Application has the privilege to lock the card.	For details, see section 9.6.3.
4	Card Terminate	Application has the privilege to terminate the card.	For details, see section 9.6.4.
5	Card Reset	Application has the privilege to modify historical bytes on one or more card interfaces. This privilege was previously labeled "Default Selected".	For details, see section 6.6.2.

Privilege Number	Privilege	Description	Notes
6	CVM Management	Application has the privilege to manage a shared CVM of a CVM Application.	For details, see section 8.2.1.
7	Mandated DAP Verification	Application is capable of and requires the verification of a DAP for all load operations: Security Domain privilege and DAP Verification privilege shall also be set.	For details, see section 9.2.1.
8	Trusted Path	Application is a Trusted Path for inter-application communication.	For details, see section 6.7.
9	Authorized Management	Application is capable of Card Content Management; Security Domain privilege shall also be set.	For details, see section 9.1.3.2.
10	Token Verification	Application is capable of verifying a token for Delegated Card Content Management.	For details, see sections 9.1.3.1 and 9.2.3.
11	Global Delete	Application may delete any Card Content.	For details, see sections 9.1.3.4 and 9.5.
12	Global Lock	Application may lock or unlock any Application.	For details, see section 9.1.3.5 and 9.6.2.
13	Global Registry	Application may access any entry in the GlobalPlatform Registry.	For details, see section 9.6.5.
14	Final Application	The only Application accessible in card Life Cycle State CARD_LOCKED and TERMINATED.	For details, see section 9.6.4.
15	Global Service	Application provides services to other Applications on the card.	For details, see section 8.1.1.
16	Receipt Generation	Application is capable of generating a receipt for Delegated Card Content Management.	For details, see section 9.1.3.6

Table 6-1: Privileges

6.6.2 Privilege Assignment

The following rules apply to the assignment of Privileges:

- Only one Application or Security Domain in the card may be set with the Card Reset privilege at a time (e.g. the Issuer Security Domain, a current legacy Application or an Application that requires specific behavior with regards to logical channels) on a given card interface;
- Once the Card Reset privilege has been assigned to an Application for a given card interface, the privilege can be reassigned to a new Application either by deleting the Application which has the privilege, or by revoking its privilege;

- The Card Reset privilege is by default assigned to the Issuer Security Domain. It may be reassigned only if the Issuer Security Domain has the Card Reset privilege;
- If the application with the Card Reset privilege is deleted, the privilege is reassigned to the Issuer Security Domain;
- The Final Application privilege is by default assigned to the Issuer Security Domain. It may be reassigned only if the Issuer Security Domain has the Final Application privilege;
- Only one Application or Security Domain in the card may be set with the Final Application privilege at a time (e.g. the Issuer Security Domain, a current legacy Application or an Application that requires specific behavior with regards to logical channels);
- Once the Final Application privilege has been assigned to an Application, the privilege can be reassigned to a new Application either by deleting the Application which has the privilege, or by revoking its privilege;
- If the application with the Final Application privilege is deleted, the Issuer Security Domain becomes the Application with Final Application privilege;
- Authorized Management and Delegated Management privileges are mutually exclusive;
- Otherwise, the privileges are not mutually exclusive; therefore, one or more privileges may be marked as set for an Application.

In the OP_READY card Life Cycle State, the Issuer Security Domain shall initially have the following set of privileges clearly identifying its functionality: Security Domain, Authorized Management, Global Registry, Global Lock, Global Delete, Token Verification, Card Lock, Card Terminate, Trusted Path, CVM Management, Card Reset, Final Application and Receipt Generation.

Privileges may be added or revoked during the Life Cycle of an Application or Security Domain.

For backward compatibility, where a card supports only privileges 0-7, the following assumptions shall apply for the remaining privileges:

Privilege Number	Privilege	Issuer Security Domain	Supplementary Security Domain	Other Application
8	Trusted Path	yes	yes	no
9	Authorized Management	yes	no	no
10	Token Verification	yes	no	no
11	Global Delete	yes	no	no
12	Global Lock	yes	no	no
13	Global Registry	yes	no	no
14	Final Application	yes	no	no
15	Global Service	no	no	no
16	Receipt Generation	yes	no	no

Table 6-2: Privilege Defaults

Several use cases can be envisaged for the assignment of privileges to Security Domains, depending on the business relationships between the Card Issuer and other off-card entities: Application Providers or Controlling Authority. The following table shows four example use cases.

		Use case			
		A	B	C	D
Issuer Security Domain	Authorized Management Privilege	✓	✓	✓	
	Delegated Management Privilege				✓
	Token Verification Privilege	✓	✓		✓
	Receipt Generation Privilege	✓	✓		✓
Supplementary Security Domain	Authorized Management Privilege				
	Delegated Management Privilege		✓		✓
	Token Verification Privilege			✓	
	Receipt Generation Privilege			✓	

Table 6-3: Privilege Assignment Example Use Cases

6.6.3 Privilege Management

Runtime Behavior

The OPEN shall use the Privileges data element in conjunction with the association hierarchy (see section 7.2 - *Security Domain Association*) of the Applications for controlling the following runtime behavioral requirements:

- Ensuring Token verification when required (see section 9.3.2 - *Card Content Loading*);
- Ensuring Receipt generation when required (see section 9.3.2 - *Card Content Loading*);
- Ensuring DAP verification when required (see section 9.3.2 - *Card Content Loading*);
- Checking for the validity of a request to change Card Contents: load, installation, extradition, registry update, or deletion;
- Checking for the validity of a request to lock or unlock the card;
- Checking for the validity of a request to lock or unlock an Application, including a Security Domain;
- Checking for the validity of a request to terminate the card;
- Checking for the validity of a request to communicate with another on-card Application;
- Checking for the validity of a request to access the GlobalPlatform Registry entry of another Application, including a Security Domain;
- Identifying the Application with Final Application privilege to be default selected when the card is in CARD_LOCKED or TERMINATED state.

The OPEN shall use the Privileges data element and the Implicit Selection parameter for each Application (where present) for controlling the following runtime behavioral requirements:

- Checking for the validity of a request to modify historical bytes; for dual interface (contact and contactless) cards, the modification applies to the historical bytes of the card interface(s) for which the requesting on-card Application is registered as implicitly selectable; this feature should preferably be used on cards that support the Basic Logical Channel only.

6.7 The GlobalPlatform Trusted Framework

An APDU command is received by the Application's Security Domain (the Receiving Entity), which may be the Issuer Security Domain or a Supplementary Security Domain, and is unwrapped by the Security Domain before being passed on to the GlobalPlatform Trusted Framework. The objective is that the GlobalPlatform Trusted Framework forwards the unwrapped command to the Target Application indicated by the Receiving Entity.

The Target Application may use cryptographic services of the Security Domain available through the Secure Channel Session established by the Security Domain (e.g. sensitive data decryption). The Current Security Level viewed by the Target Application may differ from the Security Domain view depending on the Secure Channel Protocol and the authenticated off-card entity's Application Provider ID (e.g. ANY_AUTHENTICATED for the Target Application and AUTHENTICATED for the Security Domain).

Runtime Behavior

The GlobalPlatform Trusted Framework shall check that:

- The Receiving Entity has the Trusted Path privilege;
- The Receiving Entity is a Security Domain;
- The Target Application exists in the GlobalPlatform Registry and has enabled its 'Process Data' entry point;
- The Target Application has no multi-selection restrictions if it is already selected on another logical channel;
- The Target Application is associated with the currently selected Receiving Entity Security Domain.

If all checks are passed, the GlobalPlatform Trusted Framework passes the command to the Target Application through its GlobalPlatform Application interface.

The GlobalPlatform Trusted Framework and the Applications involved are shown in the following diagram. The process as used in personalization is described in more detail in section 7.3.3 - *Personalization Support*.

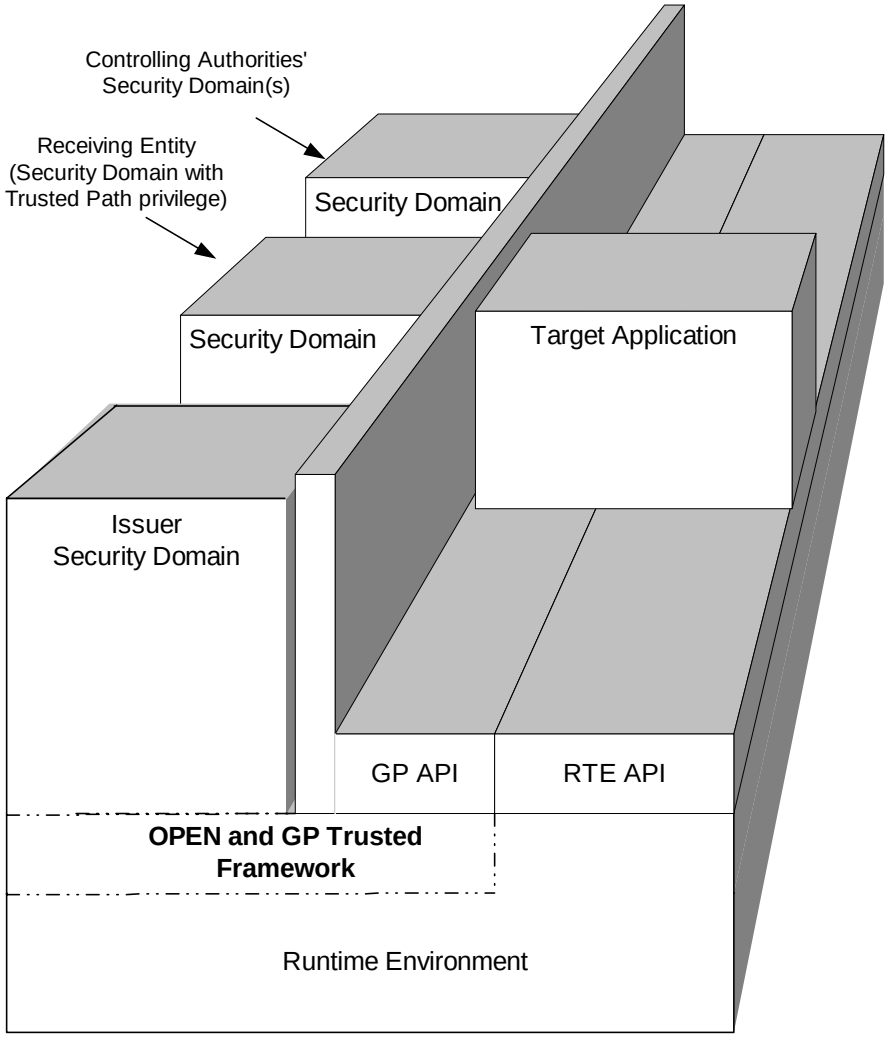

Figure 6-1: GlobalPlatform Trusted Framework Roles

7 Security Domains

7.1 General Description

Security Domains are privileged Applications. They hold cryptographic keys which can be used to support Secure Channel Protocol operations and/or to authorize card content management functions.

Each Application and each Executable Load File is associated with a Security Domain. An Application can use the cryptographic services of its associated Security Domain. It is possible to associate Applications owned by one authority with the Security Domain of another authority.

All cards have one mandatory Security Domain: the Issuer Security Domain. A card that supports multiple Security Domains can allow an Application Provider, through its own Security Domain, to manage its own Applications and provide cryptographic services using keys that are completely separate from, and not under the control of, the Card Issuer.

Key Separation

Security Domains are responsible for their own key management. This ensures that Applications and data from different Application Providers may coexist on the same card without violating the privacy and integrity of each Application Provider.

Application Services

The keys and associated cryptography for all Security Domains may be used for:

- Personalization Support: Secure communication support during personalization of an Application Provider's Applications;
- Runtime Messaging Support: secure communication support during runtime for an Application that does not contain its own secure messaging keys.

7.1.1 Issuer Security Domain

The Issuer Security Domain primarily operates as any Security Domain, but has some special characteristics that distinguish it from any other Security Domain:

- It is the first Application installed on a card. GlobalPlatform does not mandate that the Issuer Security Domain be loaded or installed in the same manner as Applications. The Issuer Security Domain, while viewed by the GlobalPlatform Registry as an Application, has implementation specific behavior relating to how it becomes an active entity on the card;
- It does not have a Security Domain Life Cycle State because it inherits the card Life Cycle State;
- It takes on the Card Reset privilege if the Application with that privilege is removed;
- It becomes the implicitly selected Application if the Application implicitly selectable on the same logical channel of the same card I/O interface is removed;
- It becomes the implicitly selected Application for a card interface on a logical channel if the Application that is implicitly selectable on that logical channel for that card interface is removed;
- It may be selected by use of the SELECT command with no command data field;
- It takes on the Final Application privilege if the Application with that privilege is removed.

7.2 Security Domain Association

Applications (including Security Domains) may use the services of their associated Security Domain to provide Secure Channel Sessions and other cryptographic services. An Application need not have any prior knowledge of its Security Domain AID: the GlobalPlatform Registry contains this information and the OPEN supplies the Application with a reference to its associated Security Domain. The Application should not store this reference for future use because its associated Security Domain may change as a result of extradition.

Extradition is the means by which an Application is associated with a different Security Domain.

An Executable Load File is initially associated with the Security Domain which loads it, but it may immediately (implicit extradition during the load process) or subsequently (explicit extradition) be extradited to a different Security Domain.

The Issuer Security Domain is effectively associated with itself, its establishment on the card is not defined by GlobalPlatform, and it is not subject to extradition.

Through extradition, a Security Domain may be associated with itself. This means that if it calls on the services of its associated Security Domain, it will be using its own services. It also makes it more isolated from other Security Domains, subject to the privileges assigned to those other Security Domains.

As a result, there is one or more hierarchies of association on a card. The root of each hierarchy is a Security Domain that is associated with itself. The following diagram shows two hierarchies: one starting from the Issuer Security Domain, the other from a Security Domain which has been extradited to itself; arrows indicate 'is associated with'.

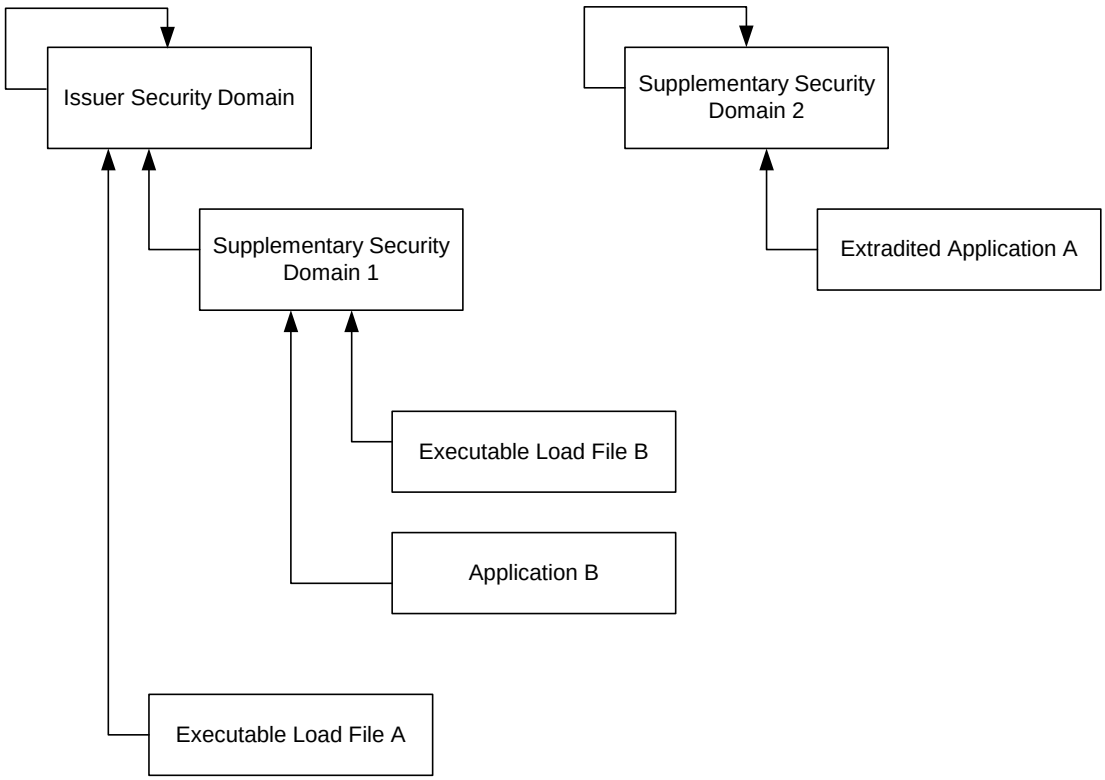

Figure 7-1: Example of Security Domain Hierarchies

A Security Domain associated with another Application is said to be directly associated with it. Another Security Domain higher up the hierarchy is said to be indirectly associated with it.

7.3 Security Domain Services

Appendix A.1 - *GlobalPlatform on a Java Card* provides the Java Card™ specification of the following interfaces.

Appendix A.2 - *GlobalPlatform on MULTOS*™ provides the MULTOS™ specification of the following interfaces.

Appendix D - *Secure Channel Protocol '01' (Deprecated)*-, appendix E - *Secure Channel Protocol '02'* and appendix F - *Secure Channel Protocol '10'* provide more details on Secure Channel Protocols implementing the following interfaces.

7.3.1 Application Access to Security Domain Services

Applications have the ability to access the services of their associated Security Domain. By using these services, the Application may rely on cryptographic support from the Security Domain to ensure confidentiality and integrity during personalization and runtime. The Security Domain services defined in this Specification are generic and shall encompass the following possibilities.

- Initiating a Secure Channel Session upon successful verification of an off-card entity;
- Unwrapping a command received within a Secure Channel Session by verifying its integrity and/or decrypting the original data in the case of confidentiality;
- Controlling the sequence of APDU commands;
- Decrypting a secret data block;
- Setting the security level: integrity and/or confidentiality, to apply to the next incoming command and/or next outgoing response;
- Closing a Secure Channel Session upon request and destroying any secret(s) relating to that Secure Channel Session.

Depending on the specific Secure Channel Protocol supported, the Security Domain services may also encompass the possibility of:

- Wrapping a response sent within a Secure Channel Session by adding integrity and/or encrypting the original data in the case of confidentiality;
- Encrypting a secret data block;
- Controlling the sequence of APDU responses.

A Security Domain may support the management of multiple Secure Channel Sessions concurrently (i.e. Applications selected on multiple logical channels each initiating a Secure Channel) or may limit itself to only managing one Secure Channel Session immaterial of the number of concurrently selected Applications that attempt to use its services. If a Security Domain does support the management of multiple Secure Channel Sessions concurrently, it shall be able to differentiate between the multiple Secure Channel Sessions and their respective logical channels. If a Security Domain does not support the management of multiple Secure Channel Sessions concurrently, when a request is made to open a Secure Channel Session on a logical channel different from the current Secure Channel Session, the Security Domain rejects this request to initiate a new Secure Channel Session.

If a request is made by an Application to use the services of its associated Security Domain on a logical channel different from the one on which its Secure Channel Session was opened, the Security Domain shall reject this request.

7.3.2 Security Domain Access to Applications

The Security Domain has the facility to receive a STORE DATA command destined to one of its associated Applications. The Security Domain unwraps this command according to the Current Security Level of the current Secure Channel Session and prior to the command being forwarded to the Application. This pre-processing leaves as is the data structures of the command message as well as the eventual encryption of the data value fields of these data structures.

The command is forwarded to the Application by the GlobalPlatform Trusted Framework which handles inter-application communication between Security Domains and Applications, as described in section 6.7.

7.3.3 Personalization Support

After an Application is installed, the Application may need to obtain its personalization data, including its own keys and Cardholder-specific data. The Application can utilize the secure communication and key decryption services of its associated Security Domain to manage the secure loading of this personalization data. This can be achieved in two ways: one being to utilize the runtime messaging support described in section 7.3.4 - *Runtime Messaging Support*, the other being to utilize the Security Domain's ability to access the Application as described in section 7.3.2.

This section addresses the second method.

Using this second method on a card that only supports the Basic Logical Channel allows the Application to be personalized without the Application being the selected Application. Rather the Security Domain is the selected Application and the Security Domain receives commands on behalf of the Application. The Security Domain would pre-process the personalization (STORE DATA) command prior to forwarding the command to the Application via the GlobalPlatform Trusted Framework.

Using this second method on a card that supports multiple logical channels differs slightly only in that the Application could have a multi-selection restriction in which case personalization would fail. If an Application does not support this second method, the Security Domain shall reject any STORE DATA command destined to this Application.

Runtime Behavior

On receipt of an INSTALL [for personalization] command and subsequent STORE DATA commands, the Security Domain performing the personalization of the Application shall:

- Apply its own secure communication policy;
- Check that the off-card entity is authenticated as the Application Provider;
- Pre-process subsequent STORE DATA commands according to the Current Security Level of the Secure Channel Session and prior to the command being forwarded to the target Application.

On receipt of a request to forward commands to an Application, the OPEN shall:

- Check that the card Life Cycle State is not CARD_LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restrictions for personalization (see Table 11-53);
- Apply the GlobalPlatform Trusted Framework runtime requirements defined in section 6.7.

Personalization Flow

The following diagram is an example of an Application receiving data from its associated Security Domain during personalization.

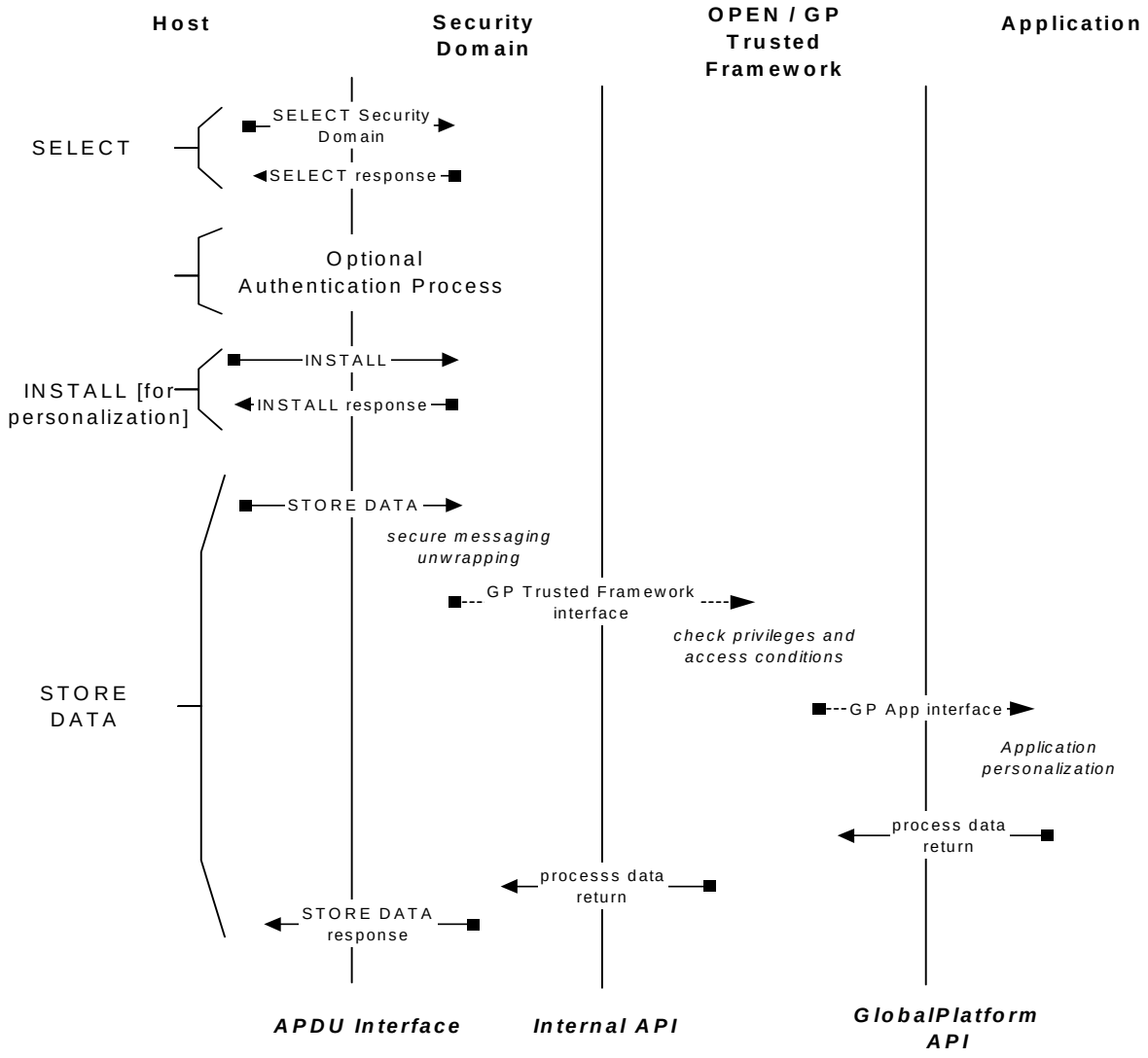

Figure 7-2: Application Personalization through Associated Security Domain

7.3.4 Runtime Messaging Support

Security Domains may provide runtime support for secure communication to their directly associated Applications. Instead of loading additional communication keys into an Application, the Application Provider may choose to use the associated Security Domain services for all Application communication throughout the life of the Application.

Runtime Messaging Flow

The following diagram is an example of an Application using the services of its associated Security Domain:

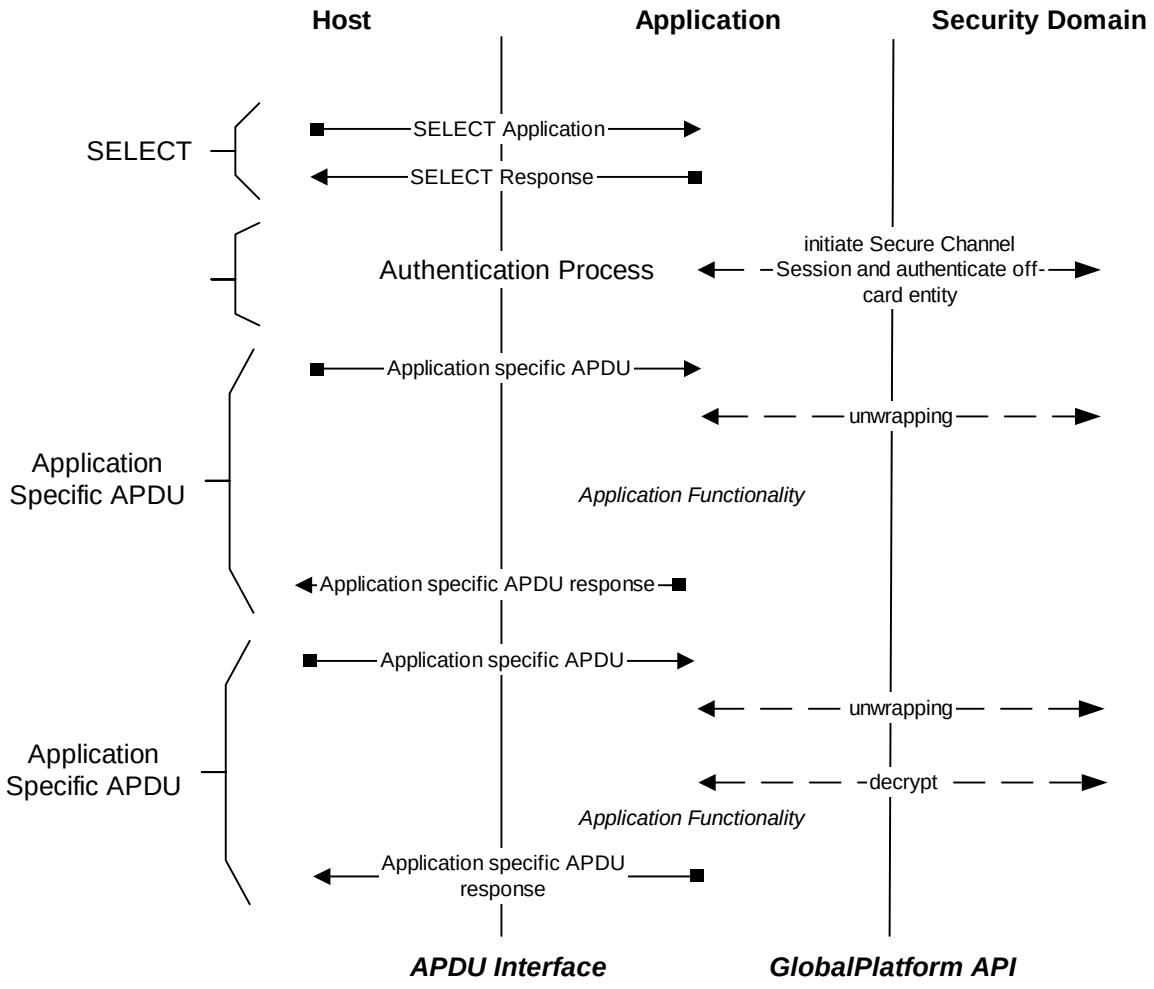

Figure 7-3: Runtime Messaging Flow

7.4 Security Domain Data

7.4.1 Issuer Security Domain

The Issuer Security Domain shall be able to handle the following data:

- Issuer Identification Number (IIN);
- Card Image Number (CIN);
- Card Recognition Data;
- Other Card Issuer's proprietary data.

These data shall be available from the card using the GET DATA command.

7.4.1.1 Issuer Identification Number

The Issuer Identification Number (IIN) may be used by an off-card entity to associate the card with a specific Card Management System. The IIN typically contains the ISO 7812 defined identification of the Issuer and is carried by the ISO/IEC 7816-6 tag '42'. The IIN data element is of variable length.

7.4.1.2 Card Image Number

The Card Image Number (CIN) may be used by a Card Management System to uniquely identify a card within its card base. It is a unique value, carried by the ISO/IEC 7816 defined tag '45' (Card Issuer's Data), which is assigned by the Card Issuer (identified itself by the IIN). The CIN data element is of variable length.

7.4.1.3 Card Recognition Data

Card Management Systems need information about a card before they can start to interact with it. This includes the kind of card it is and what Secure Channel Protocol it supports.

Card Recognition Data is the mechanism for providing information about a card, and thus avoiding the vagaries of trial-and-error. See appendix H for details.

Card Recognition Data shall be present and contained in a data template (tag '73'). This template shall in turn be contained in the “Card Data” data object (tag '66'), as defined in ISO/IEC 7816-6. Card Data can be retrieved using the GET DATA command. (For contact cards according to ISO/IEC 7816-4, Card Data may also be accessed through the ATR, if a suitable “command to perform” is included in the ATR historical bytes.)

Note that the information provided in Card Recognition Data should be enough to enable initial communication with the card without resorting to trial and error. Information not essential for this purpose should be supplied during subsequent interaction with the card.

There is no specific requirement for Card Recognition Data to be updated dynamically by the card, but additional dynamic data objects are not precluded.

7.4.2 Supplementary Security Domains

Security Domains other than the Issuer Security Domain may handle their own identification data:

- A Security Domain Provider Identification Number (SIN);
- A Security Domain Image Number;
- Security Domain Management Data;
- Other Application Provider proprietary data.

When present, these data shall be available from the Security Domain using the GET DATA command.

7.4.2.1 Security Domain Provider Identification Number

The Security Domain Provider Identification Number (SIN) may be used by an off-card entity to associate the Security Domain with a specific Card Management System. It is an IIN, typically contains the ISO 7812 defined identification of the Security Domain provider, and is carried by the ISO/IEC 7816-6 tag '42'. The SIN data element is of variable length.

7.4.2.2 A Security Domain Image Number

The Security Domain Image Number may be used by an Application Management System to uniquely identify an instance of a Security Domain on a card. If used it is a unique value, carried by the ISO/IEC 7816 defined tag '45'.

7.4.2.3 Security Domain Management Data

Application Management Systems need information about a Security Domain before they can start to interact with it. This includes the kind of Security Domain it is and what Secure Channel Protocol it supports.

Security Domain Management Data is the mechanism for providing information about a Security Domain, and thus avoiding the vagaries of trial-and-error. Security Domain Management Data (tag '73') shall be returned in the response to the SELECT command when present. Security Domain Management Data shall also be returned in response to a GET DATA command with tag '66'.

Note that the information provided in Security Domain Management Data should be enough to enable initial communication with the card without resorting to trial and error.

There is no specific requirement for Security Domain Management Data to be updated dynamically by the card, but additional dynamic data objects are not precluded.

For further details refer to appendix H.

7.5 Security Domain Keys

7.5.1 Key Information

See the appropriate appendix D, E, or F for details of the keys used with each Secure Channel Protocol. See appendix C.1 for details of token, receipt and DAP keys.

Keys have the following attributes:

- A Key Identifier, which identifies each key within an on-card entity. A key may consist of one or more key components, e.g. a symmetric key has only one key component while an asymmetric key has several components. All key components share the same Key Identifier. Different Key Identifiers within one on-card entity shall be used to differentiate keys, their usage, and their functionality. There is no restriction and no pre-defined order in assigning Key Identifiers to keys, including non-sequential Key Identifiers within the same entity;
- An associated Key Version Number: different key versions within an on-card entity may be used to differentiate several instances or versions of the same key. There is no restriction and no pre-defined order in assigning Key Version Numbers to keys;
- An associated cryptographic algorithm: a specific key is associated with one and only one cryptographic algorithm;
- A length, for cryptographic algorithms supporting several key (or key component) lengths;
- Access conditions, to control and segment access to keys.

These key attributes together indicate unambiguously to the on-card entity:

- The identity;
- The intended usage;
- The functionality of cryptographic keys.

The combination of a Key Identifier and a Key Version Number identifies unambiguously a key within an on-card entity. The key type identifies the cryptographic algorithm and key component. Identifying unambiguously a key and an algorithm within an entity prevent the misuse of cryptographic functionality.

An off-card entity may obtain information on Security Domain key(s) with a GET DATA command for Key Information Template (tag 'E0').

A Security Domain manages keys as follows:

- A Key Identifier and Key Version Number uniquely reference each key within the on-card entity. In other words, each combination Key Identifier / Key Version Number identifies a unique key slot within that entity.
- Adding a key equates to allocating a new key slot with a new key value, a new Key Identifier, or a new Key Version Number.
- Replacing a key involves updating the key slot with a new key value and the associated Key Version Number. The Key Identifier remains the same. The previous key shall no longer be available.

The off-card key management system shall be aware of the scheme used to identify keys held by the on-card entity. Key Identifiers and Key Version Numbers may have arbitrary values for the card (e.g. not being sequential, not starting with '01') and these values may vary from one key management scheme to the next. Off-card key management schemes are outside the scope of this specification.

Any velocity checking related to a particular keys usage e.g. key try counters and limits are dependent on the Security Domain provider's security policy.

The Security Domain shall store all the Key Information supplied in the PUT KEY command in association with each key.

7.5.2 Key Access Conditions

The Access Conditions assigned to a Security Domain key are either:

- Owner only: the Security Domain itself;
- Authorized users other than the owner: e.g. the Security Domain's associated Applications;
- Any authorized users, including the owner: e.g. the Security Domain and its associated Applications.

This version of the Specification defines the following Access Conditions for Security Domain keys, coded on one byte as follows:

- '00': any authorized user, including the owner; this is the Access Condition by default for Secure Channel Protocol Keys, when not explicitly provided in the PUT KEY command;
- '01': the owner only; this is the Access Condition by default for Token and DAP Keys, when not explicitly provided in the PUT KEY command;
- '02': authorized users other than the owner;
- '03' to '7F': Reserved for Future Use by GlobalPlatform;
- '80' to 'FE': Reserved for proprietary use;
- 'FF': not available.

The access control rules applicable to any Security Domain key are enforced as follows:

- In order to use any Security Domain's cryptographic service, the Application requests to the OPEN the reference of a Secure Channel interface; the OPEN identifies the Security Domain associated with the Application and provides the corresponding Secure Channel interface reference to the Application (GlobalPlatform Registry acting as an access control list);
- The Application requests cryptographic services to a Security Domain, via its Secure Channel interface; since the OPEN ensures access to associated Applications only, the Security Domain controls are simplified to enforce the Access Conditions applicable to each of its keys (e.g. rejecting requests for accessing a key with an Access Condition set to '01').

7.6 Data and Key Management

These services relate to the storing of cryptographic keys and data on the card.

Runtime Behavior

On receipt of a data/key management request, the corresponding Security Domain shall manage the data/key according to its own access control rules.

The card Life Cycle State shall not be CARD_LOCKED or TERMINATED.

On receipt of a DELETE [key], PUT KEY or STORE DATA command, the Security Domain performing the data or key management shall apply its own secure communication policy. The Security Domain provider may apply its own key management policy regarding deletion of keys.

8 Global Platform Services

8.1 Global Services Applications

One or more Global Services Applications may be present on the card to provide services to other Applications on the card. Global Services Applications are distinguished from other Applications through a specific privilege: Global Service.

8.1.1 Registering Global Services

During the installation of an Application (see section 9.3.6 -*Card Content Installation*) or during the registry update for an already installed Application (see section 9.4 - *Content Extradition and Registry Update*), an Application may be assigned the Global Service privilege and optionally Global Service Parameters comprising one (or more) service name(s). When present, each service name identifies a service family and optionally a specific service identifier within that service family, identifying which service(s) the Global Services Application offers to other Applications. When present, the Global Service Parameters shall be recorded in the GlobalPlatform Registry entry for the Application. Uniqueness on the card of this (these) service name(s) is not required in order to allow more than one Global Services Application to offer similar services.

To offer a unique service onto a card, a Global Services Application may register a Global Service with a service name that is unique on that card. A Global Services Application may register one or more Global Service(s) with unique service name(s). OPEN is responsible for ensuring the uniqueness of each service name registered by Global Services Applications.

Runtime Behavior

The following runtime behavior requirements apply to the OPEN during the registration process of a Global Service with a unique service name. On receipt of unique service registration request, the OPEN shall:

- Check that the requesting on-card entity has the Global Service privilege;
- If one (or more) service name(s) are recorded for that on-card entity, check that the requested service name matches exactly with (one of) the service name(s) recorded for that on-card entity, or belongs to the same service family if the recorded service name(s) only identifies(y) service family(ies);
- Check that the requested service name is not registered within the GlobalPlatform Registry for another on-card entity;
- Register accordingly as unique the requested service name.

The following runtime behavior requirements apply to the OPEN during the deregistration process of a Global Service with a unique service name. On receipt of service deregistration request, the OPEN shall:

- Check that the requesting on-card entity has the Global Service privilege;
- Check that the requested service name is registered in the GlobalPlatform Registry entry of the requesting on-card entity;
- Deregister accordingly the requested service name as being unique.

8.1.2 Application Access to Global Services

GlobalPlatform API is the interface Applications use to access Global Services. An Application may access a uniquely registered Global Service or a specific Global Services Application.

Runtime Behavior

The following runtime behavior requirements apply to the OPEN during the access of a uniquely registered Global Service or a specific Global Services Application. On receipt of service access request, the OPEN shall:

- If the request indicates a specific service name without any associated AID, check that the requested service name matches exactly with (one of) the service name(s) uniquely registered, or belongs to the same service family uniquely registered;
- If the request indicates a specific AID, check that the on-card entity identified in the request has the Global Service privilege, and that the requested service name matches exactly with (one of) the service name(s) recorded for that on-card entity, or belongs to (one of) the same service family(ies) recorded for that on-card entity;
- Identify the corresponding Global Services Application;
- Obtain the GlobalPlatform Service interface of the corresponding Global Services Application and forward it to the requesting on-card entity.

The requesting on-card entity can then directly invoke the GlobalPlatform Service interface of the corresponding Global Services Application and authenticate itself in order to obtain the requested service.

8.1.3 Global Service Parameters

Global Service Parameters may be defined for an Application and list one or more service names. Each service name is coded on two bytes. The first byte identifies the service family, the second the service id within that family. The following values are assigned to the service family identifier:

- '00' to '7F' – Reserved for proprietary use and not registered by GlobalPlatform,
- '80' to '9F' – Reserved for use by GlobalPlatform
  - '80' – Not available,
  - '81' – GlobalPlatform Secure Channel,
  - '82' – GlobalPlatform CVM;
- 'A0' to 'FE' – Reserved for use by individual schemes registered by GlobalPlatform;
  - 'A0' – USSM;
- 'FF' – Not available;

The values assigned to the service id within a service family are specific to each service family.

- A service id value of '00' is typically reserved to indicate ‘any service id within that family’; or may be used as a default value in the case where the service family does not require a service identifier;
- For the GlobalPlatform Secure Channel family, the Secure Channel Protocol identifiers (see 10.7 - *Secure Channel Protocol Identifier*) are the assigned service id values for that family;
- For the GlobalPlatform CVM family, the CVM identifier values (see section 8.2.2.1 - *Registering CVM*) are the assigned service id values for that family.

8.2 CVM Application

The CVM Application, if present on a card, provides a mechanism for a Cardholder Verification Method (CVM), including velocity checking, that may be used by all Applications on the card. In this version of the Specification there is one CVM standardized by GlobalPlatform: the global Personal Identification Number (global PIN).

The CVM verification services may be accessed by any Application. CVM management services shall only be accessible to privileged on-card Applications. The CVM application may also offer CVM management services to suitably authenticated off-card entities through an APDU interface that is beyond the scope of this specification.

A CVM Application may support multiple CVMs, and there may be more than one CVM Application present on a card. Each CVM shall have a unique CVM identifier, which shall be unique across the whole card. The standardized CVM (global PIN), if present on a card, shall be given the CVM identifier of '11'.

For each CVM supported, the CVM Application shall:

- Hold securely CVM management data: CVM value, CVM State, CVM Retry Limit, and CVM Retry Counter;
- Perform CVM-specific risk management, such as internal velocity checks on the CVM to prevent card and Application access violations;
- Implement one or more CVM interfaces;
- Request the OPEN to register the CVM identifier if that CVM is unique;
- Provide the CVM services to Applications;
- Manage the CVM state;
- Manage the CVM Retry Limit and CVM Retry Counter.

Depending on the Issuer policy, a value for the Retry Limit may be set by default.

8.2.1 Application Access to CVM Services

The following CVM services shall be provided by a CVM Application to other on-card Applications:

- Retrieving the CVM state (e.g. to determine if the CVM value has been submitted, verified or blocked);
- Retrieving the number of remaining times the CVM value can be incorrectly presented prior to the CVM being blocked;
- Setting a new value for the CVM value. This depends on the requesting Application having the CVM Management privilege;
- Verifying the content of an incoming CVM value by comparing the incoming CVM value to the stored CVM value;
- Setting the maximum number of times the CVM value can be incorrectly presented prior to the CVM being blocked. This depends on the requesting Application having the CVM Management privilege.

8.2.2 CVM Management

8.2.2.1 Registering CVM

In order to offer unique CVM services that are accessible by other Applications, a CVM Application shall register with the OPEN the CVM identifier(s) for which it offers services. The OPEN shall ensure that the registered CVM identifiers are unique on the whole card. Reuse is only possible if the CVM identifier is deregistered or the corresponding CVM Application is deleted.

The following values are assigned to CVM identifiers:

- '00' - Not available;

- '01' to '7F' - Reserved for use by GlobalPlatform;
  - '01' to '10' - PIN as defined in ETSI TS 102 221 specification;
  - '11' - Global PIN;
  - '12' to '1F' - PIN as defined in ETSI TS 102 221 specification;
- '80' to 'EF' - Reserved for use by individual schemes registered by GlobalPlatform;
  - '81' to '9F' - PIN as defined in ETSI TS 102 221 specification;
- 'F0' to 'FF' - Reserved for proprietary use and not registered by GlobalPlatform.

8.2.2.2 CVM States

The CVM state may be used by a CVM Application to assist in managing CVM services. The non-atomic states of the CVM may be seen within a Card Session. The CVM state, the Retry Limit, and the Retry Counter are closely related. All CVM state transitions are immediately visible to the Application that caused the transition as well as to any Applications that may be selected on other logical channels.

The CVM states are:

- ACTIVE
- INVALID_SUBMISSION
- VALIDATED
- BLOCKED

8.2.2.2.1 CVM State ACTIVE

The CVM state shall first become ACTIVE when both the CVM value and the Retry Limit are set. The CVM state may also transition back to ACTIVE from any other CVM state if a privileged Application unblocks the CVM or changes the CVM value. Changing the CVM value shall also reset the Retry Counter. At the end of a Card Session the CVM state shall transition back to ACTIVE, except if the CVM state transitioned to the CVM state BLOCKED during the Card Session. The end of the Card Session shall not reset the Retry Counter.

8.2.2.2.2 CVM State INVALID_SUBMISSION

During the CVM verification, the CVM state shall transition to INVALID_SUBMISSION if the CVM verification fails. The Retry Counter shall be updated.

The CVM state shall remain INVALID_SUBMISSION until:

- The Card Session ends;
- A valid CVM verification is performed;
- An Application resets the CVM state;
- A privileged Application either blocks the CVM or changes the CVM value or CVM Retry Limit;
- Subsequent CVM verification(s) fail causing the CVM state to transition to BLOCKED.

8.2.2.2.3 CVM State VALIDATED

During CVM verification, the CVM state shall transition to VALIDATED and the Retry Counter shall be reset if the CVM verification is successful.

The CVM state shall remain VALIDATED until:

- The Card Session ends;
- An invalid CVM verification is performed;
- An Application resets the CVM state;
- A privileged Application either blocks the CVM, or changes the CVM value or CVM Retry Limit.

8.2.2.2.4 CVM State BLOCKED

During CVM verification, if the CVM verification fails and the Retry Limit has been reached, the CVM state shall transition to BLOCKED. The CVM state may also transition to BLOCKED if a privileged Application initiates this transition. The BLOCKED state shall not transition when the Card Session ends. The CVM state may only transition from the BLOCKED state back to the ACTIVE state on instruction from a privileged Application, which either resets (unblocks) the CVM state or changes the CVM value or CVM Retry Limit. The CVM verification shall always fail in the BLOCKED state.

8.2.2.3 CVM Format

The 3 following formats are defined for the CVM value:

- Format BCD includes only numerical digits, coded on a nibble (4 bits), left justified, and eventually padded on the right with an 'F' nibble if necessary (i.e. the number of digits is odd);
- Format ASCII includes all displayable characters (alphabetic, numerical, and special) and space (i.e. format ASCII ranges from ‘20’ to ‘7E’), coded on one byte and left justified.
- Format HEX is equivalent to a transparent mode (“as is”) and includes all binary values coded on one byte.

The following rules apply for the CVM format conversion:

- No conversion from and to HEX format is valid;
- Conversion from BCD format to ASCII format is valid: the numeric nibbles are expanded to the corresponding characters coded on one byte and the padding nibble 'F' is deleted (if present);
- Conversion from ASCII format to BCD format is valid for numeric characters only: the numeric characters coded on one byte are converted to numeric nibbles, padded together in bytes, and a padding nibble 'F' is added on the right if necessary.

The internal format for storing the CVM value by a CVM Application is implementation dependent and shall be transparent to any other on-card Application that uses CVM services. It shall not preclude any format used by Applications requesting CVM services.

9 Card and Application Management

9.1 Card Content Management

9.1.1 Overview

Card Content management on a GlobalPlatform card is the capability for the loading, installation, extradition, registry update and removal of Card Content. GlobalPlatform is designed for providing maximum flexibility to the Card Issuer and its business partners regarding Card Content management. The design of the GlobalPlatform takes into account the possibility that the Card Issuer may not necessarily want to manage all Card Content changes, especially when the Card Content does not belong to the Card Issuer.

Thus the Card Issuer may delegate Card Content management to an Application Provider with or without authorization:

- It may authorize all Card Content management operations performed by an Application Provider;
- It may authorize an Application Provider to have full control of its Card Content;
- It may authorize an Application Provider to isolate its own Security Domain(s) and Application(s) from other Application Providers, and potentially from the Card Issuer itself.

Card Content changes are permitted according to the privileges that have been assigned to the various Security Domains on the card.

The following sections describe the OPEN and Security Domain requirements to support the operation and authorization of Card Content management.

9.1.2 OPEN Requirements

The OPEN:

- Performs the physical loading and installation;
- May prohibit more than one Card Content management operation occurring concurrently;
- Prohibits Card Content management in the card Life Cycle States CARD_LOCKED or TERMINATED.

9.1.3 Security Domain Requirements

The Security Domain through which Card Content management is performed applies its own secure communication policy, which should be checked for consistency with any Card Issuer policies before the Security Domain is installed.

Card Content management involves a Security Domain performing loading, installation, extradition, update to GlobalPlatform Registry and content removal operations.

9.1.3.1 Security Domain with Token Verification Privilege

This privilege allows a Security Domain Provider, typically the Card Issuer, to authorize any Card Content management operation. A Security Domain with Token Verification privilege requires the knowledge of keys and algorithms used for Tokens.

Note that this privilege does not provide Card Content management capability.

In this version of the specification, one Security Domain with Token Verification privilege per card is assumed.

9.1.3.2 Security Domain with Authorized Management Privilege

Having a Security Domain with this privilege allows a Security Domain provider to perform Card Content management without authorization (i.e. a token) in the case where the off-card entity is authenticated as the owner (Security Domain Provider) of the Security Domain. In that case the Security Domain that has Token Verification privilege is not involved. However, a Token is still required if the off-card entity is authenticated but is not the Security Domain Provider (see section 10.4 - *Entity Authentication*).

9.1.3.3 Security Domain with Delegated Management Privilege.

The Delegated Management privilege allows an Application Provider's Security Domain with this privilege to perform:

- Delegated loading;
- Delegated installation and make selectable;
- Delegated extradition;
- Delegated update to the GlobalPlatform Registry;
- Delegated deletion.

The privilege allows an Application Provider to manage Card Content with authorization. The Security Domain that has Token Verification privilege controls such authorization.

Delegated Management is not a mandated feature of a GlobalPlatform card and is only necessary for Card Issuers that choose to offer this flexibility. In order to achieve it, close co-operation is required between the OPEN and the Security Domains.

9.1.3.4 Security Domain with Global Delete Privilege

This privilege provides the capability to remove any Executable Load File or Application from the card even if the Executable Load File or Application does not belong to this Security Domain.

Note that a Security Domain without Global Delete privilege and with Card Content management capability can only delete Executable Load Files or Applications directly or indirectly associated with it.

9.1.3.5 Security Domain with Global Lock Privilege

This privilege provides the right to initiate the locking and unlocking of any Application on the card, independent of its Security Domain association and hierarchy. It also provides the capability to restrict the Card Content Management functionality of OPEN.

9.1.3.6 Security Domain with Receipt Generation Privilege

This privilege allows a Security Domain Provider, typically the Card Issuer, to provide a confirmation for the performed card content management. A Security Domain with Receipt Generation privilege requires the knowledge of keys and algorithms used for Receipts generation. It shall also keep track of a Confirmation Counter that is incremented when generating each Receipt. When reaching its maximum value, the Confirmation Counter shall not be reset to zero. Cards are not required to support counter values beyond 32767.

Note that this privilege does not provide Card Content management capability.

In this version of the specification, one Security Domain with Receipt Generation privilege per card is assumed.

9.2 Authorizing and Controlling Card Content

The following section details the authorization and control features that may be used during Card Content loading and installation. The responsibility of ensuring that these controls are present when required rests with the OPEN.

9.2.1 DAP Verification

An Application Provider may require that their Application code to be loaded on the card shall be checked for integrity and authenticity. The DAP Verification privilege of the Application Provider's Security Domain detailed in this Specification provides this service on behalf of an Application Provider.

A Controlling Authority may require that all Application code to be loaded onto the card shall be checked for integrity and authenticity. The Mandated DAP Verification privilege of the Controlling Authority's Security Domain detailed in this Specification provides this service on behalf of the Controlling Authority.

The key and algorithm to be used for DAP Verification or Mandated DAP Verification are implicitly known by the corresponding Security Domain.

Flow control and runtime behavior as described in section 9.3.5 - *Card Content Loading Process* applies; more detail on DAP Blocks is provided in appendix C.3.

9.2.2 Load File Data Block Hash

The Load File Data Block Hash is an integrity check across the whole Load File Data Block to be transferred to the card and is present as a field in the INSTALL [for load] command; more detail is provided in appendix C.2. The Load File Data Block Hash is mandatory when a Token or DAP Block is present in a Load File, and is optional otherwise. If a Load File Data Block Hash is present in a Load File, then it shall be checked.

See appendix B.2.1 - *Secure Hash Algorithm (SHA-1)* for more details on the hash algorithm.

Flow control and runtime behavior as described in section 9.3.5 - *Card Content Loading Process* applies.

9.2.3 Tokens

Tokens relate specifically to Delegated Management, and to Authorized Management where the off-card entity is not the Security Domain Provider. They are not allowed in other cases. More detail is provided in appendix C.4. The entity owning the Security Domain with Token Verification Privilege provides a Token to the Security Domain Provider performing the content management function. During the processing of the content management function the token is verified on-card by the Security Domain with Token Verification Privilege.

9.3 Card Content Loading, Installation and Make Selectable

9.3.1 Overview

The GlobalPlatform Card Content loading process is designed to allow the addition of code to Mutable Persistent Memory in the card. Card Content loading may be prohibited if a load process is already in progress on another logical channel.

The GlobalPlatform Card Content installation process is designed to allow the Card Issuer to make previously loaded application code executable on the card.

Card Content installation is either performed simultaneously with the load process, immediately following the load process or at a later time.

The Load File Data Block contains the information required in order to create an Executable Load File. The internal organization of the Load File Data Block is beyond the scope of this Specification. The Java Card™ CAP file definition and MULTOS™ Application Load Unit are examples of an expected Load File Data Block. The Load File Data Block may also contain information on the Load File Data Block attributes such as its name, version number and size. The Card Content loading and installation process may include implementation specific linking and actual verification of the executable code. Additional authentication data may also be present in the Load File.

Upon the successful completion of the Card Content loading, an Executable Load File shall be present on the card and the OPEN shall create an entry in the GlobalPlatform Registry for the Executable Load File. The OPEN shall also create an entry in the GlobalPlatform Registry for each Executable Module present within an Executable Load File. However, Executable Modules are not yet ready for execution. Applications may then be installed. Installing an Application results in another entry in the GlobalPlatform Registry.

Figure 9-1 details the two possible phases of the Card Content loading and installation.

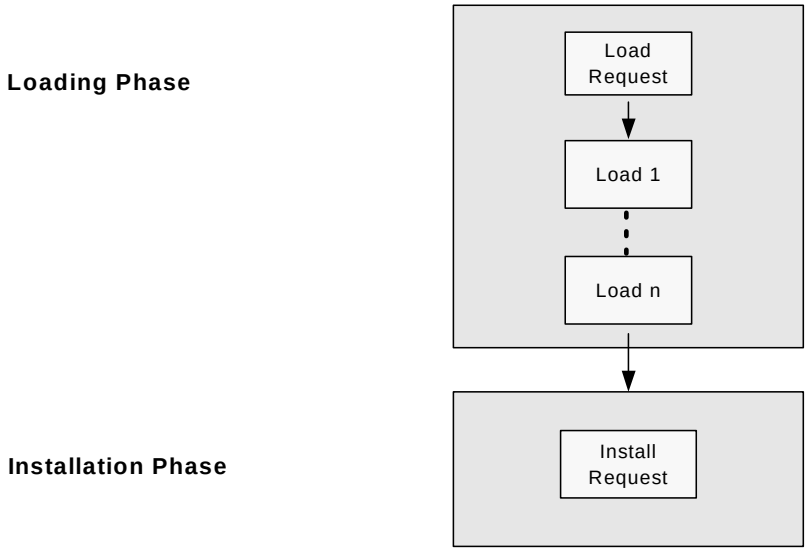

Figure 9-1: Loading and Installation Process

9.3.2 Card Content Loading

When requested to do so by the OPEN, the Security Domain with Token Verification privilege shall verify the transmission of the Load File from the off-card entity to the card, and when applicable, Security Domains with the relevant privilege shall verify the integrity of the Load File Data Block before the OPEN commits the new content to memory.

A Security Domain with Authorized Management or Delegated Management privilege may load an Executable Load File to any Security Domain. The Executable Load File is subject to acceptance by the receiving Security Domain where applicable.

The load process comprises an INSTALL [for load] command and one or more LOAD commands all of which are processed by the Security Domain. The Security Domain then passes the load request and Load File information to the OPEN for additional verification and processing.

The Load Token allows the OPEN, via the Security Domain with Token Verification privilege, to ensure that the Token Issuer authorized the load process and the loading of the content of the Load File Data Block. (In this version of the specification, the Token Issuer is assumed to be the same as the Card Issuer.)

The Load File Data Block Hash links the Token to the actual Load File Data Block.

The response to the last LOAD command identifies the end of the load process. Following the completion of the load process, an optional Load Receipt is returned to the Security Domain performing the Delegated Management operation and shall be transmitted by the Security Domain to the off-card entity.

The Application Provider may then forward the Load Receipt to the corresponding off-card entity as a proof that the loading process was successfully performed. The purpose of the optional Load Receipt is to assist the Card Issuer in keeping its Card Management System synchronized with its card base.

9.3.3 Card Content Installation

A Security Domain with Authorized Management or Delegated Management privilege may install an Application in the Security Domain that the Executable Load File is associated with. If the Security Domain performing the installation is not the Security Domain the Executable Load File is associated with, then this is referred to as implicit extradition of the Application and is subject to the rules in section 9.4 - *Content Extradition and Registry Update*.

The installation process comprises an INSTALL [for install] command processed by the Security Domain. The Security Domain then passes the install request to the OPEN for additional verification and processing.

The Install Token allows the OPEN, via the Security Domain with Token Verification privilege, to ensure that the Card Issuer authorized the installation process.

The response to the INSTALL [for install] command identifies the end of the installation process. Following the completion of the installation process, an optional Install Receipt is returned to the Security Domain performing the Delegated Management operation and shall be transmitted by the Security Domain to the off-card entity.

The Application Provider may then forward the Install Receipt to the corresponding off-card entity as a proof that the installation process was successfully performed. The purpose of the optional Install Receipt is to assist the Card Issuer in keeping its Card Management System synchronized with its card base.

9.3.4 Card Content Combined Loading, Installation and Make Selectable

When requested to do so by the OPEN, the Security Domain with Token Verification privilege shall verify the transmission of the Load File from the off-card entity to the card, and when applicable, Security Domains with the relevant privilege shall verify the integrity of the Load File Data Block before the OPEN commits the new content to memory.

A Security Domain with Delegated Management privilege may load an Executable Load File to any Security Domain, install an Application from the Executable Load File and make the Application selectable.

The combined load, install and make selectable process comprises a first INSTALL [for load, install and make selectable] command, one or more LOAD commands and a last INSTALL [for load, install and make selectable] command all of which are processed by the Security Domain. The Executable Load File is subject to acceptance by the receiving Security Domain where applicable.

The combined Load, Install and Make Selectable Token allows the OPEN, via the Security Domain with Token Verification privilege, to ensure that the Token Issuer authorized the load process and the loading of the content of the Load File Data Block as well as the installation of an Application from the previously loaded Executable Load File. (In this version of the specification, the Token Issuer is assumed to be the same as the Card Issuer.)

The response to the last INSTALL [for load, install and make selectable] command identifies the end of the combined load and install process. Following the completion of the load and install process, an optional combined

Load, Install and Make Selectable Receipt is returned to the Security Domain performing the Delegated Management operation and shall be transmitted by the Security Domain to the off-card entity.

The Application Provider may then forward the combined Load, Install and Make Selectable Receipt to the corresponding off-card entity as a proof that the loading process was successfully performed. The purpose of the optional combined Load, Install and Make Selectable Receipt is to assist the Card Issuer in keeping its Card Management System synchronized with its card base.

The response to the last INSTALL [for load, install and make selectable] command completes the combined load, install and make selectable process.

9.3.5 Card Content Loading Process

The phases in Figure 9-1 use a combination of multiple occurrences of two different APDU commands (INSTALL and LOAD). The following sequence of APDU commands apply to the loading:

- An INSTALL [for load] command serves as the load request for loading. The INSTALL [for load] command data field details the requirements regarding a Load File;
- Multiple LOAD commands are then used to transport the Load File in blocks according to the size of the file and the communications buffer size of the card;
- Each INSTALL or LOAD command is processed by the receiving Security Domain before forwarding the load request and Load File to the OPEN for processing.

The following runtime behavior requirements apply during the content loading process.

Load Request Runtime Behavior

On receipt of an INSTALL [for load] command, the Security Domain performing the load shall:

- Apply its own secure communication policy;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);
- If a Load File Data Block Hash is present in the INSTALL [for load] command, request the OPEN to initiate the hash verification of the subsequent Load File Data Block;
- If the Security Domain performing the load has Delegated Management privilege, check that a Load Token is present in the INSTALL [for load] command;
- If the Security Domain performing the load has the Authorized Management privilege and the off-card entity at the origin of the load request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that a Load Token is present in the INSTALL [for load] command;
- If a Token is present in the INSTALL [for load] command, request the OPEN to obtain verification of the Load Token;
- If the Application Provider identifier is present in the load request, request the OPEN to save this in the GlobalPlatform Registry for the Executable Load File.

On receipt of a load request (arising from an INSTALL [for load] command), the OPEN shall:

- Check that the card Life Cycle State is not CARD LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for load;
- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;

- Check that the AID of the Load File is not already present in the GlobalPlatform Registry as an Executable Load File or Application;
- If an associated Security Domain AID is present, check that this AID exists within the GlobalPlatform Registry and is registered with the Security Domain privilege. As this equates to the extradition of the Load File, if the Security Domain performing the load is not directly or indirectly associated with the associated Security Domain, check that the associated Security Domain accepts this extradition. If no associated Security Domain AID is indicated, the Security Domain performing the load is by default the associated Security Domain;
- At the request of the Security Domain performing the load, request the Security Domain with Token Verification privilege to authorize the load request (e.g. to verify the Load Token).

At the request of OPEN, a Security Domain with Token Verification privilege shall:

- Verify the Load Token.

At the request of OPEN, a Security Domain accepting an implicit extradition shall:

- Apply the Security Domain Provider's policy to accept or reject this implicit extradition;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);

Load Phase Runtime Behavior

On receipt of the LOAD commands, the Security Domain performing the load shall:

- Apply its own secure communication policy;
- Discover whether any Security Domain has the Mandated DAP Verification privilege and if so:
  - Ensure that the required authentication data (DAP Block identifying the above Security Domain) is present in the Load File
- Check if the associated Security Domain has the DAP Verification privilege and if so:
  - Ensure that the required authentication data (DAP Block identifying the associated Security Domain) is present in the Load File;
- If authentication data (one or more DAP Blocks) is present in the Load File:
  - Ensure that a Load File Data Block Hash was received during the load request process;
  - Extract the authentication data (one or more DAP Blocks) from the Load File;
  - For each DAP Block of the Load File, request the OPEN to obtain verification of the DAP by the Security Domain indicated in the DAP Block.

On receipt of the Load File the OPEN shall:

- Verify the resource requirements of the Load File (see section 9.7 - *Memory Resource Management*) and that sufficient card resources are available;
- Check that each DAP verification request from the Security Domain performing the load relates to a Security Domain present in the GlobalPlatform Registry with DAP or Mandated DAP Verification privilege and if so request the Security Domain to verify the DAP;
- Compute the hash of the Load File Data Block when verification of a DAP Block or Load Token is requested.

At the request of OPEN, the Security Domain(s) verifying the DAP(s) shall:

- Verify that the DAP matches with the Load File Data Block Hash received in the load request.

Load Completion Runtime Behavior

On receipt of the last LOAD command, the Security Domain performing the load shall:

- Apply its own secure communication policy;
- Request the OPEN to obtain a Load Receipt.

On completion of the load process the OPEN shall:

- At the request of the Security Domain performing the load, verify the Load File Data Block Hash received in the load request;
- Check in the GlobalPlatform Registry if any Security Domain has the Mandated DAP Verification privilege and if so:
  - Ensure that the above Security Domain has successfully verified a DAP;
- Check in the GlobalPlatform Registry if the associated Security Domain has the DAP Verification privilege and if so:
  - Ensure that the associated Security Domain has successfully verified a DAP;
- If one or more DAP verifications were performed, verify the Load File Data Block Hash received in the load request;
- If the Security Domain performing the load has Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a Token;
- If a Load Token was verified, verify the Load File Data Block Hash received in the load request;
- Create an Executable Load File using the Load File Data Block;
- Create an entry in the GlobalPlatform Registry for the Executable Load File indicating its associated Security Domain;
- Create an entry for each Executable Module within the Executable Load File in the GlobalPlatform Registry. This shall include the Application Provider identifier if requested by the Security Domain in the load request. The associated Security Domain for each Executable Module shall be the same as the associated Security Domain for the Executable Load File;
- At the request of the Security Domain performing the load, request the Security Domain with Receipt Generation privilege to generate a Load Receipt.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not a Load Receipt.

If, at any stage, the OPEN determines that card resources are insufficient for the loading process or that any verification step has failed, the OPEN shall terminate the loading process, shall return the appropriate error and shall reclaim any memory allocated to the load process.

9.3.6 Card Content Installation Process

An INSTALL [for install] command is then used to request the installation of an application. The receiving Security Domain processes the INSTALL command before forwarding the install request to the OPEN for processing.

The installation internal processing is beyond the scope of this Specification. However, it is assumed that the installation process includes the creation of instances and allocation of Application data memory.

Following the installation, the OPEN shall register additional information in the GlobalPlatform Registry regarding the Application Life Cycle State, Security Domain association, and Privileges.

Applications inherit the associated Security Domain of the Executable Load File from which they are installed. They may be extradited to another Security Domain.

The following runtime behavior requirements apply during the content installation process:

Runtime Behavior (Installation)

On receipt of the INSTALL [for install] command, the Security Domain performing the installation shall:

- Apply its own secure communication policy;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);
- If the Security Domain performing the installation has the Authorized Management privilege and the off-card entity at the origin of the installation request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that an Install Token is present in the INSTALL [for install] command.
- If a Token is present in the INSTALL [for install] command, request the OPEN to obtain verification of the Install Token.
- If the Application Provider identifier is present in the install request, request the OPEN to save this in the GlobalPlatform Registry for the Application.
- Request the OPEN to obtain an Install Receipt.

On receipt of the install request, the OPEN shall:

- Check that the card Life Cycle State is not CARD_LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for installation;
- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;
- Check that the Executable Module AID is present in the GlobalPlatform Registry;
- Check that the Application AID (for future selection of the Application) is not already present in the GlobalPlatform Registry as an Application or Executable Load File;
- Check that the Security Domain performing the installation is the Security Domain associated with the Executable Module from which the Application is being installed; or if the Security Domain performing the installation is not the Security Domain directly or indirectly associated with the Executable Module from which the Application is being installed, check that the Security Domain associated with the Executable Load File accepts this installation;
- At the request of the Security Domain performing the installation, request the Security Domain with Token Verification privilege to authorize the installation (e.g. to verify the Install Token);
- Verify the resource requirements indicated for the Application (see section 9.7 - *Memory Resource* Management) and that sufficient card resources are available;
- Perform the installation of the Application according to the underlying runtime environment requirements;
- If the Security Domain performing the installation has Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a Token;
- Create an Application from the Executable Module;

- Ensure that the Application, depending on the underlying runtime environment, has the knowledge of its AID, its Privileges and its Install Parameters;
- Create an entry in the GlobalPlatform Registry for the Application indicating its associated Security Domain, Life Cycle State, Privileges and, when present in the install request, Implicit Selection, Service and Memory Resource Management parameters; and including the Application Provider identifier if supplied by the Security Domain in the installation request;
- At the request of the Security Domain performing the installation, request the Security Domain with Receipt Generation privilege to generate an Install Receipt.

At the request of OPEN, the Security Domain with Token Verification privilege shall:

- Verify the Install Token.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not an Install Receipt.

If the OPEN determines that card resources are insufficient for installing the Application or the runtime environment does not currently allow the installation, the OPEN shall terminate the installation process, return the appropriate error and reclaim any memory allocated to the install process.

9.3.7 Card Content Make Selectable Process

An INSTALL [for make selectable] command is used to request to make selectable a previously installed application. The receiving Security Domain processes the INSTALL command before forwarding the make selectable request to the OPEN for processing.

The make selectable internal processing is beyond the scope of this Specification.

Following the make selectable, the OPEN shall register additional information in the GlobalPlatform Registry regarding the Application Life Cycle State and Privileges.

The following runtime behavior requirements apply during the content make selectable process:

Runtime Behavior (Make Selectable)

On receipt of the INSTALL [for make selectable] command, the Security Domain making the Application selectable shall:

- Apply its own secure communication policy;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);
- If the Security Domain making the Application selectable has the Authorized Management privilege and the off-card entity at the origin of the installation request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that a Make Selectable Token is present in the INSTALL [for make selectable] command.
- If a Token is present in the INSTALL [for make selectable] command, request the OPEN to obtain verification of the Make Selectable Token.
- If the Application Provider identifier is present in the make selectable request, request the OPEN to save this in the GlobalPlatform Registry for the Application.
- Request the OPEN to obtain a Make Selectable Receipt.

On receipt of the make selectable request, the OPEN shall:

- Check that the card Life Cycle State is not CARD_LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for make selectable;
- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;
- Check that the Application AID is present in the GlobalPlatform Registry;
- Check that the Security Domain making the Application selectable is the Security Domain directly or indirectly associated with the Application;
- At the request of the Security Domain making the Application selectable, request the Security Domain with Token Verification privilege to authorize the make selectable (e.g. to verify the Make Selectable Token);
- Make the Application selectable according to the underlying runtime environment requirements;
- If the Security Domain making the Application selectable has Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a Token;
- Update accordingly the GlobalPlatform Registry entry for the Application (e.g. Privileges, Implicit Selection parameters);
- At the request of the Security Domain making the Application selectable, request the Security Domain with Receipt Generation privilege to generate a Make Selectable Receipt.

At the request of OPEN, the Security Domain with Token Verification privilege shall:

- Verify the Make Selectable Token.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not a Make Selectable Receipt.

9.3.8 Card Content Combined Loading, Installation and Make Selectable Process

The phases in Figure 9-1 are combined into a single process that uses a combination of multiple occurrences of two different APDU commands (INSTALL and LOAD). The following sequence of APDU commands apply to the loading:

A first INSTALL [for load, install and make selectable] command serves as the combined load and install request for loading and installation. The INSTALL [for load, install and make selectable] command data field details the requirements regarding a Load File.

Multiple LOAD commands are then used to transport the Load File in blocks according to the size of the file and the communications buffer size of the card.

A last INSTALL [for load, install and make selectable] command serves as the combined load and install commit for loading and installation. The last INSTALL [for load, install and make selectable] command data field details the requirements regarding a Load File.

Each INSTALL or LOAD command is processed by the receiving Security Domain before forwarding the load request and Load File Data Block to the OPEN for processing.

The following runtime behavior requirements apply during the content combined loading and installation process.

Combined Load, Install and Make Selectable Request Runtime Behavior

On receipt of an INSTALL [for load, install and make selectable] command, the Security Domain performing the combined load and install shall:

- Apply its own secure communication policy;

- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);
- If a Load File Data Block Hash is present in the INSTALL [for load, install and make selectable] command, request the OPEN to initiate the hash verification of the subsequent Load File Data Block;
- If the Application Provider identifier is present in the load request, request the OPEN to save this in the GlobalPlatform Registry for the Executable Load File.

On receipt of a combined load and installation request (arising from an INSTALL [for load, install and make selectable] command), the OPEN shall:

- Check that the card Life Cycle State is not CARD LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for load, installation and make selectable;
- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;
- Check that the AID of the Load File is not already present in the GlobalPlatform Registry as an Executable Load File or Application;
- If an associated Security Domain AID is present, check that this AID exists within the GlobalPlatform Registry and is registered with the Security Domain privilege. As this equates to the extradition of the Load File, if the Security Domain performing the combined load, install and make selectable is not directly or indirectly associated with the associated Security Domain, check that the associated Security Domain accepts this extradition. If no associated Security Domain AID is indicated, the Security Domain performing the load is by default the associated Security Domain.

At the request of OPEN, a Security Domain accepting an implicit extradition shall:

- Apply the Security Domain Provider's policy to accept or reject this implicit extradition;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain).

Load Phase Runtime Behavior

On receipt of the LOAD commands, the Security Domain performing the load shall:

- Apply its own secure communication policy;
- Discover whether any Security Domain has the Mandated DAP Verification privilege and if so:
  - Ensure that the required authentication data (DAP Block identifying the above Security Domain) is present in the Load File.
- Check if the associated Security Domain has the DAP Verification privilege and if so:
  - Ensure that the required authentication data (DAP Block identifying the associated Security Domain) is present in the Load File.
- If authentication data (one or more DAP Blocks) is present in the Load File:
  - Ensure that a Load File Data Block Hash was received during the combined load, install and make selectable request process;
  - Extract the authentication data (one or more DAP Blocks) from the Load File;
  - For each DAP Block of the Load File, request the OPEN to obtain verification of the DAP by the Security Domain indicated in the DAP Block.

On receipt of the Load File the OPEN shall:

- Verify the resource requirements of the Load File (see section 9.7- *Memory Resource Management*) and that sufficient card resources are available;
- Check that each DAP verification request from the Security Domain performing the load relates to a Security Domain present in the GlobalPlatform Registry with DAP or Mandated DAP Verification privilege and if so request the Security Domain to verify the DAP;
- Compute the hash of the Load File Data Block when verification of a DAP Block or a combined Load, Install and Make Selectable Token is requested;

At the request of OPEN, the Security Domain(s) verifying the DAP(s) shall:

- Verify that the DAP matches with the Load File Data Block Hash received in the load request.

On completion of the load process the OPEN shall:

- At the request of the Security Domain performing the load, verify the Load File Data Block Hash received in the combined load, install and make selectable request;
- Check in the GlobalPlatform Registry if any Security Domain has the Mandated DAP Verification privilege and if so:
  - Ensure that the above Security Domain has successfully verified a DAP;
- Check in the GlobalPlatform Registry if the associated Security Domain has the DAP Verification privilege and if so:
  - Ensure that the associated Security Domain has successfully verified a DAP;
- If one or more DAP verifications were performed, verify the Load File Data Block Hash received in the load request.

Combined Load, Install and Make Selectable Completion Runtime Behavior

On receipt of the last INSTALL [for load, install and make selectable] command, the Security Domain performing the combined load and install shall:

- Apply its own secure communication policy;
- If the Security Domain performing the installation has the Authorized Management privilege and the off-card entity at the origin of the installation request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that a Load, Install and Make Selectable Token is present in the INSTALL [for load, install and make selectable] command;
- If a Token is present in the INSTALL [for load, install and make selectable] command, request the OPEN to obtain verification of the Load, Install and Make Selectable Token;
- If the Application Provider identifier is present in the combined load, install and make selectable request, request the OPEN to save this in the GlobalPlatform Registry for the Executable Load File and the Application;
- Request the OPEN to obtain a combined Load, Install and Make Selectable Receipt.

On completion of the load, install and make selectable process the OPEN shall:

- If the Security Domain performing the combined load, install and make selectable has Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a Token;
- Create an Executable Load File using the Load File Data Block;

- Create an entry in the GlobalPlatform Registry for the Executable Load File indicating its associated Security Domain;
- Create an entry for each Executable Module within the Executable Load File in the GlobalPlatform Registry. This shall include the Application Provider identifier if requested by the Security Domain in the combined load, install and make selectable request. The associated Security Domain for each Executable Module shall be the same as the associated Security Domain for the Executable Load File;
- Check that the Application AID (for future selection of the Application) is not already present in the GlobalPlatform Registry as an Application or Executable Load File;
- Perform the installation of the Application according to the underlying runtime environment requirements;
- Create an Application from the Executable Module;
- Ensure that the Application, depending on the underlying runtime environment, has the knowledge of its AID, its Privileges and its Install Parameters;
- Create an entry in the GlobalPlatform Registry for the Application indicating its associated Security Domain, Life Cycle State, Privileges and, when present in the combined load, install and make selectable request, Implicit Selection, Service and Memory Resource Management parameters; and including the Application Provider identifier if supplied by the Security Domain in the combined load, install and make selectable request;
- At the request of the Security Domain performing the combined load, install and make selectable, request the Security Domain with Token Verification privilege to generate a Load, Install and Make selectable Receipt;
- Verify the resource requirements indicated for the Application (see section 9.7- *Memory Resource Management*) and that sufficient card resources are available.

At the request of OPEN, the Security Domain with Token Verification privilege shall:

- Verify the combined Load, Install and Make Selectable Token.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not a combined Load, Install and Make Selectable Receipt.

If, at any stage, the OPEN determines that card resources are insufficient for the loading process or that any verification step has failed, the OPEN shall terminate the loading process, shall return the appropriate error and shall reclaim any memory allocated to the load process.

9.3.9 Examples of Loading and Installation Flow

The following diagram is an example of the loading and installing of an Application to a GlobalPlatform card. In this example loading and installation are performed by a Security Domain with Authorized Management privilege. The Load File is loaded on the card and stored in memory as an Executable Load File. The installation phase occurs immediately following the loading phase.

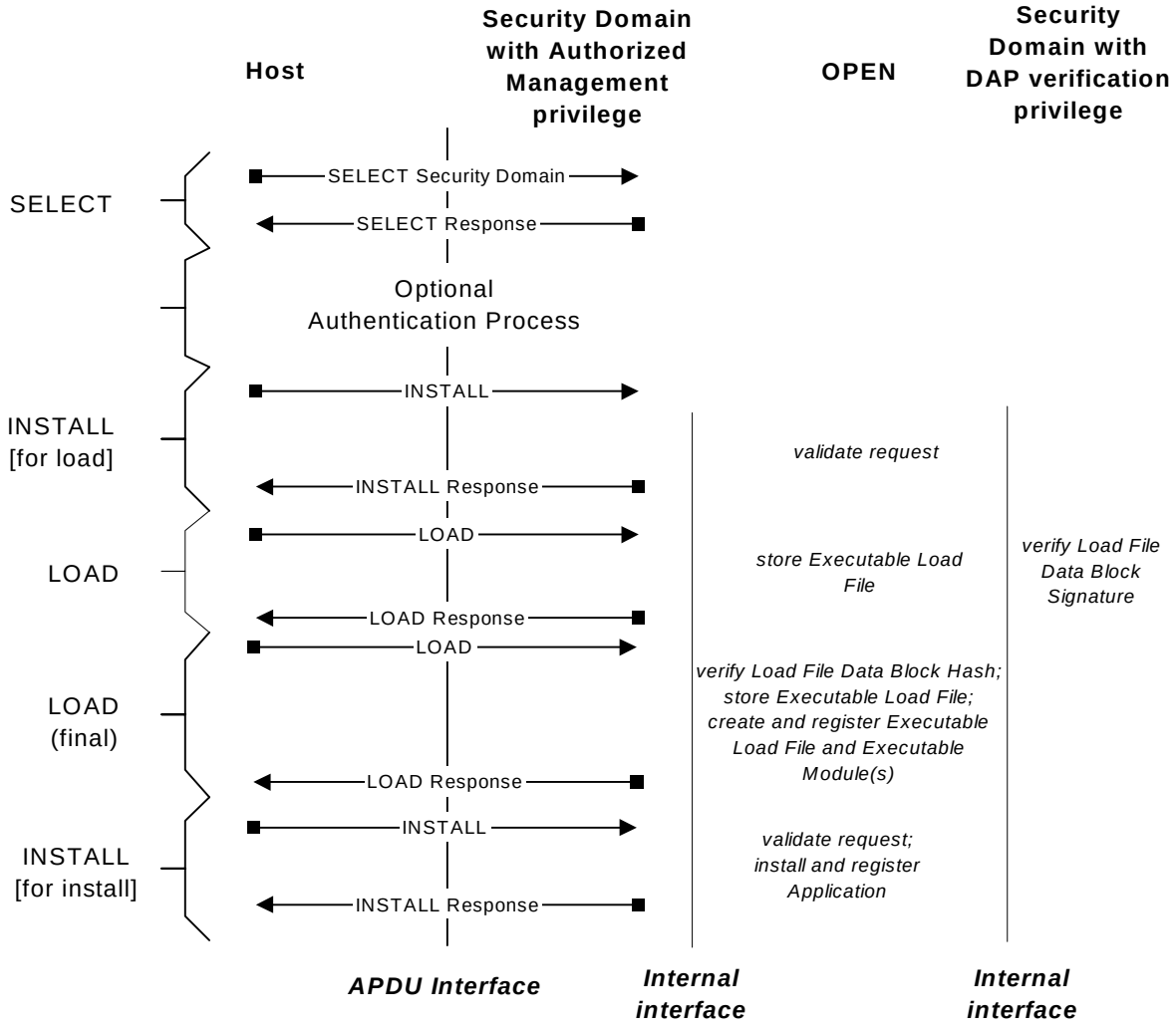

Figure 9-2: Load and Installation Flow Diagram

The following diagram is an example of the loading of an Application to a GlobalPlatform card. In this example loading is performed by a Security Domain with Delegated Management privilege.

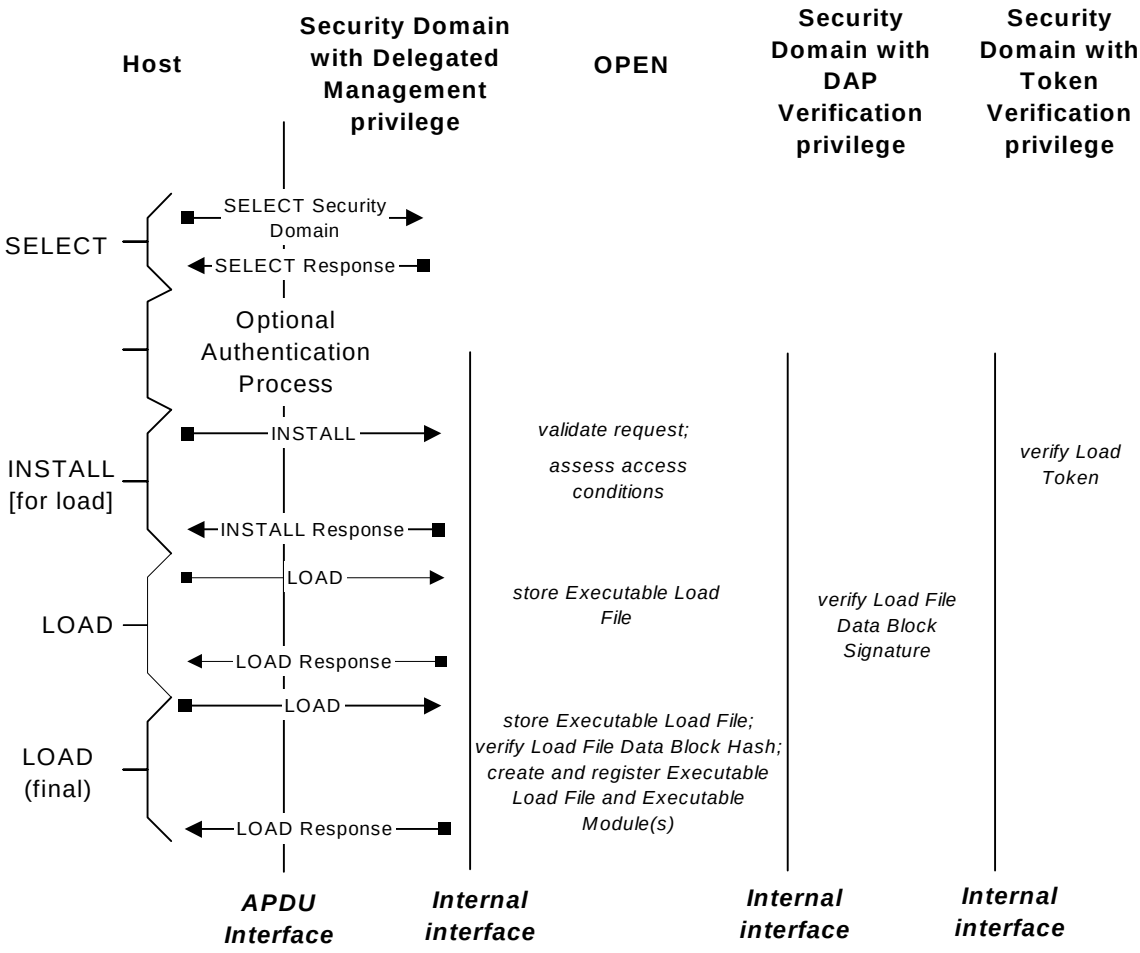

Figure 9-3: Load Flow Diagram

The following diagram is an example of installing an Application from an Executable Load File already present on the card. In this example loading and installation are performed by a Security Domain with Delegated Management privilege.

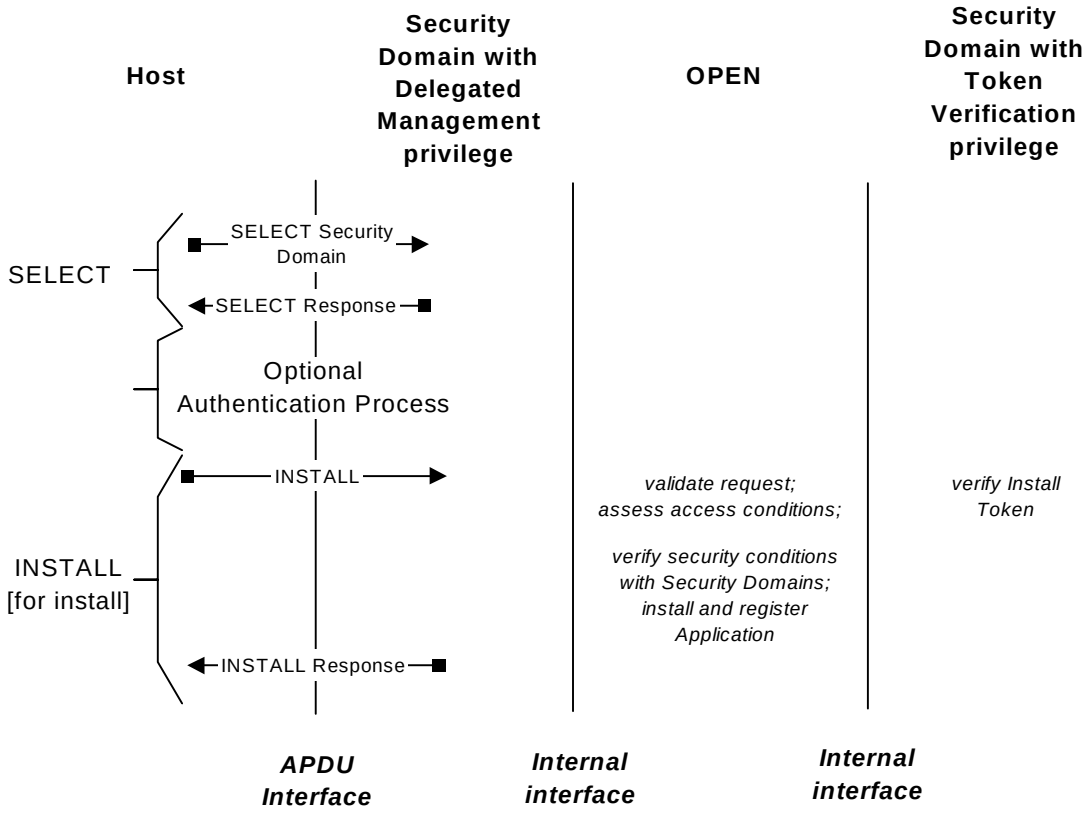

Figure 9-4: Install Flow Diagram

9.4 Content Extradition and Registry Update

9.4.1 Content Extradition

The GlobalPlatform Card Content extradition process is designed to allow a previously installed Application or a previously loaded Executable Load File to be associated with a different Security Domain.

Delegated Extradition allows an Application Provider to extradite one of its Applications to another Security Domain, if the current Security Domain has Delegated Management privilege.

The extradition may apply at any time during the Application Life Cycle. Extradition may apply for any Security Domain (in the PERSONALIZED state) or the Issuer Security Domain (in any card Life Cycle State other than CARD_LOCKED and TERMINATED), that accepts the extradited Application.

The extradition process comprises an INSTALL [for extradition] command processed by the receiving Security Domain. The Security Domain then passes the extradition request to the OPEN for additional verification and processing.

The Extradition Token allows the OPEN, via the Security Domain with Token Verification privilege, to ensure that the Card Issuer authorized the extradition process.

The response to the INSTALL [for extradition] command identifies the end of the extradition process. Following the completion of the extradition process, an optional Extradition Receipt is returned to the Security Domain performing the Delegated Management operation and shall be transmitted by the Security Domain to the off-card entity.

The Application Provider may then forward the Extradition Receipt to the corresponding off-card entity as a proof that the extradition process was successfully performed. The purpose of the optional Extradition Receipt is to assist the Card Issuer in keeping its Card Management System synchronized with its card base.

The following runtime behavior requirements apply during the Card Content extradition process.

Runtime Behavior

On receipt of the INSTALL [for extradition] command, the Security Domain performing the extradition shall:

- Apply its own secure communication policy;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);
- If the Security Domain performing the extradition has the Authorized Management privilege and the off-card entity at the origin of the extradition request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that an Extradition Token is present in the INSTALL [for extradition] command;
- If a Token is present in the INSTALL [for extradition] command, request the OPEN to obtain verification of the Extradition Token;
- Request the OPEN to obtain an Extradition Receipt.

On receipt of an extradition request, the OPEN shall:

- Check that the card Life Cycle State is not CARD_LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for extradition;
- Determine if the Application or Executable Load File being extradited exists within the GlobalPlatform Registry;
- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;
- Check that the Security Domain requesting the extradition is directly or indirectly associated with the Application or Executable Load File being extradited;
- Check that an on-card entity with the same AID as the Security Domain to which the Application or Executable Load File is being extradited exists within the GlobalPlatform Registry, and that this on-card entity has the Security Domain privilege;
- If the Security Domain performing the extradition is not directly or indirectly associated with the Security Domain to which the Application or Executable Load File is being extradited, check that this Security Domain accepts this Card Content extradition;
- If the Security Domain performing the extradition has the Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a Token;
- Update accordingly the GlobalPlatform Registry entry for the Application or Executable Load File;
- At the request of the Security Domain performing the extradition, request the Security Domain with Receipt Generation privilege to generate an Extradition Receipt.

At the request of OPEN, the Security Domain with Token Verification privilege shall:

- Verify the Extradition Token.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not an Extradition Receipt.

At the request of OPEN, the Security Domain accepting the explicit extradition shall:

- Apply the Security Domain Provider's policy to accept or reject this Card Content extradition;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain).

Extradition Flow

The following figure is an example of extradition, and shows delegated extradition:

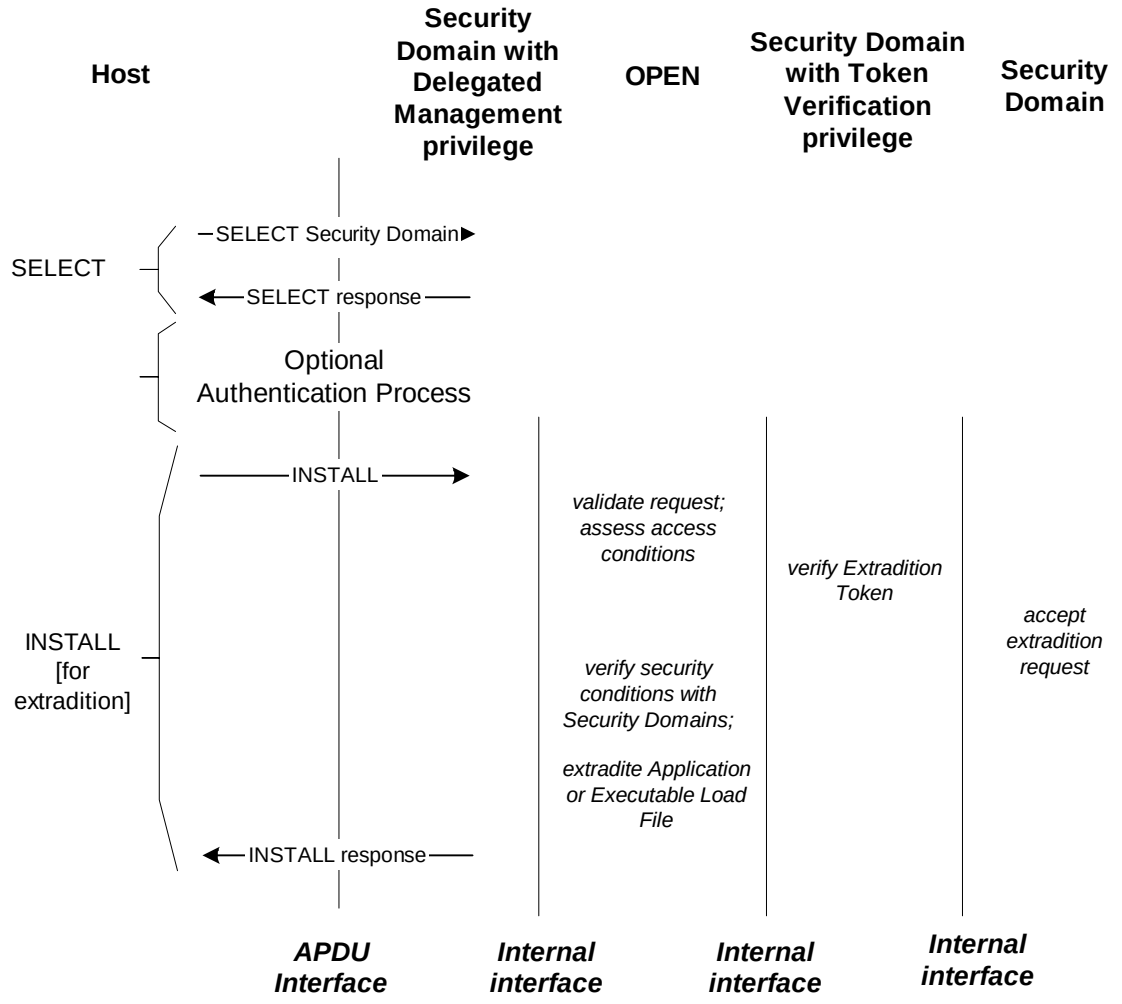

Figure 9-5: Delegated Extradition Flow

9.4.2 Registry Update

9.4.2.1 Generic Registry Update

The registry update process allows GlobalPlatform Registry data associated with an Application, such as Privileges and Implicit Selection parameters, to be modified. This process also allows the restricting of Card Content Management functionality of a specific Security Domain or OPEN itself (i.e. of all existing Security Domains present on the card and any eventual Security Domain installed afterwards).

The registry update may apply at any time during the Application Life Cycle or card Life Cycle (other than CARD_LOCKED or TERMINATED).

The registry update process comprises one or more INSTALL [for registry update] commands processed by the receiving Security Domain. To restrict Card Content Management functionality of OPEN, no Application AID is provided in the INSTALL [for registry update] command. The Security Domain then passes the registry update request to the OPEN for additional verification and processing.

The Registry Update Token allows the OPEN, via the Security Domain with Token Verification privilege, to ensure that the Card Issuer authorized the update of the GlobalPlatform Registry.

The response to the INSTALL [for registry update] command identifies the end of the registry update process. Following the completion of the registry update process, an optional Registry Update Receipt is returned to the Security Domain performing the Delegated Management operation and shall be transmitted by the Security Domain to the off-card entity.

The Application Provider may then forward the Registry Update Receipt to the corresponding off-card entity as a proof that the registry update process was successfully performed. The purpose of the optional Registry Update Receipt is to assist the Card Issuer in keeping its Card Management System synchronized with its card base.

The following runtime behavior requirements apply during the registry update process.

Runtime Behavior

On receipt of the INSTALL [for registry update] command, the Security Domain performing the registry update shall:

- Apply its own secure communication policy;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);
- If the Security Domain performing the registry update has the Authorized Management privilege and the off-card entity at the origin of the registry update request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that a Registry Update Token is present in the INSTALL [for registry update] command;
- If a Token is present in the INSTALL [for registry update] command, request the OPEN to obtain verification of the Registry Update Token;
- Request the OPEN to obtain a Registry Update Receipt.

On receipt of a registry update request, the OPEN shall:

- Check that the card Life Cycle State is not CARD_LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for registry update;
- When updating an Application, determine if the Application being updated exists within the GlobalPlatform Registry;

- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;
- When restricting functionality of OPEN, check that the requesting on-card entity is a Security Domain with Global Lock privilege;
- Check that the Security Domain requesting the registry update is directly or indirectly associated with the Application being updated;
- If the Security Domain performing the registry update has the Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a Token;
- Update accordingly the GlobalPlatform Registry entry for the Application being updated;
- At the request of the Security Domain performing the registry update, request the Security Domain with Receipt Generation privilege to generate a Registry Update Receipt.

At the request of OPEN, the Security Domain with Token Verification privilege shall:

- Verify the Registry Update Token.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not a Registry Update Receipt.

9.4.2.2 Extradition using Registry Update

An extradition may be performed using the registry update process. The simultaneous extradition and registry update process is achieved by an appropriately formed INSTALL [for registry update] command.

The runtime behavior requirements for an extradition using registry update are the sum of the runtime behavior requirements for extradition and the runtime behavior requirements for registry update.

If a Token is required the Registry Update Token is used, and if a Receipt is to be returned the Registry Update Receipt is used; both of which allow for extradition as well as for registry update.

9.5 Content Removal

This section defines the content removal process that enables a flexible management of the Mutable Persistent Memory space of the cards through the removal of Applications and/or Executable Load Files. Only code and data not referenced by another entity on the card may be deleted. Code and data that is currently being accessed by the runtime environment is deemed to be referenced by another entity. If the Application is selected on another logical channel this Application cannot be deleted.

The DELETE command (see section 11.2 - *DELETE Command*) allows for the actual removal of content from Mutable Persistent Memory and the logical removal of content from Immutable Persistent Memory. The DELETE command is processed by the receiving Security Domain before forwarding the removal request to the OPEN for processing.

Depending on the memory location of the removed Applications and/or Executable Load File, the OPEN shall perform the different actions detailed in the following sections. When the Application or Executable Load File has been successfully removed, its contents in the memory shall no longer be accessible.

A Security Domain with Global Delete privilege has the privilege to delete any Application or Executable Load File from the card regardless of which Security Domain the Application or Executable Load File is associated with.

The Application Provider may instruct the OPEN to delete its associated Applications and Executable Load Files. According to the Card Issuer's policies, the OPEN may require pre-authorization from the Card Issuer, i.e. a Delete Token may be required.

The response to the DELETE command identifies the end of the deletion process. Following the completion of the deletion process, an optional Delete Receipt is returned to the Security Domain performing the Delegated Management operation and shall be transmitted by the Security Domain to the off-card entity.

The Application Provider may then forward the Delete Receipt to the corresponding off-card entity as a proof that the deletion process was successfully performed. The purpose of the optional Delete Receipt is to assist the Card Issuer in keeping its Card Management System synchronized with its card base.

Deletion Flow

The following figure is an example of deleting an Application, an Executable Load File, or both:

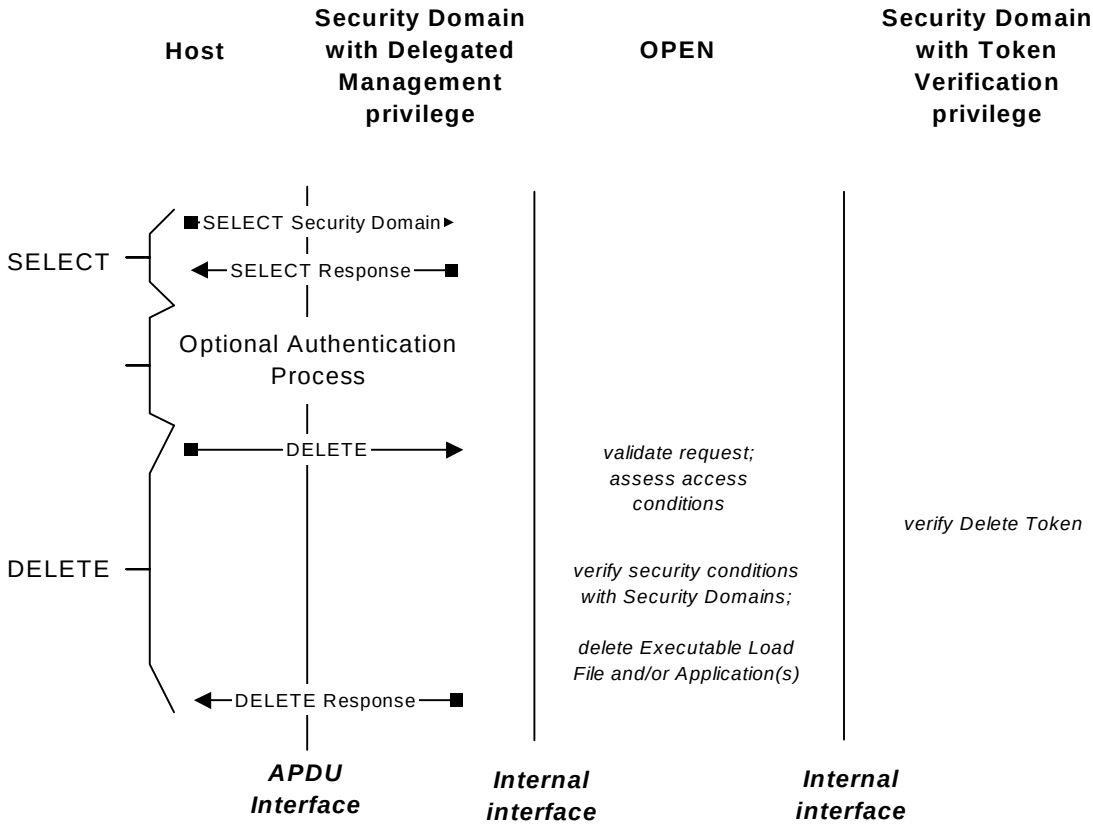

Figure 9-6: Content Deletion Flow

9.5.1 Application Removal

Application removal may involve the removal of Application instances as well as any Application data associated with the Application.

The following runtime behavior requirements apply during the Application removal process.

Runtime Behavior

On receipt of an Application deletion request (DELETE command), the Security Domain performing the deletion shall:

- Apply its own secure communication policy;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (applicable to a Security Domain other than the Issuer Security Domain);
- If the Security Domain performing the deletion has the Authorized Management privilege and the off-card entity at the origin of the delete request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that a Delete Token is present in the DELETE command;
- If a Token is present in the DELETE command, request the OPEN to obtain verification of the Delete Token;
- Request the OPEN to obtain a Delete Receipt.

On receipt of an Application deletion request, the OPEN shall:

- Check that the card Life Cycle State is not CARD_LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for deletion;
- Determine that the Application being deleted has an entry within the GlobalPlatform Registry;
- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;
- Check that the Security Domain performing the delete is the Security Domain directly or indirectly associated with the Application being deleted, or that it has Global Delete privilege. Otherwise (i.e. if the Security Domain performing the deletion is not directly or indirectly associated with the Application being deleted and does not have the Global Delete privilege), check that the Security Domain associated with the Application accepts this deletion;
- At the request of the Security Domain performing the deletion, request the Security Domain with Token Verification privilege to verify the Delete Token;
- Determine that the Application is not currently selected on another logical channel;
- Determine that no other Applications present in the card make references to this Application;
- Determine that no other Applications present in the card make references to any data within this Application;
- If a Security Domain is being deleted, determine that none of the Applications or Executable Load Files present in the card are associated with this Security Domain;
- If the Security Domain performing the deletion has the Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a token;
- Remove the entry for the Application from within the GlobalPlatform Registry;
- Re-assign the Application's privileges (if any) to the Issuer Security Domain as defined in section 6.6.2 - *Privilege Assignment*;

- If the Application is implicitly selectable, re-assign implicit selection to the Issuer Security Domain as defined in section 6.4.1 - *Implicit Selection Assignment*;
- Release and mark as available any Mutable Persistent Memory and when supported, apply the memory resource management rules described in section 9.7 - *Memory Resource Management*;
- At the request of the Security Domain performing the deletion, request the Security Domain with Receipt Generation privilege to generate a Delete Receipt.

If the OPEN determines that any of the above verification steps have failed, the OPEN shall not initiate the delete process and shall inform the Security Domain to return the appropriate response. Once this delete process begins it shall complete in the current Card Session or, in the event of an interruption, at least the updates to the GlobalPlatform Registry shall complete in a subsequent Card Session.

At the request of OPEN, the Security Domain with Token Verification privilege shall:

- Apply the issuer's policy to accept or reject a deletion authorization request without the presence of a Delete Token;
- Verify the Delete Token.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not a Delete Receipt.

9.5.2 Executable Load File Removal

An Executable Load File contains Executable Modules. The removal applies to the entire Executable Load File. Physical removal may occur in Mutable Persistent Memory while only logical removal is possible in Immutable Persistent Memory.

This version of the Specification does not cover the removal of a specific Executable Module within an Executable Load File.

The following runtime behavior requirements apply during the Executable Load File removal process.

Runtime Behavior

On receipt of an Executable Load File deletion request (DELETE command), the Security Domain performing the deletion shall:

- Apply its own secure communication policy;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);
- If the Security Domain performing the deletion has the Authorized Management privilege and the off-card entity at the origin of the delete request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that a Delete Token is present in the DELETE command;
- If a Token is present in the DELETE command, request the OPEN to obtain verification of the Delete Token;
- Request the OPEN to obtain a Delete Receipt.

On receipt of an Executable Load File removal process deletion request, the OPEN shall:

- Check that the card Life Cycle State is not CARD_LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for deletion;

- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;
- Check that the Security Domain performing the delete is the Security Domain directly or indirectly associated with the Executable Load File being deleted, or that it has Global Delete privilege. Otherwise (i.e. if the Security Domain performing the deletion is not directly or indirectly associated with the Executable Load File being deleted and does not have the Global Delete privilege) check that the Security Domain associated with the Executable Load File accepts this deletion;
- At the request of the Security Domain performing the deletion, request the Security Domain with Token Verification privilege to verify the Delete Token;
- Determine that the Executable Load File being deleted has an entry within the GlobalPlatform Registry;
- Determine that no other Applications and no other Executable Load Files present in the card maintain references to this Executable Load File;
- If the Security Domain performing the deletion has the Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a Token;
- Remove the entry for the Executable Load File and the entries for any Executable Modules present in the Executable Load File from within the GlobalPlatform Registry;
- Release and mark as available any Mutable Persistent Memory and when supported, apply the memory resource management rules described in section 9.7 - *Memory Resource Management*;
- At the request of the Security Domain performing the deletion, request the Security Domain with Receipt Generation privilege to generate a Delete Receipt.

If the OPEN determines that any of the above verification steps have failed, the OPEN shall not initiate the delete process and shall inform the Security Domain to return the appropriate response. Once this delete process begins it shall all complete in the current Card Session or, in the event of an interruption, at least the updates to the GlobalPlatform Registry shall complete in a subsequent Card Session.

Only Mutable Persistent Memory is released and marked as available. Executable Load Files contained in Immutable Persistent Memory cannot be deleted but the entry for the Executable Load File and the entries for the Executable Modules present in the Executable Load File shall be deleted from the GlobalPlatform Registry.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to accept or reject a deletion authorization request without the presence of a Delete Token;
- Verify the Delete Token.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not a Delete Receipt.

9.5.3 Executable Load File and related Application Removal

An Executable Load File contains Executable Modules from which Applications have been installed. This optional feature removes the Executable Load File as well as all these related Applications. Physical removal may occur in Mutable Persistent Memory while only logical removal is possible in Immutable Persistent Memory.

When supported by the card, the following runtime behavior requirements apply during the Executable Load File and related Application removal process.

Runtime Behavior

On receipt of an Executable Load File deletion request (DELETE command), the Security Domain performing the deletion shall:

- Apply its own secure communication policy;
- Apply its own security policy, e.g. check that its Life Cycle State is PERSONALIZED (only applicable to a Security Domain other than the Issuer Security Domain);
- If the Security Domain performing the deletion has the Authorized Management privilege and the off-card entity at the origin of the delete request is not authenticated as its Security Domain Provider (see section 10.4 - *Entity Authentication*), check that a Delete Token is present in the DELETE command;
- If a Token is present in the DELETE command, request the OPEN to obtain verification of the Delete Token;
- Request the OPEN to obtain a Delete Receipt.

On receipt of a request to remove an Executable Load File and its related Applications, the OPEN shall:

- Check that the card Life Cycle State is not CARD_LOCKED or TERMINATED;
- Check that OPEN and the requesting on-card entity have no restriction for deletion;
- Check that the requesting on-card entity is a Security Domain with Delegated Management or Authorized Management privilege;
- Check that the Security Domain performing the delete is the Security Domain directly or indirectly associated with each of the related Applications being deleted, or that it has Global Delete privilege. Otherwise (i.e. if the Security Domain performing the deletion is not directly or indirectly associated with one or more of the Applications being deleted and does not have the Global Delete privilege) check in each case that the Security Domain associated with the related Application accepts this deletion;
- Check that the Security Domain performing the delete is the Security Domain directly or indirectly associated with the Executable Load File being deleted, or that it has Global Delete privilege. Otherwise (i.e. if the Security Domain performing the deletion is not directly or indirectly associated with the Executable Load File being deleted and does not have the Global Delete privilege) check that the Security Domain associated with the Executable Load File accepts this deletion;
- At the request of the Security Domain performing the deletion, request the Security Domain with Token Verification privilege to verify the Delete Token;
- If the Security Domain performing the deletion has the Delegated Management privilege, ensure that the Security Domain with Token Verification privilege has successfully verified a Token;
- Determine that the Executable Load File and Applications being deleted have entries within the GlobalPlatform Registry;
- Locate each Application installed from an Executable Module within this Executable Load File and for each Application:
  - Determine that the Application is not currently selected on another logical channel;
  - Determine that no other non-related Applications present in the card make reference to this Application;
  - Determine that no other non-related Applications present in the card maintain references to any data within this Application;
  - If a Security Domain is being deleted, determine that none of the non-related Applications present in the card are associated with this Security Domain;

- Determine that no other Applications and no other Executable Load Files present in the card maintain references to this Executable Load File;
- Remove the entry for the Executable Load File and the entries for any Executable Modules present in the Executable Load File from within the GlobalPlatform Registry;
- Remove the entry for each related Application within the GlobalPlatform Registry;
- Re-assign the privileges of all related Applications (if any) to the Issuer Security Domain as defined in section 6.6.2 - *Privilege Assignment*;
- If any related Application is implicitly selectable, re-assign implicit selection to the Issuer Security Domain as defined in section 6.4.1 - *Implicit Selection Assignment*;
- Release and mark as available any Mutable Persistent Memory and when supported, apply the memory resource management rules described in section 9.7 - *Memory Resource Management*;
- At the request of the Security Domain performing the deletion, request the Security Domain with Receipt Generation privilege to generate a series of Delete Receipts: one for the Executable Load File and one for each Application.

If the OPEN determines that any of the above verification steps have failed, the OPEN shall not initiate the delete process and shall inform the Security Domain to return the appropriate response. Once the delete processes begin they shall all complete in the current Card Session or, in the event of an interruption, at least the updates to the GlobalPlatform Registry shall complete in a subsequent Card Session.

Only Mutable Persistent Memory is released and marked as available. Executable Load Files contained in Immutable Persistent Memory cannot be deleted but the entry for the Executable Load File and the entries for the Executable Modules present in the Executable Load File shall be deleted from the GlobalPlatform Registry.

At the request of OPEN, the Security Domain with Token Verification privilege shall:

- Apply the issuer's policy to accept or reject a deletion authorization request without the presence of a Delete Token;
- Verify the Delete Token.

At the request of OPEN, the Security Domain with Receipt Generation privilege shall:

- Apply the issuer's policy to generate or not a Delete Receipt.

9.6 Security Management

9.6.1 Life Cycle Management

The following set of services is provided to manage the Life Cycles of on-card entities:

- Application locking and unlocking;
- Card locking and unlocking;
- Card termination;
- Application Life Cycle interrogation;
- Card Life Cycle interrogation.

Life Cycle Management Flow

Figure 9-7 shows the flow that is common to all the Life Cycle management services:

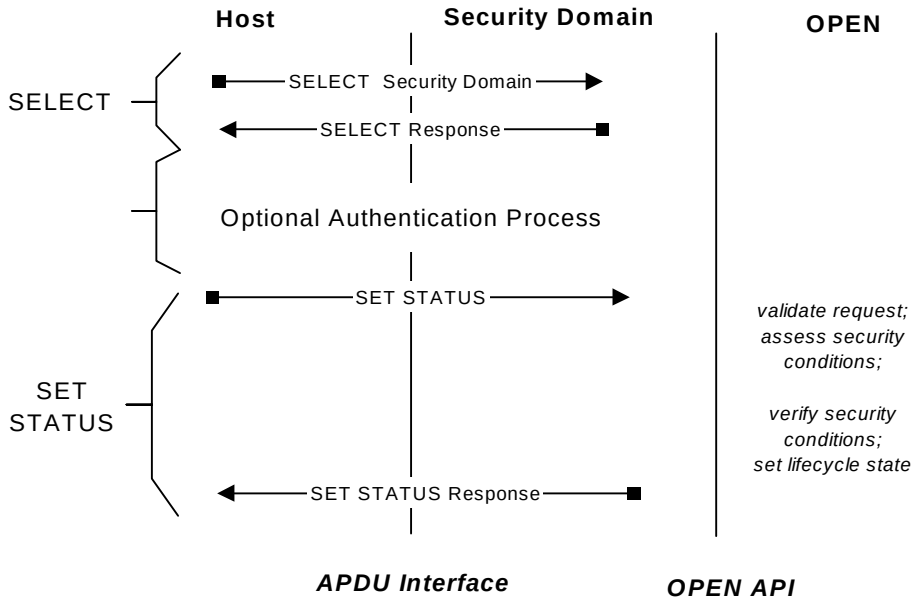

Figure 9-7: Life Cycle Management Flow

9.6.2 Application Locking and Unlocking

The card has a mechanism to disable and subsequently re-enable the continued execution status of an on-card Application. This mechanism may be invoked from within the OPEN based on exceptions handled by the OPEN or from the use of externally invoked commands. An Application with Global Lock privilege, the Application itself or a directly or indirectly associated Security Domain are the only entities that may initiate the locking of an Application. Only an Application with Global Lock privilege or a directly or indirectly associated Security Domain may initiate the unlocking of an Application.

Runtime Behavior

On receipt of a SET STATUS command, the Security Domain performing the lifecycle management shall:

- Apply its own secure communication policy;
- Check that the off-card entity at the origin of the lock/unlock request is authenticated as the Application Provider of the target Application or as the Security Domain Provider (see section 10.4 - *Entity Authentication*).

On receipt of a request to lock or unlock an Application, the OPEN shall:

- Check that the requesting on-card entity is the Application itself, or the Security Domain directly or indirectly associated with the Application being locked or unlocked, or that the requesting entity has the Global Lock privilege;
- Check that the Application is not requesting to unlock itself;
- Check that an entry for the Application being locked or unlocked is present in the GlobalPlatform Registry and does not have the Final Application privilege;

- For locking, set the Application Life Cycle State to LOCKED;
- For unlocking, reverse the Application Life Cycle State to its previous state;
- For locking, keep a record of the previous Application Life Cycle State to ensure that the Application Life Cycle State can be transitioned back to and only to this previous state.

An Application being locked that is currently selected on a logical channel shall remain the currently selected Application on that logical channel until the end of the Application Session though its Application Life Cycle State is set to LOCKED.

Once the Application Life Cycle State is set to LOCKED, the Application shall not be selectable implicitly or explicitly on any logical channel.

If the locked Application is implicitly selectable on any card interface(s) and logical channel(s), the application with Final Application privilege will be implicitly selected for those card interface(s) and logical channel(s).

9.6.3 Card Locking and Unlocking

Once the card Life Cycle State is CARD_LOCKED, all Applications except the Application with the Final Application privilege shall be disabled. Due to the severity of this action, the card locking shall only be allowed under the most stringent conditions and shall only be performed with the proper security mechanisms and authorizations.

Card locking requests may originate from two sources:

Internal source— either the OPEN, the Issuer Security Domain, or a privileged Application may initiate card locking requests from within the card. Internal requests are likely to occur in response to certain card runtime situations. For example, the OPEN or an Application with the necessary privilege may initiate the card locking process as a response to a perceived security threat based on data collected from velocity checking data.

Off-card source— explicit card locking requests may be initiated by APDU commands sent by the Card Issuer to the Issuer Security Domain or by an authorized representative to an Application with the Card Lock Privilege.

Runtime Behavior

On receipt of a SET STATUS command, the Security Domain performing the lifecycle management shall:

- Apply its own secure communication policy;
- Check that the off-card entity at the origin of the lock/unlock request is authenticated as the Security Domain Provider (see section 10.4 - *Entity Authentication*).

On receipt of a request to lock the card, the OPEN shall:

- Check that the requesting on-card entity has the required Card Lock privilege;
- Check that the current card Life Cycle State is SECURED;
- Set the card Life Cycle State to CARD_LOCKED.

On receipt of a request to unlock the card, the OPEN shall:

- Check that the requesting on-card entity has the required Card Lock privilege;
- Reverse the card Life Cycle State to SECURED.

Applications that are currently selected on any logical channel shall remain the currently selected Applications on their respective logical channels until the end of their respective Application Sessions. As soon as the current Application Session on a Supplementary Logical Channel ends, the logical channel on which the Application was selected is closed. Once all the Supplementary Logical Channels are closed the Basic Logical Channel becomes the

only available interface. As soon as the current Application Session on the Basic Logical Channel ends, the Application with the Final Application privilege is the only selectable Application.

Attempts to modify Card Content are denied from the instant the card transitions to the CARD_LOCKED Life Cycle State.

9.6.4 Card Termination

In the card Life Cycle State TERMINATED, all communication to the card are directed to the application with the Final Application privilege. If the Issuer Security Domain is the application with this privilege all commands except the GET DATA command processed by the Issuer Security Domain shall be disabled. Due to the severity of this action, the card termination shall only be allowed under the most stringent conditions and shall only be performed with the proper security mechanisms and authorizations.

The decision to terminate a card may either be triggered by an internal card event that violates a policy of the Card Issuer or may be invoked externally:

Internal source— either the OPEN, the Issuer Security Domain, or a privileged Application may initiate a card termination request from within the card. An internal card termination request is likely to occur in response to certain card runtime situations.

Off-card source— explicit card termination requests can be initiated by APDU commands sent by the Card Issuer to the Issuer Security Domain or by an authorized representative to an Application with the Card Terminate privilege.

Runtime Behavior

On receipt of a SET STATUS command, the Security Domain performing the lifecycle management shall:

- Apply its own secure communication policy;
- Check that the off-card entity at the origin of the termination request is authenticated as the Security Domain Provider (see section 10.4 - *Entity Authentication*).

On receipt of a request to terminate the card, the OPEN shall:

- Check that the requesting on-card entity has the Card Terminate privilege;
- Set the card Life Cycle State to TERMINATED from its current state.

Applications that are currently selected on any logical channel shall remain the currently selected Applications on their respective logical channels until the end of their respective Application Sessions. As soon as the current Application Session on a Supplementary Logical Channel ends, the logical channel on which the Application was selected is closed. Once all the Supplementary Logical Channels are closed the Basic Logical Channel becomes the only available interface. As soon as the current Application Session on the Basic Logical Channel ends, no Applications are selectable, the Application with Final Application privilege is always implicitly selected on the Basic Logical Channel.

As the transition to the TERMINATED Life Cycle State can be due to the detection of a severe security threat, the security policy of the Issuer may require additional behavior such as closing the current Card Session by performing an internal card reset.

Card Content management requests are denied from the instant the card transitions to the TERMINATED Life Cycle State. Depending on the security policy of the Issuer, other operations may be allowed to continue during the remaining current Application Sessions.

9.6.5 Application Status Interrogation

The status of an Application (or a Security Domain): its Life Cycle State, Privileges and other parameters registered in the GlobalPlatform Registry, may be accessed by suitably authorized entities.

Runtime Behavior

On receipt of a GET STATUS command, the Security Domain performing the lifecycle interrogation shall:

- Apply its own secure communication policy;
- Check that the off-card entity at the origin of the request is authenticated as the Application Provider of the interrogated Application or as the Security Domain Provider (see section 10.4 - *Entity Authentication*).

On receipt of a request to interrogate the status of an Application, the OPEN shall:

- Check that the requesting on-card entity is the entity itself being interrogated, a Security Domain directly or indirectly associated with the Application being interrogated, or that the requesting entity has the Global Registry privilege

9.6.6 Card Status Interrogation

The status of the card and the Issuer Security Domain: its AID, Privileges and other parameters registered in the GlobalPlatform Registry, may be accessed by suitably authorized entities.

Runtime Behavior

On receipt of a GET STATUS command, the Security Domain performing the lifecycle interrogation shall:

- Apply its own secure communication policy;
- Check that the off-card entity at the origin of the request is authenticated as the Security Domain Provider (see section 10.4 - *Entity Authentication*).

On receipt of a request to interrogate the card Life Cycle State, the OPEN shall respond.

9.6.7 Operational Velocity Checking

The OPEN shall be implemented with a velocity checking security mechanism. For the purpose of this document, “velocity checking” is defined as the active monitoring, handling and management of security sensitive activities on the card.

These activities may include, but are not limited to the following:

- Content installation;
- Card exceptions;
- Application exceptions.

Typically, velocity checking is used to counter security attacks that use repeated attempts in their schemes. These attempts can be from internal (on-card) or external (off-card) entities. Velocity checking is implemented to track the number of consecutive failures in each of the related management and security events.

9.6.7.1 Content Loading and Installation

The OPEN may keep track of the number of unsuccessful consecutive attempts of the Card Content load and installation process by a particular Application and the total number of such attempts by all Applications. Actions may include such defensive measures as the locking or termination of the card.

9.6.7.2 Exceptions

Velocity checking may be implemented in cases in which the OPEN generates exceptions. However, it does not have to be implemented such that each individual exception is handled separately. A trace buffer and event log may be used to complement velocity checking.

For example, an implementation of a Security Domain may enable velocity checking to enforce strict APDU command sequencing for card and application management operations (e.g., Card Content loading and installation). The OPEN may also enable velocity checking against repeated failed attempts by an Application to allocate additional memory beyond its allowed limit that is stored in the GlobalPlatform Registry. The OPEN may choose to lock an Application that is exhibiting such behavior.

9.6.8 Tracing and Event Logging

Tracing and event logging functions may be enabled. These functions shall be implemented according to a Card Issuer's policy requirements.

9.7 Memory Resource Management

Memory resource management is an optional feature of a GlobalPlatform card. It consists of the management by OPEN of counters associated with Executable Load Files or Applications regarding their individual allocation of memory resources, as defined in this specification. It applies to both persistent and volatile memory. Memory resource management data elements describe memory usage requirements applicable to each type of memory: volatile or persistent; and for each type of storage: code or data. Memory requirements may be defined for each Executable Load File and Application at the time of their load and installation. The OPEN, in conjunction with the Runtime Environment, is responsible for managing the memory resources of the card; including the security requirements defined in section 4.2.4.1 - *Runtime Environment Security Requirements*.

A memory resource management data element indicates an amount of memory resources in bytes. In the INSTALL [for load] command, the System Parameters may include memory requirements specifying for each type of memory a Minimum Memory, or Reserved Memory, or both data elements. In the INSTALL [for install] command, the System Parameters may include memory requirements specifying for each type of memory a Memory Quota, or Reserved Memory, or both.

When memory resource management is supported, the OPEN shall assign, if currently available, each type of memory resources as described in the corresponding memory resource management data element. Memory resources shall be assigned according to the following rules:

- Assigning Minimum Memory to an Executable Load File (and its subsequent Application instance) shall not reduce the memory resources currently available on the card;
- Assigning Memory Quota to an Application shall not reduce the memory resources currently available on the card;
- The value of Memory Quota assigned to an Application shall not be less than the value of its Reserved Memory;
- Assigning Reserved Memory to an Executable Load File / Application shall reduce the memory resources currently available on the card at the time of its successful load / installation.

When memory resource management is supported, the OPEN shall manage available memory resources for each type of memory according to the following rules:

- At the time of the load's request (INSTALL [for load] command), the Minimum Memory requirements shall be currently available on the card;

- At the time of the successful load of an Executable Load File (last LOAD command), the amount of memory effectively allocated to that Executable Load File shall be charged against the Reserved Memory of that Executable Load File. If no Reserved Memory was assigned to that Executable Load File, the amount of memory shall reduce the memory resources currently available on the card;
- At the time of the installation of an Application (INSTALL [for install] command), the amount of memory effectively allocated to that Application shall first be charged against the Reserved Memory of that Application until it is entirely exhausted. When the Reserved Memory (if any) is exhausted, the amount of allocated memory shall be charged against that Application's Memory Quota and shall reduce the memory resources currently available on the card. When either the Memory Quota is exceeded or the memory resources currently available on the card are exhausted, the installation of the Application shall fail;
- The successful removal of an Executable Load File / Application (DELETE command) shall augment the memory resources available on the card by the amount of memory effectively released and any unused part of the Reserved Memory (if any) of that Executable Load File / Application;
- The amount of memory effectively allocated to the creation of data shall first be charged against the Reserved Memory of that Application, until it is entirely exhausted. When the Reserved Memory (if any) is exhausted, the amount of allocated memory shall be charged against that Application's Memory Quota and shall reduce the memory resources currently available on the card. When either the Memory Quota is exceeded or the memory resources currently available on the card are exhausted, the resource allocation shall fail;
- The results of reporting memory resources are implementation dependent;
- The amount of memory effectively released by the deletion of data shall be reallocated to the Reserved Memory and Memory Quota assigned to that Application. If no Reserved Memory was assigned to that Application, the amount of released memory shall augment the memory resources available on the card.

10 Secure Communication

The GlobalPlatform documentation broadly defines the notion of secure communication over and above that defined in ISO standards. Specific mention is made of secure messaging for APDU commands; an optional authentication process is implied in most of the flow diagrams and Application access to Security Domain services implies that the ability to create a Secure Channel Session exists. This chapter defines the general rules that apply to all Secure Channel Protocols. Applications that utilize the services of Security Domains can use the Secure Channel Protocol supported by their associated Security Domains.

These protocols provide a means by which a Security Domain or an Application may communicate with an off-card entity within a logically secure environment.

Logical channels facilitate the possibility of more than one of the above off-card entities communicating concurrently with multiple Applications on the card, each within its own logically secure environment. It is the responsibility of the Security Domain provider to define whether this is possible or not.

10.1 Secure Channel

The Secure Channel provides a secure communication channel between a card and an off-card entity during an Application Session.

A Secure Channel Session is divided into three sequential phases:

- **Secure Channel Initiation** – when the on-card Application and the off-card entity have exchanged sufficient information enabling them to perform the required cryptographic functions. The Secure Channel Session initiation always includes the authentication of the off-card entity by the on-card Application;
- **Secure Channel Operation** – when the on-card Application and the off-card entity exchange data within the cryptographic protection of the Secure Channel Session. The Secure Channel services offered may vary from one Secure Channel Protocol to the other;
- **Secure Channel Termination** – when either the on-card Application or the off-card entity determines that no further communication is required or allowed via an established Secure Channel Session.

The 3 levels of security provided by the Secure Channel are defined in section 4.3.2 - *Secure Communication*.

A further level of security applies to sensitive data (e.g. secret keys) that shall always be transmitted as confidential data.

10.2 Explicit / Implicit Secure Channel

A Secure Channel Session is open after a successful initiation, including the authentication of the off-card entity by the on-card Application. A Secure Channel Session is terminated after the last successful communication within the same Secure Channel Session.

10.2.1 Explicit Secure Channel Initiation

Initiating a Secure Channel Session may be achieved by the off-card entity using appropriate APDU command(s) or by the on-card Application using the appropriate API. This method is the explicit Secure Channel creation. Appropriate APDU commands allow the off-card entity to instruct the card what level of security is required for the current Secure Channel Session (integrity and/or confidentiality). They also give the off-card entity the possibility of selecting the keys to be used.

10.2.2 Implicit Secure Channel Initiation

The Secure Channel Session is initiated by the on-card Secure Channel Protocol handler, directly or through an API, when receiving the first APDU command that contains a cryptographic protection. The Secure Channel Protocol handler implicitly knows the required level of security. The Secure Channel Protocol handler may implicitly know which keys are to be used or may have been previously instructed by the off-card entity using appropriate APDU command(s) prior to initiating the Secure Channel Session.

10.2.3 Secure Channel Termination

Termination of the Secure Channel Session can be achieved by the on-card Application using the appropriate API or by the off-card entity using the appropriate APDU command.

Secure Channel termination causes all session data to be reset and all ICVs and session keys to be erased. The Current Security Level is set to NO_SECURITY and the Session Security Level is reset.

Termination of the Secure Channel Session occurs when one of the following conditions is met:

- The on-card Application Session is terminated - for instance when another Application is selected (i.e. the Application Session ends) on the same logical channel of the same card I/O interface. It may be the responsibility of the Application to initiate the termination of the Secure Channel Session when this occurs;
- The associated logical channel is explicitly closed;
- The card is reset (see ISO/IEC 7816-3 for contact cards), deactivated (see ISO/IEC 14443-3 for contactless cards) or powered off: i.e. the Card Session ends.

A Secure Channel Session is aborted but not terminated in the following cases:

- The on-card Application receives the first APDU command with an erroneous cryptographic protection;
- The on-card Application receives an APDU command without the required cryptographic protection set up during Secure Channel Session initiation;

If a Secure Channel Session is aborted, the Current Security Level is set to NO_SECURITY, the Session Security Level is not reset and the error condition persists until the Secure Channel Session is terminated.

A Secure Channel Session on a logical channel is never terminated in favor of a new Secure Channel Session being initiated on another logical channel.

10.3 Direct / Indirect Handling of a Secure Channel Protocol

There are two ways for the on-card Application to handle the Secure Channel Protocol.

- **Direct** - The Application owns its Secure Channel key set and fully implements the protocol: an example is Security Domains.
- **Indirect** – The Application uses the Security Domain services to handle the Secure Channel Protocol. This allows the Application using these services to be coded independently from the Secure Channel Protocol supported by the card.

10.4 Entity Authentication

Off-card entity authentication is achieved through the process of initiating a Secure Channel Session and provides assurance to the card that it is communicating with an authenticated entity. If any step in the authentication process fails, the process shall be restarted.

Authentication of the off-card entity is only provided for a limited time, a Secure Channel Session, and is valid only for the messages within that Secure Channel. This Secure Channel Session relates to the establishment and termination of a Secure Channel. If the Secure Channel Session is closed for any reason the off-card entity shall no longer be considered authenticated.

10.4.1 Authentication with symmetric cryptography

With a symmetric key Secure Channel Protocol (e.g. SCP01 or SCP02), an authenticated off-card entity is the entity which knows the secret Secure Channel keys needed to initiate a Secure Channel Session. The card cannot distinguish between the actual Application Provider of the Security Domain and one of its agents to whom a (set of) secret Secure Channel key(s) was provided. In this case, the card cannot distinguish between the Security Domain's provider and the provider of one of its associated Applications. Both are assumed to be the same entity: the Application Provider.

The Application is informed whether that unique entity, the Application Provider, has been successfully authenticated or not by examining the AUTHENTICATED indicator of the Current Security Level (see section 10.6 - *Security Levels*).

10.4.2 Authentication with asymmetric cryptography

With an asymmetric key Secure Channel Protocol (e.g. SCP10), any off-card entity that owns a pair of asymmetric keys and obtained a certificate for its public key certified by an authority recognized by the Security Domain can be successfully authenticated by that Security Domain. An authenticated off-card entity is the subject identified in the off-card entity's public key certificate verified during the Secure Channel Initiation. The card can distinguish between the actual Application Provider of the Security Domain and any other off-card entity using the off-card entity public key certificate's subject identifier.

In this case, the card can distinguish between the Security Domain's provider, the provider of one of its associated Applications, and any other off-card entity. The three are not necessarily the same entity: the Application Provider Id of each on-card entity is registered in the GlobalPlatform Registry at the time of the load or installation of that on-card entity and cannot be changed subsequently.

The Application is informed whether its own Application Provider has been authenticated by examining the AUTHENTICATED indicator of the Current Security Level (see section 10.6 - *Security Levels*). The ANY_AUTHENTICATED indicator of the Current Security Level indicates if another off-card entity, including the Security Domain's provider, has been authenticated or if the Application Provider Id is not registered. The Current Security Level viewed by the Target Application may differ from the Security Domain's view depending on the Secure Channel Protocol and the authenticated off-card entity's Application Provider Id (e.g. ANY_AUTHENTICATED for the Target Application and AUTHENTICATED for the Security Domain).

10.5 Secure Messaging

Secure messaging allows a sending entity to add confidentiality and/or integrity and authentication to the composition of an APDU message prior to the message being transmitted to a receiving entity:

- Integrity of an APDU command sent to the card or confidentiality of the command message and integrity of an APDU command sent to the card;
- Integrity of the sequence of APDU commands sent to the card within a Secure Channel Session;
- Optionally, depending on the Secure Channel Protocol, confidentiality and/or integrity of an APDU response message sent by the card, and integrity of the sequence of responses within a Secure Channel Session.

In the context of secure messaging, message integrity also provides data origin authentication.

10.6 Security Levels

A Secure Channel Protocol operates according to the Security Level that is established. This is done at two levels:

- A mandatory minimum Session Security Level is set at the initialization of the Secure Channel Session either explicitly or implicitly;
- A different Current Security Level may be set for an individual command or an individual response.

These Security Levels establish the minimum acceptable secure messaging protection that must be applied to protected messages, either for the whole session or for a specific command-response pair.

The following table shows the coding of Current Security Level:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0	-	-	-	-	-	-	**AUTHENTICATED**
0	1	-	-	-	-	-	-	**ANY_AUTHENTICATED**
-	-	-	-	-	-	1	-	**C_DECRYPTION**
-	-	-	-	-	-	-	1	**C_MAC**
-	-	1	-	-	-	-	-	**R_ENCRYPTION**
-	-	-	1	-	-	-	-	**R_MAC**
-	-	-	-	X	X	-	-	**RFU**
0	0	0	0	0	0	0	0	**NO_SECURITY_LEVEL**

Table 10-1: Current Security Level

Note that the Current Security Level cannot have its AUTHENTICATED and ANY_AUTHENTICATED indicators set simultaneously.

The rules for how Security Levels are handled are defined individually for Secure Channel Protocols '01', '02' and '10': see Appendices D, E and F for details.

10.7 Secure Channel Protocol Identifier

The Secure Channel Protocol identifies which particular secure communication protocol and set of security services are implemented in a Security Domain. The following values are assigned to the Secure Channel Protocol identifier:

- '**00**' – Not available
- '**01**' to '**7F**' - Reserved for use by GlobalPlatform
  - '**01**' – deprecated Secure Channel Protocol 1 defined in appendix D;
  - '**02**' – Secure Channel Protocol 2 defined in appendix E;
  - '**10**' - Secure Channel Protocol '10' defined in appendix F.
- '**80**' **to** '**EF**' – Reserved for use by individual schemes registered by GlobalPlatform
  - '**80**' - Secure Channel Protocol '80' defined in ETSI TS 102 225 and 102 226.
- '**F0**' **to** '**FF**' – Reserved for proprietary use and not registered by GlobalPlatform

The deprecated symmetric key Secure Channel Protocol '01' is backward compatible with the Open Platform Card Specification version 2.0.1'; it is deprecated in this version of the Specification.

The symmetric key Secure Channel Protocol '02' includes services in addition to those provided by Secure Channel Protocol '01' as well as optimizing the operation of some services compared to the deprecated Secure Channel Protocol '01'.

The Secure Channel Protocol '80' supports the Over The Air security scheme defined in ETSI TS 102 225.

The asymmetric key Secure Channel Protocol '10' offers authentication services using a Public Key Infrastructure (PKI) and secure messaging protection of commands and responses using symmetric cryptography.

Part IV
APDU
Command
Reference

11 APDU Command Reference

This chapter details the GlobalPlatform APDU commands that may be implemented. The commands are listed alphabetically.

The following table summarizes the APDU commands that shall be supported by a Security Domain with Authorized Management privilege, a Security Domain with Delegated Management privilege, a Security Domain with Final Application privilege and the requirements for support of these APDU commands by other Security Domains. When logical channels are supported, the MANAGE CHANNEL command is only processed by the OPEN and no Security Domain support is required for this command.

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED		TERMINATED	
	AM SD	DM SD	SD	AM SD	DM SD	SD	AM SD	DM SD	SD	FA SD	SD	FA SD	SD
DELETE Executable Load File													
DELETE Executable Load File and related Application(s)													
DELETE Application	✓			✓			✓						
DELETE Key													
GET DATA	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓		✓	
GET STATUS	✓			✓			✓			✓			
INSTALL [for load]													
INSTALL [for install]													
INSTALL [for load, install and make selectable]													
INSTALL [for install and make selectable]	✓	✓		✓	✓		✓	✓					
INSTALL [for make selectable]													
INSTALL [for extradition]													
INSTALL [for registry update]													

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED		TERMINATED	
	AM SD	DM SD	SD	AM SD	DM SD	SD	AM SD	DM SD	SD	FA SD	SD	FA SD	SD
INSTALL [for personalization]													
LOAD													
PUT KEY	✓			✓			✓						
SELECT	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓			
SET STATUS	✓			✓			✓			✓			
STORE DATA	✓			✓			✓						

Table 11-1: Authorized GlobalPlatform Commands per Card Life Cycle State

Legend of Table 11-1:

AM SD: Security Domain with Authorized Management privilege.

DM SD: Security Domain with Delegated Management privilege.

FA SD: Security Domain with Final Application privilege. (Note: command support for an Application with Final Application Privilege which is not a Security Domain is subject to Issuer policy, within the constraints of what is permitted according to the card Life Cycle State.)

SD: Other Security Domain.

✓ : Support required.

Blank cell: Support optional.

Striped cell: Support prohibited.

Table 11-2 summarizes the minimum security requirements for the APDU commands.

Command	Minimum Security
DELETE	Secure Channel initiation or digital signature verification
GET DATA	None
GET STATUS	Secure Channel initiation
INSTALL	Secure Channel initiation or digital signature verification
LOAD	Secure Channel initiation or digital signature verification
MANAGE CHANNEL	Not applicable
PUT KEY	Secure Channel initiation
SELECT	Not applicable
SET STATUS	Secure Channel initiation
STORE DATA	Secure Channel initiation

Table 11-2: Minimum Security Requirements for GlobalPlatform Commands

11.1 General Coding Rules

The use of 'RFU' and '-' (dash) in tables in this chapter is as defined in Table 1-3: Abbreviations and Notations. If a bit is shown as RFU the bit should be set to zero by the off-card entity and the card shall ignore the value.

11.1.1 Life Cycle State Coding

The Executable Load File Life Cycle is coded on one byte as described in the following table:

b8	b7	b6	b5	b4	b3	B2	b1	Meaning
0	0	0	0	0	0	0	1	LOADED

Table 11-3: Executable Load File Life Cycle Coding

The Application Life Cycle has a bit-oriented coded value on one byte as described in the following table.

Note: the application has free use of bits b4-b7, and the coding of these bits is beyond the scope of this specification.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	1	1	INSTALLED
0	0	0	0	0	1	1	1	SELECTABLE
0	X	X	X	X	1	1	1	Application Specific State
1	-	-	-	-	-	1	1	LOCKED

Table 11-4: Application Life Cycle Coding

The Security Domain Life Cycle has a bit-oriented coded value on one byte as described in the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	1	1	INSTALLED
0	0	0	0	0	1	1	1	SELECTABLE
0	0	0	0	1	1	1	1	PERSONALIZED
1	0	0	0	-	-	1	1	LOCKED

Table 11-5: Security Domain Life Cycle Coding

The allowed Application and Security Domain Life Cycle transitions are described in chapter 5 - *Life Cycle Models*.

The Issuer Security Domain inherits the card Life Cycle State that has a bit-oriented coded value on one byte as described in the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	0	1	OP_READY
0	0	0	0	0	1	1	1	INITIALIZED
0	0	0	0	1	1	1	1	SECURED
0	1	1	1	1	1	1	1	CARD_LOCKED
1	1	1	1	1	1	1	1	TERMINATED

Table 11-6: Card Life Cycle Coding

The allowed card Life Cycle transitions are described in chapter 5 - *Life Cycle Models*.

11.1.2 Privileges Coding

Privileges are coded on three bytes as described in Table 11-7 and Table 11-8 (See section 6.6 - *Privileges* for additional details):

b8	b7	b6	b5	b4	b3	b2	b1	Meaning	Privilege Number
1	-	-	-	-	-	-	-	Security Domain	0
1	1	-	-	-	-	-	0	DAP Verification	1
1	-	1	-	-	-	-	-	Delegated Management	2
-	-	-	1	-	-	-	-	Card Lock	3
-	-	-	-	1	-	-	-	Card Terminate	4
-	-	-	-	-	1	-	-	Card Reset	5
-	-	-	-	-	-	1	-	CVM Management	6
1	1	-	-	-	-	-	1	Mandated DAP Verification	7

Table 11-7: Privileges (byte 1)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning	Privilege Number
1	-	-	-	-	-	-	-	Trusted Path	8
-	1	-	-	-	-	-	-	Authorized Management	9
-	-	1	-	-	-	-	-	Token Management	10
-	-	-	1	-	-	-	-	Global Delete	11
-	-	-	-	1	-	-	-	Global Lock	12
-	-	-	-	-	1	-	-	Global Registry	13
-	-	-	-	-	-	1	-	Final Application	14
-	-	-	-	-	-	-	1	Global Service	15

Table 11-8: Privileges (byte 2)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning	Privilege Number
1	-	-	-	-	-	-	-	Receipt Generation	16
-	X	X	X	X	X	X	X	RFU	-

Table 11-9: Privileges (byte 3)

11.1.3 General Error Conditions

The following table describes the error conditions that may be returned by any command:

SW1	SW2	Meaning
'64'	'00'	No specific diagnosis
'67'	'00'	Wrong length in Lc
'68'	'81'	Logical channel not supported or is not active
'69'	'82'	Security status not satisfied
'69'	'85'	Conditions of use not satisfied
'6A'	'86'	Incorrect P1 P2
'6D'	'00'	Invalid instruction

SW1	SW2	Meaning
'6E'	'00'	Invalid class

Table 11-10: General Error Conditions

11.1.4 Class Byte Coding

The class byte of all commands shall conform to ISO/IEC 7816-4 and shall be coded according to section 11.1.4.1 for commands addressed to the Basic Logical Channel and Supplementary Logical Channels 1, 2 and 3. For cards implementing four or more Supplementary Logical Channels the class byte of all commands addressed to Supplementary Logical Channel four to nineteen shall be coded according to section 11.1.4.2.

11.1.4.1 First Interindustry Class Byte Coding

The following table describes the first interindustry class byte coding:

b8	b7	b6	b5(*)	b4	b3	b2	b1	Meaning
0	0	0	0	-	-	-	-	Command defined in ISO/IEC 7816
1	0	0	0	-	-	-	-	GlobalPlatform command
-	0	0	0	0	0	-	-	No secure messaging
-	0	0	0	0	1	-	-	Secure messaging - GlobalPlatform proprietary
-	0	0	0	1	0	-	-	Secure messaging - ISO/IEC 7816 standard, command header not processed (no C-MAC)
-	0	0	0	1	1	-	-	Secure messaging - ISO/IEC 7816 standard, command header authenticated (C-MAC)
-	0	0	0	-	-	X	X	Logical channel number

Table 11-11: CLA Byte Coding

Class byte bits b1 and b2 may be set to define the required logical channel number according to ISO/IEC 7816;

- A class byte with bits b1 and b2 set to 00 indicates receipt of the command on the Basic Logical Channel;
- A class byte with bits b1 and b2 set to 01 (1), 10 (2) or 11 (3) indicates receipt of the command on a Supplementary Logical Channel.

Class byte bits b4 and b3 may be set to define the required secure messaging according to ISO/IEC 7816-4:

- A class byte with bits b4 and b3 set to 00 indicates no secure messaging;
- A class byte with bits b4 and b3 set to 01 indicates secure messaging with GlobalPlatform format;
- A class byte with bits b4 and b3 set to 11 or 10 indicates secure messaging with ISO/IEC 7816-4 format.

(*) Note: class byte bit b5 is always 0 except if it is coded for command chaining as defined in appendix F.

11.1.4.2 Further Interindustry Class Byte Coding

The following table describes the further interindustry class byte coding.

b8	b7	b6	b5(*)	b4	b3	b2	b1	Meaning
0	1	-	0	-	-	-	-	Command defined in ISO/IEC 7816
1	1	-	0	-	-	-	-	GlobalPlatform command

b8	b7	b6	b5(*)	b4	b3	b2	b1	Meaning
-	1	0	0	-	-	-	-	No secure messaging
-	1	1	0	-	-	-	-	Secure messaging - ISO/IEC 7816 or GlobalPlatform proprietary
-	1	-	0	X	X	X	X	Logical channel number

Table 11-12: CLA Byte Coding

Class byte bits b1 to b4 may be set to define the required logical channel number according to ISO/IEC 7816;

- A class byte with bit b7 set to 1 indicates in bits b4 to b1 (values 0000 to 1111) receipt of the command on Supplementary Logical Channel 4 to 19.

Class byte bit b6 may be set to define the required secure messaging according to ISO/IEC 7816-4 or proprietary GlobalPlatform format:

- A class byte with bit b6 set to 0 indicates no secure messaging;
- A class byte with bit b6 set to 1 indicates secure messaging with GlobalPlatform format or with ISO/IEC 7816-4 format.

11.1.5 APDU Message and Data Length

All GlobalPlatform APDU messages and data elements are counted in bytes. All GlobalPlatform commands comply with ISO/IEC 7816 short message lengths i.e. Lc and Le bytes coded on one byte.

All GlobalPlatform command messages (including the APDU header) are limited to 255 bytes in length. All GlobalPlatform commands that expect response data have their Le byte set to zero, indicating that all available response data shall be returned. According to ISO/IEC 7816-4, all GlobalPlatform responses returned in APDU response messages shall have a maximum length of 256 bytes of response data.

All length fields of GlobalPlatform messages and data objects defined in this specification, except for Lc and Le, are coded as defined for ASN.1 BER-TLV (see ISO 8825-1): 1 byte for a length up to 127, 2 bytes for a length up to 255, and 3 bytes for a length up to 65535, except where otherwise stated.

This version of the Specification is written as though most command functions can be handled with a single APDU command and response pair. This is done in order to simplify the description, and is not intended to preclude the use of multiple commands being used where such a mechanism is provided, as indicated for each command: such as (for GlobalPlatform commands) the 'more commands' bit in the P1 byte and status bytes controlling the return of additional data; and (for ISO commands) command and response chaining.

11.1.6 Confirmations in Response Messages

The confirmation shall be structured according to the following table:

Length	Data Element	Presence
1-2	Length of Receipt ('00' - '7F' or '81 80' - '81 FF')	Mandatory
0-n	Receipt	Conditional
5-n	Confirmation Data	Mandatory

Table 11-13: Confirmation Structure

The length field for the Receipt is coded according to ASN.1 BER-TLV (see ISO 8825-1).

Confirmation Data is structured according to the following table:

Length	Data Element	Presence
1	Length of Confirmation Counter	Mandatory
2	Confirmation Counter	Mandatory
1	Length of Card Unique Data	Mandatory
1-n	Card Unique Data	Mandatory
1	Length of Token identifier	Optional
0-n	Token identifier	Optional
1	Length of Token Data digest	Conditional
0-n	Token Data digest	Optional

Table 11-14: Confirmation Data

If Length of Token identifier is present, Length of Token Data digest shall also be present. The presence of Token identifier depends on whether a Token identifier was included in the original Token.

Card Unique Data is the concatenation without delimiters of the IIN and the CIN.

11.1.7 Implicit Selection Parameter Coding

The Implicit Selection parameter (tag 'CF') indicates that this Application is to be implicitly selected on one (or more) specific card interface(s) for one (or more) specific logical channel(s), see section 6.6.3 - *Privilege Management*. Setting both bits 8 and 7 to zero indicates that this Application is to be selectable only explicitly with a SELECT [by name] command and the logical channel number shall be ignored by the card. The value field is coded on one byte as follows:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	-	-	-	-	-	-	-	Contactless I/O
-	1	-	-	-	-	-	-	Contact I/O
-	-	X	-	-	-	-	-	RFU
-	-	-	X	X	X	X	X	Logical channel number (0 to 19)

Table 11-15: Implicit Selection Parameter

11.1.8 Key Type Coding

Key type may be coded on one or two bytes. When coded on two bytes, the first byte shall be set to 'FF' to indicate an extended format of the key data field. The second or only byte shall be coded according to the following table:

Value	Meaning
'00'-'7F'	Reserved for private use
'80'	DES – mode (ECB/CBC) implicitly known
'81'	Reserved (Triple DES)
'82'	Triple DES in CBC mode
'83'	DES in ECB mode
'84'	DES in CBC mode
'85'-'87'	RFU (symmetric algorithms)
'88'	AES (16, 24, or 32 long keys)
'89'-'8F'	RFU (symmetric algorithms)
'90'	HMAC-SHA1 – length of HMAC is implicitly known

Value	Meaning
'91'	HMAC-SHA1-160 – length of HMAC is 160 bits
'92'-'9F'	RFU (symmetric algorithms)
'A0'	RSA Public Key - public exponent e component (clear text)
'A1'	RSA Public Key - modulus N component (clear text)
'A2'	RSA Private Key - modulus N component
'A3'	RSA Private Key - private exponent d component
'A4'	RSA Private Key - Chinese Remainder P component
'A5'	RSA Private Key - Chinese Remainder Q component
'A6'	RSA Private Key - Chinese Remainder PQ component ($q^{-1} \bmod p$)
'A7'	RSA Private Key - Chinese Remainder DP1 component ($d \bmod (p-1)$)
'A8'	RSA Private Key - Chinese Remainder DQ1 component ($d \bmod (q-1)$)
'A9'-'FE'	RFU (asymmetric algorithms)
'FF'	Extended format

Table 11-16: Key Type Coding

11.1.9 Key Usage Coding

The values for the Key usage parameter are set according to the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	-	-	-	-	-	-	-	Verification (DST, CCT, CAT), Encipherment (CT)
-	1	-	-	-	-	-	-	Computation (DST, CCT, CAT), Decipherment (CT)
-	-	1	-	-	-	-	-	Secure messaging in response data fields (CT, CCT)
-	-	-	1	-	-	-	-	Secure messaging in command data fields (CT, CCT)
-	-	-	-	1	-	-	-	Confidentiality (CT)
-	-	-	-	-	1	-	-	Cryptographic Checksum (CCT)
-	-	-	-		-	1	-	Digital Signature (DST)
-	-	-	-	-	-	-	1	Cryptographic Authorization (CAT)

Table 11-17: Key Usage

The following values are used by GlobalPlatform. Values are independent of the cryptographic algorithm used.

- C-MAC = '14', R-MAC = '24', C-MAC + R-MAC = '34';
- C-ENC = '18', R-ENC = '28', C-ENC + R-ENC = '38';
- C-DEK = '48', R-DEK = '88', C-DEK + R-DEK = 'C8';
- PK_SD_AUT = '82';
- SK_SD_AUT = '42';

- Token = '81';
- Receipt = '44';
- DAP = '84';
- PK_SD_AUT + Token = '83';
- SK_SD_AUT + Receipt = '43';
- PK_SD_AUT + DAP = '86';
- PK_SD_AUT + Token + DAP = '87'.

11.1.10 Key Access Coding

The values for the Key Access parameter are set according to the following table (see appendix 7.5.2 - *Key Access Conditions* for details):

Value	Description
'00'	The key may be used by the Security Domain and any associated Application
'01'	The key may only be used by the Security Domain
'02'	The key may be used by any Application associated with the Security Domain but not by the Security Domain itself
'03'- '1F'	RFU
'20'- 'FE'	Proprietary usage
'FF'	Not available

Table 11-18: Key Access

11.1.11 Tag Coding

All tags of GlobalPlatform data objects are coded as defined by ASN.1 BER-TLV rules (see ISO 8825-1).

For Security Domain data objects present in GlobalPlatform APDU messages, BER-TLV tags are coded according to the following rules:

- '00' to '7E' – Reserved for use by ISO/IEC;
- '80' to '9E' and 'A0' to 'BE' – Reserved for use depending on the context (see note);
- 'C0' to 'DD' and 'E0' to 'FD' – Reserved for use by GlobalPlatform or individual schemes registered by GlobalPlatform;
  - 'CA' and 'EA' – Reserved for use by ETSI TS 102 226 specification;
- 'DE' and 'FE' – Reserved for proprietary use and not registered by GlobalPlatform;
- '1F 1F' to '7F 7F' – Reserved for use by ISO/IEC;
- '9F 1F' to '9F 7F' and 'BF 1F' to 'BF 7F' – Reserved for use depending on the context (see note);

- 'DF 1F' to 'DF 7F' and 'FF 1F' to 'FF 7F' – Reserved for use by GlobalPlatform or individual schemes registered by GlobalPlatform;
    - 'FF 1F' to 'FF 3F' – Reserved for use by ETSI TS 102 226 specification.

Note: Context dependant tags are assigned by the authority defining the context, e.g. ISO/IEC for data objects embedded in other ISO/IEC data objects or GlobalPlatform for data objects embedded in other GlobalPlatform data objects.

Tag allocation for any data objects embedded in the constructed tag 'FE' (reserved for proprietary use) is beyond the scope of this specification. Applying BER-TLV rules in this case is recommended.

11.2 DELETE Command

11.2.1 Definition and Scope

The DELETE command is used to delete a uniquely identifiable object such as an Executable Load File, an Application, an Executable Load File and its related Applications or a key. In order to delete an object, the object shall be uniquely identifiable by the selected Application.

11.2.2 Command Message

The DELETE command message is coded according to the following table:

Code	Value	Meaning
CLA	'80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'E4'	DELETE
P1	'xx'	Reference control parameter P1
P2	'xx'	Reference control parameter P2
Lc	'xx'	Length of data field
Data	'xxxx…'	TLV coded objects (and MAC if present)
Le	'00'	

Table 11-19: DELETE Command Message

11.2.2.1 Reference Control Parameter P1

Reference control parameter P1 allows for command data to be longer than 255 bytes and to be segmented into components of arbitrary size and transmitted in a series of DELETE commands. P1 indicates whether the command data is one of a sequence of components or the last (or only) component.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	Last (or only) command
1	-	-	-	-	-	-	-	More DELETE commands
-	X	X	X	X	X	X	X	RFU

Table 11-20: DELETE Reference Control Parameter P1

11.2.2.2 Reference Control Parameter P2

The reference control parameter P2 indicates whether the object in the data field is being deleted or whether the object in the data field and its related objects are being deleted. It shall be coded according to the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	Delete object
1	-	-	-	-	-	-	-	Delete object and related object

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
-	X	X	X	X	X	X	X	RFU

Table 11-21: DELETE Reference Control Parameter P2

11.2.2.3 Data Field Sent in the Command Message

The data field of the DELETE command message shall contain the TLV coded name(s) of the object to be deleted.

11.2.2.3.1 Delete [card content] Data Field

The Application or Executable Load File AID may be followed by a Control Reference Template for Digital Signature and a Delete Token, as follows:

Tag	Length	Name	Presence
'4F'	5-16	Executable Load File or Application AID	Mandatory
'B6'	Variable	Control Reference Template for Digital Signature	Conditional
'42'	1-n	Token Issuer Id	Optional
'45'	1-n	Card Image Number	Optional
'5F20'	1-n	Application Provider identifier	Optional
'93'	1-n	Token identifier/number (digital signature counter)	Conditional
'9E'	1-n	Delete Token	Conditional

Table 11-22: Delete [card content] Command Data Field

The identity of the Application or Executable Load File to delete shall be specified using the tag for an AID ('4F') followed by a length and the AID of the Application or Executable Load File. When simultaneously deleting an Executable Load File and all its related Applications, only the identity of the Executable Load File shall be provided.

The presence of the Control Reference Template for Digital Signature and Delete Token depends on the Card Issuer's policy. The presence of the data objects in tag 'B6' and the sub-tags ‘42’, ‘45’, ‘5F20’ and ‘93’ are strongly recommended when using SCP10.

The length fields for the Control Reference Template for Digital Signature and the Delete Token are coded according to ASN.1 BER-TLV (see ISO 8825-1).

11.2.2.3.2 Delete (Key) Data Field

To delete one or more keys, the Key Version Number and/or the Key Identifier are included in the command data field as follows:

Tag	Length	Meaning	Presence
'D0'	1	Key Identifier	Conditional
'D2'	1	Key Version Number	Conditional

Table 11-23: DELETE [key] Command Data Field

A single key is deleted when both the Key Identifier ('D0') and the Key Version Number ('D2') are provided in the DELETE command message data field. Multiple keys may be deleted if one of these values is omitted (i.e. all keys with the specified Key Identifier or Key Version Number). The options available for omitting these values are conditional on the Issuer's policy.

11.2.3 Response Message

A data field shall always be returned in the response message. The content of the data field is only relevant in the case of Delegated Management i.e. if a Security Domain with the Delegated Management privilege is deleting an Application or Executable Load File, a Receipt may be present in the data field depending on the security policy of the Card Issuer.

11.2.3.1 Data Field Returned in the Response Message

For a DELETE [key] command, a single byte of '00' shall be returned indicating that no additional data is present.

For a DELETE [card content] command being issued to a Security Domain with the Delegated Management privilege, the data field may contain the confirmation of the delete procedure. The overall length of the response message shall not exceed 256 bytes.

The following table describes the structure of the DELETE response data field. The length field for the Delete Confirmation is coded according to ASN.1 BER-TLV (see ISO 8825-1).

Name	Length	Value	Presence
Length of delete confirmation	1-2	'00' - '7F' or '81 80' - '81 FF'	Mandatory
Delete confirmation	0-n	'xxxx...' - see section 11.1.6	Conditional

Table 11-24: DELETE Response Data Field

11.2.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'65'	'81'	Memory failure
'6A'	'88'	Referenced data not found
'6A'	'82'	Application not found
'6A'	'80'	Incorrect values in command data

Table 11-25: DELETE Error Conditions

11.3 GET DATA Command

11.3.1 Definition and Scope

The GET DATA command is used to retrieve either a single data object, which may be constructed, or a set of data objects. Reference control parameters P1 and P2 coding is used to define the specific data object tag. The data object may contain information pertaining to a key.

11.3.2 Command Message

The GET DATA command message is coded according to the following table:

Code	Value	Meaning
CLA	'00' - '0F', '40' - '4F', '60' - '6F', '80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'CA' or 'CB'	GET DATA - If CLA = '00' - '0F', '40' - '4F', '60' - '6F', even or odd instruction code 'CA' or 'CB'; - If CLA = '80' - '8F', 'C0' - 'CF' or 'E0' - 'EF', even instruction code 'CA'.
P1	'xx'	'00' or high order tag value
P2	'xx'	Low order tag value
Lc	'xx'	Not present if no command data, or length of data field.
Data	'xxxx…'	Tag list and/or MAC if present.
Le	'00'	

Table 11-26: GET DATA Command Message

According to ISO/IEC 7816-4, the odd instruction code ‘CB’ in conjunction with a class byte indicating an ISO command (CLA set to '00' - '0F', '40' - '4F' or '60' - '6F') is used to retrieve the contents of a file. It may be used to retrieve the list of Applications on the card (P1-P2 set to ‘2F 00’).

11.3.2.1 Parameter P1 and P2

Parameters P1 and P2 define the tag of the data object to be read.

The value '2F 00' indicates a request to obtain details of Applications on the card, as defined in ISO/IEC 7816-4. The instruction code shall be set to 'CA' if the class byte indicates a GlobalPlatform command (CLA set to '80' - '8F', 'C0' - 'CF' or 'E0' - 'EF') or 'CB' if the class byte indicates an ISO command (CLA set to '00' - '0F', '40' - '4F' or '60' - '6F').

Security Domains shall support at least the following data object tags:

- Tag '42': Issuer Identification Number (or Security Domain Provider Identification Number);
- Tag '45': Card Image Number (or Security Domain Image Number);
- Tag '66': Card Data (or Security Domain Management Data);

- Tag 'E0': Key Information Template.

A Security Domain may support the following data object tags:

- Tag 'D3': Current Security Level;
- Tag '2F00': List of Applications belonging to the Security Domain, or every application on the card if the Security Domain has Global Registry Privilege;
- Tag 'FF21': Extended Card Resources Information available for Card Content Management, as defined in ETSI TS 102 226.

If present, the Security Domain with Receipt Generation privilege shall support the following additional data object tag:

- Tag 'C2': Confirmation Counter.

A Security Domain supporting Secure Channel Protocol '02' shall support the following data object tag:

- Tag 'C1': Sequence Counter of the default Key Version Number.

For a Security Domain supporting only Secure Channel Protocol '01', an attempt to retrieve the Sequence Counter of the default Key Version Number (tag 'C1') shall be rejected.

Other tags as defined in section 11.1.11 - *Tag Coding* may be available for data objects supported by a Security Domain.

11.3.2.2 Data Field Sent in the Command Message

The data field of the GET DATA command message shall be empty unless a tag list and/or a MAC is required. For retrieving a list of applications present on the card (P1-P2 set to '2F 00') a tag list shall be present and coded as '5C 00'.

11.3.3 Response Message

11.3.3.1 Data Field Returned in the Response Message

When the class byte indicates a GlobalPlatform proprietary command (bit b8 = 1), the GET DATA response data field shall contain the TLV coded data object referred to in reference control parameters P1 and P2 of the command message, except where P1-P2 are set to '2F 00'.

Where the class byte indicates an ISO command (bit b8 = 0), the GET DATA response data field shall contain only the value of the TLV coded data object referred to in reference control parameters P1 and P2 of the command message, except where P1-P2 are set to '2F 00'.

When retrieving key information for the currently selected Application, the key information is returned in the template 'E0' where each Key Information Data data object is introduced by the tag 'C0'. Its structure depends on the value of Key Type. If the Key Type is coded on one byte (value other than 'FF'), the Key Information Data is structured as follows:

Name	Length	Value	Presence
Key Identifier	1	Key Identifier value	Mandatory
Key Version Number	1	Key Version Number value	Mandatory
Key type of first (or only) key component	1	'00' - 'FE', See section 11.1.8 - *Key Type Coding*	Mandatory
Length of first (or only) key component	1	'01' - 'FF'	Mandatory

Name	Length	Value	Presence
…	…	…	
Key type of last key component	1	'00' - 'FE', See section 11.1.8 - *Key Type Coding*	Conditional
Length of last key component	1	'01' - 'FF'	Conditional

Table 11-27: Key Information Data Structure - Basic

If the Key Type is coded on two bytes (first byte = 'FF'), the Key Information Data is structured as follows:

Name	Length	Value	Presence
Key Identifier	1	Key Identifier value	Mandatory
Key Version Number	1	Key Version Number value	Mandatory
Key type of first (or only) key component	2	'FF xx' - see section 11.1.8 - *Key Type Coding*	Mandatory
Length of first (or only) key component	2	'00 01' - '7F FF'	Mandatory
…	…	…	
Key type of last key component (if any)	2	'FF xx' - See section 11.1.8 - *Key Type Coding*	Conditional
Length of last key component (if any)	2	'00 01' - '7F FF'	Conditional
Length of Key Usage	1	'00' - '01'	Mandatory
Key usage	0 or 1	'xx' - See section 11.1.9	Conditional
Length of Key Access	1	'00' - '01'	Mandatory
Key Access	0 or 1	'xx' - See section 11.1.10	Conditional

Table 11-28: Key Information Data Structure - Extended

The presence of Key Usage and Key Access is conditional on the data having been supplied when the key was loaded on the card.

When retrieving Extended Card Resources Information with tag 'FF21', the response shall be coded as defined in ETSI TS 102 226.

When retrieving the list of Applications present on the card, the response shall be coded as a series of Application Template data objects (tag '61') as defined in ISO/IEC 7816-4. An example of the response is as follows:

Tag	Length	Name	Presence
'61'	7-n	Application Template	Mandatory
'4F'	5-16	Application AID	Mandatory
	...	...	
'61'	7-n	Application Template	Mandatory
'4F'	5-16	Application AID	Mandatory
	...	...	

Table 11-29: List of On-Card Applications

11.3.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or the following error condition.

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table 11-30: GET DATA Error Conditions

11.4 GET STATUS Command

11.4.1 Definition and Scope

The GET STATUS command is used to retrieve Issuer Security Domain, Executable Load File, Executable Module, Application or Security Domain Life Cycle status information according to a given match/search criteria.

11.4.2 Command Message

The GET STATUS command message shall be coded according to the following table.

Code	Value	Meaning
CLA	'80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'F2'	GET STATUS
P1	'xx'	Reference control parameter P1
P2	'xx'	Reference control parameter P2
Lc	'xx'	Length of data field
Data	'xx...'	Search criteria (and MAC if present)
Le	'00'	

Table 11-31: GET STATUS Command Message

11.4.2.1 Reference Control Parameter P1

Reference control parameter P1 is used to select a subset of statuses to be included in the response message. It is coded as follows:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	-	-	-	-	-	-	-	Issuer Security Domain
-	1	-	-	-	-	-	-	Applications, including Security Domains
-	-	1	-	-	-	-	-	Executable Load Files
-	-	-	1	-	-	-	-	Executable Load Files and Executable Modules
-	-	-	-	X	X	X	X	RFU

Table 11-32: GET STATUS Reference Control Parameter P1

The following values of the reference control parameter shall be supported:

'80' – Issuer Security Domain only. In this case the search criteria is ignored and the Issuer Security Domain information is returned.

'40' – Applications and Supplementary Security Domains only.

'20' – Executable Load Files only.

'10' – Executable Load Files and their Executable Modules only.

The ability to differentiate a Security Domain from an Application is achieved through privileges.

11.4.2.2 Reference Control Parameter P2

The reference control parameter P2 controls the number of consecutive GET STATUS command and indicates the format of the response message. It shall be coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
X	X	X	X	X	X	-	-	RFU
-	-	-	-	-	-	-	0	Get first or all occurrence(s)
-	-	-	-	-	-	-	1	Get next occurrence(s)
-	-	-	-	-	-	0	-	Deprecated: response data structure according to Table 11-35 and Table 11-37
-	-	-	-	-	-	1	-	Response data structure according to Table 11-36

Table 11-33: GET STATUS Reference Control Parameter P2

The get next occurrence indicator shall be rejected if no prior GET STATUS [get first or all occurrence(s)] was received within the current Application Session.

When retrieving the status of the Issuer Security Domain only, the get next occurrence indicator shall be rejected.

Requesting a response coded according to Table 11-36 is recommended when interrogating a card compliant to this version of the specification or Amendment A of version 2.1.1. Note that cards compliant to previous versions of this specification may only support response data structures as defined in Table 11-35 and Table 11-37.

11.4.2.3 Data Field Sent in the Command Message

The data field is structured as follows:

Tag	Length	Name	Presence
'4F'	0-16	Application AID	Mandatory
'xx' or 'xxxx'	0-n	Other search criteria	Optional
...	...	...	...
'5C'	1-n	Tag list	Optional

Table 11-34: GET STATUS Command Data Field

The GET STATUS command message data field shall contain at least one TLV coded search qualifier: the AID (tag '4F'). It shall be possible to search for all the occurrences that match the selection criteria according to the reference control parameter P1 using a search criteria of '4F' '00'.

It shall be possible to search for all occurrences with the same RID.

Other search criteria may be added. In such cases, the additional search criteria shall be TLV coded and appended after the mandatory search criterion AID (tag ‘4F’). When not supported by the card, additional search criteria shall be ignored.

The search is limited to the Executable Load Files, Applications and Security Domains that are directly or indirectly associated with the on-card entity receiving the command. When the on-card entity receiving the command has the Global Registry privilege, the search applies to all Executable Load Files, Applications and Security Domains registered in the GlobalPlatform Registry.

The tag list (tag ‘5C’) indicates to the card how to construct the response data for each on-card entity matching the search criteria. Its value field contains a concatenation of tags (without delimitation) indicating the data objects to include in the response. If an on-card entity that matches the search criteria is found in GlobalPlatform Registry without the corresponding data object, an error status shall be returned. The tag list may only be present when

reference control parameter P2 indicates a TLV-coded response (i.e. bit b2 of P2 set to 1). When not supported by the card, the tag list shall be ignored.

11.4.3 Response Message

11.4.3.1 Data Field Returned in the Response Message

Based upon the search criteria of the GET STATUS command data field and the selection criteria of reference control parameter P1 and P2, multiple occurrences of the data structure in the following tables may be returned. Responses coded according to Table 11-35 and Table 11-37 are deprecated.

Name	Length	Value	Presence
Length of Application AID	1	'05' - '10'	Mandatory
Application AID	5-16	'xxxx...'	Mandatory
Life Cycle State	1	'xx' - see section 11.1.1	Mandatory
Privileges (byte 1)	1	'xx' - see section 11.1.2	Mandatory

Table 11-35: Issuer Security Domain, Application and Executable Load File Information Data

Note for backward compatibility: when receiving Privileges coded on one byte for any Application or Security Domain, the off-card entity should consider the second and third byte as being set to the default values defined in section 6.6.2.

Tag	Length	Name	Presence
'E3'	Variable	GlobalPlatform Registry related data	Conditional
'4F'	5-16	AID	Conditional
'9F70'	1	Life Cycle State	Conditional
'C5'	0,1,3	Privileges (byte 1 - byte 2 – byte 3) see section 11.1.2	Conditional
'C4'	1-n	Application's Executable Load File AID	Conditional
'CE'	1-n	Executable Load File Version Number	Conditional
'84'	1-n	First or only Executable Module AID	Conditional
…	…	…	…
'84'	1-n	Last Executable Module AID	Conditional
'CC'	1-n	Associated Security Domain's AID	Conditional

Table 11-36: GlobalPlatform Registry Data (TLV)

When tag lists are supported by the card, the response shall be coded according to Table 11-36 and shall only contain the data objects whose tags are listed in the tag list (tag '5C') of the command, the order of the data objects within the GlobalPlatform Registry data (template 'E3') being arbitrary. When no tag list (tag '5C') is present in the command or when tag lists are not supported by the card, a response coded according to Table 11-36 shall contain the following data objects:

- The AID (tag '4F') and Life Cycle State (tag '9F70') shall be present;
- Privileges (tag 'C5') shall be present for the Issuer Security Domain, Supplementary Security Domains and Applications and shall be absent for Executable Load Files;

- The Executable Modules (tag '84') shall be present if reference control parameter P1 indicates Executable Load Files and their Executable Modules only (bit b5 of P1 set to 1) and if the Runtime Environment supports Executable Modules.

Note: the Executable Load File Version Number format and contents are beyond the scope of this specification. It shall consist of the version information contained in the original Load File: on a Java Card based card, this version number represents the major and minor version attributes of the original Load File Data Block.

Name	Length	Value	Presence
Length of Executable Load File AID	1	'05' - '10'	Mandatory
Executable Load File AID	5-16	'xxxx…'	Mandatory
Executable Load File Life Cycle State	1	'xx' - see section 11.1.1	Mandatory
Privileges	1	'00'	Mandatory
Number of Executable Modules	1	'01' - '7F'	Conditional
Length of Executable Module AID	1	'05' - '10'	Conditional
Executable Module AID	5-16	'xxxx…'	Conditional
…	…	…	...
Length of Executable Module AID	1	'05' - '10'	Conditional
Executable Module AID	5-16	'xxxx…'	Conditional

Table 11-37: Executable Load File and Executable Module Information Data

The Executable Modules shall be present if reference control parameter P1 indicates Executable Load Files and their Executable Modules only (bit b5 of P1 set to 1) and if the Runtime Environment supports Executable Modules.

11.4.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may return the following warning condition:

SW1	SW2	Meaning
'63'	'10'	More data available

Table 11-38: GET STATUS Warning Condition

Following status '63 10' a subsequent GET STATUS [get next occurrence(s)] may be issued to retrieve additional data.

This command may return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found
'6A'	'80'	Incorrect values in command data

Table 11-39: GET STATUS Error Conditions

11.5 INSTALL Command

11.5.1 Definition and Scope

The INSTALL command is issued to a Security Domain to initiate or perform the various steps required for Card Content management.

11.5.2 Command Message

The INSTALL command message shall be coded according to the following table.

Code	Value	Meaning
CLA	'80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'E6'	INSTALL
P1	'xx'	Reference control parameter P1
P2	'00', '01' or '03'	Reference control parameter P2
Lc	'xx'	Length of data field
Data	'xxxx…'	Install data (and MAC if present)
Le	'00'	

Table 11-40: INSTALL Command Message

11.5.2.1 Reference Control Parameter P1

The reference control parameter P1 of the INSTALL command defines the specific role of the INSTALL command and also allows for command data to be longer than 255 bytes and to be segmented into arbitrary components and transmitted in a series of INSTALL commands. It is coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	Last (or only) command
1	-	-	-	-	-	-	-	More INSTALL commands
-	1	0	0	0	0	0	0	For registry update
-	0	1	0	0	0	0	0	For personalization
-	0	0	1	0	0	0	0	For extradition
-	0	0	0	1	-	-	0	For make selectable
-	0	0	0	-	1	-	0	For install
-	0	0	0	-	-	1	0	For load

Table 11-41: INSTALL Command Reference Control Parameter P1

Bits b8 to b1 shall be coded as follows:

b8 = 1 indicates that the command data is one of a sequence of components (and not the last), **b8 = 0** indicates that it is the last (or only) component.

b7 = 1 indicates that the GlobalPlatform Registry is to be updated, or that functions are to be restricted.

b6 = 1 indicates that the currently selected Security Domain shall personalize one of its associated Applications and a subsequent STORE DATA command is to be expected

b5 = 1 indicates that the Application shall be extradited.

b4 = 1 indicates that the Application shall be made selectable. This applies to an Application being installed or an Application that is already installed.

b3 = 1 indicates that the Application shall be installed.

b2 = 1 indicates that the Load File shall be loaded. A subsequent LOAD command is to be expected.

A combination of the [for install] and [for make selectable] options may apply. A combination of the [for load], [for install] and [for make selectable] options may also apply.

11.5.2.2 Reference Control Parameter P2

The reference control parameter P2 shall be set as follows:

'00' indicates that no information is provided.

'01' indicates the beginning of the combined Load, Install and Make Selectable process.

'03' indicates the end of the combined Load, Install and Make Selectable process.

11.5.2.3 Data Field Sent in the Command Message

The data field of the command message contains LV coded data. The LV coded data is represented without delimiters.

11.5.2.3.1 Data Field for INSTALL [for load]

The following table details the INSTALL [for load] command data field. It also applies to the first combined INSTALL [for load, install and make selectable] command data field.

Name	Length	Value	Presence
Length of Load File AID	1	'05' - '10'	Mandatory
Load File AID	5-16	'xxxx...'	Mandatory
Length of Security Domain AID	1	'00' or '05' - '10'	Mandatory
Security Domain AID	0 or 5-16	'xxxx...'	Conditional
Length of Load File Data Block Hash	1	'00' - '7F'	Mandatory
Load File Data Block Hash	0-n	'xxxx...' - see appendix C.2	Conditional
Length of Load Parameters field	1-2	'00' - '7F', or '81 80' - '81 FF'	Mandatory
Load Parameters field	0-n	'xxxx...' - see section 11.5.2.3.7	Conditional
Length of Load Token	1-3	'00' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Mandatory
Load Token	0-n	'xxxx...' - see appendix C.4.1	Conditional

Table 11-42: INSTALL [for load] Command Data Field

The Load Token is mandatory for Delegated Management (except for the combined [for load, install and make selectable] command), and for Authorized Management if the off-card entity is not the Security Domain Provider. In all other cases an error status shall be returned if a Load Token is present. The length field for a Load Token is as defined for ASN.1 BER-TLV (see ISO 8825-1) except that the length 128 may also be coded on one byte as '80'.

The Load File Data Block Hash is mandatory if a Load Token is present or if the Load File contains one or more DAP Blocks. In all other cases, the Load File Data Block Hash is optional and may be verified by the card.

The Load File AID and the Load Parameters shall be consistent with the information contained in the Load File Data Block (if any).

11.5.2.3.2 Data Field for INSTALL [for install]

The following table details the INSTALL [for install] command data field. It also applies to the final combined [for load, install and make selectable] command data field.

Name	Length	Value	Presence
Length of Executable Load File AID	1	'00' or '05' - '10'	Mandatory
Executable Load File AID	0 or 5-16	'xxxx...'	Conditional
Length of Executable Module AID	1	'00' or '05' - '10'	Mandatory
Executable Module AID	0 or 5-16	'xxxx...'	Conditional
Length of Application AID	1	'05' - '10'	Mandatory
Application AID	5-16	'xxxx...'	Mandatory
Length of Privileges	1	'01', '03'	Mandatory
Privileges	1, 3	byte 1 - byte 2 – byte 3: see section 11.1.2	Mandatory
Length of Install Parameters field	1-2	'01' - '7F' or '81 80' - '81 FF'	Mandatory
Install Parameters field	2-n	'xxxx' - see section 11.5.2.3.7	Mandatory
Length of Install Token	1-3	'00' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Mandatory
Install Token	0-n	'xxxx...' - see appendix C.4.2 or C.4.7	Conditional

Table 11-43: INSTALL [for install] Command Data Field

The Install Token is mandatory for Delegated Management and for Authorized Management if the off-card entity is not the Security Domain Provider. In all other cases an error status shall be returned if an Install Token is present. The length fields for the Install Token and the Install Parameters field are formatted as defined for ASN.1 BER-TLV (see ISO 8825-1) except that the length 128 may also be coded on one byte as '80'.

The presence of the Executable Load File AID is optional for the combined Load, Install and Make Selectable command. If present it shall match the Load File AID of the first combined Load, Install and Make Selectable command.

The Executable Module AID is the AID of the Executable Module previously loaded. The presence of the Executable Module depends on the requirements of the Run Time Environment.

GlobalPlatform cards use the instance AID to indicate the AID with which the installed Application will be selected.

The presence of the Privileges is required. If an Application is only being installed and not made selectable with the same INSTALL command the Card Reset privilege cannot be set.

The instance AID, the Privileges and the Application Specific Parameters shall be made known to the Application.

Note for backward compatibility: when receiving Privileges coded on one byte for any Application or Security Domain, the OPEN shall assign the second byte with the default values defined in section 6.6.2.

11.5.2.3.3 Data Field for INSTALL [for make selectable]

The following table details the INSTALL [for make selectable] command data field.

Name	Length	Value	Presence
Length of data	1	'00'	Mandatory
Length of data	1	'00'	Mandatory
Length of Application AID	1	'05' - '10'	Mandatory
Application AID	5-16	'xxxx...'	Mandatory
Length of Privileges	1	'01', '03'	Mandatory
Privileges	1, 3	byte 1 - byte 2 – byte 3: see section 11.1.2	Mandatory
Length of Make Selectable Parameters field	1	'00' - '7F'	Mandatory
Make Selectable Parameters field	0-n	'xxxx' - see section 11.5.2.3.7	Conditional
Length of Make Selectable Token	1-3	'00' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Mandatory
Make Selectable Token	0-n	'xxxx...' - see appendix C.4.3	Conditional

Table 11-44: INSTALL [for make selectable] Command Data Field

If the Card Reset privilege is set in the Privileges field, the GlobalPlatform Registry shall be updated according to the rules defined in section 6.6 - *Privileges*. Any other privilege set in the Privileges field, regardless of its length, shall be ignored by the card.

The Make Selectable Token is mandatory for Delegated Management, and for Authorized Management if the off-card entity is not the Security Domain Provider. In all other cases an error status shall be returned if a Make Selectable Token is present. The length field for the Make Selectable Token is as defined for ASN.1 BER-TLV (see ISO 8825-1) except that the length 128 may also be coded on one byte as '80'.

11.5.2.3.4 Data Field for INSTALL [for extradition]

The following table details the INSTALL [for extradition] command data field.

Name	Length	Value	Presence
Length of Security Domain AID	1	'05' - '10'	Mandatory
Security Domain AID	5-16	'xxxx...'	Mandatory
Length of data	1	'00'	Mandatory
Length of Application or Executable Load File AID	1	'05' - '10'	Mandatory
Application or Executable Load File AID	5-16	'xxxx...'	Mandatory
Length	1	'00'	Mandatory
Length of Extradition Parameters field	1	'00' - '7F'	Mandatory
Extradition Parameters field	0-n	'xxxx' - see section 11.5.2.3.7	Conditional

Name	Length	Value	Presence
Length of Extradition Token	1-3	'00' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Mandatory
Extradition Token	0-n	'xxxx...' - see appendix C.4.4	Conditional

Table 11-45: INSTALL [for extradition] Command Data Field

The Security Domain AID indicates to which Security Domain this Application or Executable Load File is being extradited. The Security Domain with which this Application or Executable Load File is currently associated is the currently selected Application.

The Extradition Token is mandatory for Delegated Management, and for Authorized Management if the off-card entity is not the Security Domain Provider. In all other cases an error status shall be returned if an Extradition Token is present. The length field for the Extradition Token is as defined for ASN.1 BER-TLV (see ISO 8825-1) except that the length 128 may also be coded on one byte as '80'.

11.5.2.3.5 Data Field for INSTALL [for registry update]

The following table details the INSTALL [for registry update] command data field.

Name	Length	Value	Presence
Length of Security Domain AID	1	'00' or '05' - '10'	Mandatory
Security Domain AID	0 or 5-16	'xxxx...'	Conditional
Length of data	1	'00'	Mandatory
Length of Application AID	1	'00' or '05' - '10'	Mandatory
Application AID	0 or 5-16	'xxxx...'	Conditional
Length of Privileges	1	'00', '01; '03'	Mandatory
Privileges	0, 1, 3	byte 1 - byte 2 – byte 3: see section 11.1.2	Conditional
Length of Registry Update Parameters field	1	'00' - '7F'	Mandatory
Registry Update Parameters field	0-n	'xxxx' - see section 11.5.2.3.7	Conditional
Length of Registry Update Token	1-3	'00' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Mandatory
Registry Update Token	0-n	'xxxx...' - see appendix 0	Conditional

Table 11-46: INSTALL [for registry update] Command Data Field

The Security Domain AID, if present, indicates to which Security Domain this Application is being extradited. The Security Domain with which this Application is currently associated is the currently selected Application. The Application AID shall be present except when restricting the functionality of OPEN.

For updating or revoking privileges the Privileges field shall be present. For updating implicit selection parameters or service parameters the corresponding tag within the Registry Update Parameters field shall be present - see section 11.5.2.3.7.

The Registry Update Token is mandatory for Delegated Management, and for Authorized Management if the off-card entity is not the Security Domain Provider. In all other cases an error status shall be returned if a Registry Update Token is present. The length field for the Registry Update Token is as defined for ASN.1 BER-TLV (see ISO 8825-1) except that the length 128 may also be coded on one byte as '80'.

11.5.2.3.6 Data Field for INSTALL [for personalization]

The following table details the INSTALL [for personalization] command data field.

Name	Length	Value	Presence
Length of data	1	'00'	Mandatory
Length of data	1	'00'	Mandatory
Length of Application AID	1	'05' - '10'	Mandatory
Application AID	5-16	'xxxx...'	Mandatory
Length of data	1	'00'	Mandatory
Length of data	1	'00'	Mandatory
Length of data	1	'00'	Mandatory

Table 11-47: INSTALL [for personalization] Command Data Field

11.5.2.3.7 INSTALL Command Parameters

The Load and Install Parameters fields are TLV structured values including optional System Specific Parameters and Application Specific Parameters. While the presence of the System Specific Parameters is optional for both loading and installation, even if they are present, it is not required that the system take heed of these i.e. their presence shall be anticipated but the content may be ignored.

In all cases, the presence of the data objects in tag 'B6' and the sub-tags '42', '45', 5F20' and '93' is strongly recommended when using asymmetric Secure Channel Protocol '10'.

In this version of the specification, the Token Issuer is assumed to be the Card Issuer.

The following table identifies the possible tags for use in the Load Parameters field of the INSTALL [for load] command:

Tag	Length	Name	Presence
'EF'	1-n	System Specific Parameters	Conditional
'C6'	2 or 4	Non volatile code Minimum Memory requirement	Optional
'C7'	2 or 4	Volatile data Minimum Memory requirement	Optional
'C8'	2 or 4	Non volatile data Minimum Memory requirement	Optional
'CD'	1	Load File Data Block format id	Optional
'DD'	1-n	Load File Data Block Parameters	Conditional
'D6'	2 or 4	Code Reserved Memory	Optional
'B6'	1-n	Control Reference Template for Digital Signature (Token)	Conditional

Tag	Length	Name	Presence
‘42’	1-n	Token Issuer id	Optional
'45'	1-n	Card Image Number	Optional
‘5F20’	1-n	Application Provider identifier	Optional
‘93’	1-n	Token identifier/number (digital signature counter)	Optional

Table 11-48: Load Parameter Tags

Some of the data objects in Load Parameters relate to memory management, which is an optional feature of a card. A Memory Management data object is a data object that represent an amount of memory resources counted in bytes. The minimum memory requirements (tags 'C6', 'C7' and 'C8') and code reserved memory (tag 'D6') are coded as a 2-byte integer for values up to 32767 and a 4-byte integer above 32767 - see section 9.7 - *Memory Resource Management*. When present, the Provider id (tag '5F20') shall be registered in the GlobalPlatform Registry.

The presence of Load File Data Block Parameters depends on the value of Load File Data Block format id.

The following table identifies the possible tags for use in the Install Parameters field of the INSTALL [for install] command:

Tag	Length	Name	Presence
'C9'	1-n	Application Specific Parameters	Mandatory
'EF'	1-n	System Specific Parameters	Conditional
'C7'	2 or 4	Volatile Memory Quota	Optional
'C8'	2 or 4	Non volatile Memory Quota	Optional
'CB'	2-n	Global Service Parameters	Optional
'D7'	2	Volatile Reserved Memory	Optional
'D8'	2	Non volatile Reserved Memory	Optional
'CA'	1-n	TS 102 226 specific parameter	Optional
‘CF’	1-n	Implicit selection parameter	See note below
'EA'	1-n	TS 102 226 specific template	Optional
'B6'	1-n	Control Reference Template for Digital Signature (Token)	Conditional
‘42’	1-n	Token Issuer id	Optional
'45'	1-n	Card Image Number	Optional
‘5F20’	1-n	Application Provider identifier	Optional
‘93’	1-n	Token identifier/number (digital signature counter)	Optional

Table 11-49: Install Parameter Tags

Note: tag 'CF' may be present in Install Parameters for the combined INSTALL [for install and make selectable] command and in the final combined INSTALL [for load, install and make selectable] command.

Some of the data objects in Install Parameters relate to memory management, which is an optional feature of a card. A Memory Management data object is a data object that represent an amount of memory resources counted in bytes. The memory quota values (tags 'C7' and 'C8') are coded as a 2-byte integer for values up to 32767 and a 4-byte integer above 32767 - see section 9.7 - *Memory Resource Management*.

When present, the Provider id (tag '5F20') shall be registered in the GlobalPlatform Registry.

When present in the INSTALL [for install] command, the information contained in tag 'C9' (length + data) shall be passed to the Application being installed.

When present in the INSTALL [for install] command, the information contained in tag 'CB' shall comprise one or more two-byte service parameters as defined in section 8.1.3 - *Global Service Parameters*.

The following table identifies the possible tags for use in the Make Selectable Parameters field of the INSTALL [for make selectable] command:

Tag	Length	Name	Presence
'EF'	1-n	System Specific Parameters	Conditional
‘CF’	1-n	Implicit selection parameter	Optional
'B6'	1-n	Control Reference Template for Digital Signature (Token)	Conditional
‘42’	1-n	Token Issuer id	Optional
'45'	1-n	Card Image Number	Optional
‘5F20’	1-n	Application Provider identifier	Optional
‘93’	1-n	Token identifier/number (digital signature counter)	Optional

Table 11-50: Make Selectable Parameter Tags

The following table identifies the possible tags for use in the Extradition Parameters field of the INSTALL [for extradition] command:

Tag	Length	Name	Presence
'B6'	1-n	Control Reference Template for Digital Signature (Token)	Conditional
‘42’	1-n	Token Issuer id	Optional
'45'	1-n	Card Image Number	Optional
‘5F20’	1-n	Application Provider identifier	Optional
‘93’	1-n	Token identifier/number (digital signature counter)	Optional

Table 11-51: Extradition Parameter Tags

The following table identifies the possible tags for use in the Registry Update Parameters field of the INSTALL [for registry update] command:

Tag	Length	Name	Presence
'EF'	1-n	System Specific Parameters	Conditional
'CF'	1	Implicit selection parameter	Optional
'CB'	2-n	Global Service Parameters	Optional
'D9'	1	Restrict Parameter (see Table 11-53)	Conditional

Tag	Length	Name	Presence
'B6'	1-n	Control Reference Template for Digital Signature (Token)	Conditional
'42'	1-n	Token Issuer id	Optional
'45'	1-n	Card Image Number	Optional
'5F20'	1-n	Application Provider identifier	Optional
'93'	1-n	Token identifier/number (digital signature counter)	Optional

Table 11-52: Registry Update Parameter Tags

When no Application AID is present in the INSTALL [for registry update] command, the Restrict Parameter (tag 'D9') applies to OPEN. The effects of this function shall be irreversible. When an Application AID is present in the INSTALL [for registry update] command, the Restrict Parameter (tag 'D9') applies to the specific Security Domain identified by the Application AID.

If the Restrict Parameter is present and the card does not implement the option the card shall reject the command.

The Restrict Parameter (tag 'D9') lists the functionalities that shall be disabled. Table 11-53 indicates the functionalities that could possibly be disabled.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
X	-	-	-	-	-	-	-	RFU
-	1	-	-	-	-	-	-	Registry Update
-	-	1	-	-	-	-	-	Personalization
-	-	-	1	-	-	-	-	Extradition
-	-	-	-	1	-	-	-	Make selectable
-	-	-	-	-	1	-	-	Install
-	-	-	-	-	-	1	-	Load
-	-	-	-	-	-	-	1	Delete

Table 11-53: Values for Restrict Parameter (Tag 'D9')

11.5.3 Response Message

A data field shall always be returned in the response message. Confirmation data or a single byte of '00' may be returned.

11.5.3.1 Data Field Returned in the Response Message

If an INSTALL [for load] command, an INSTALL [for make selectable] command alone or an INSTALL [for personalization] command is being issued, a single byte of '00' shall be returned indicating that no additional data is present.

For INSTALL [for install], INSTALL [for install and make selectable] and INSTALL [for registry update] commands being issued to a Security Domain with the Delegated Management privilege, the data field may contain the confirmation of the install procedure. The overall length of the response message shall not exceed 256 bytes.

The following table describes the structure of the INSTALL response data field. The length field for the Install Confirmation is coded according to ASN.1 BER-TLV (see ISO 8825-1).

Name	Length	Value	Presence
Length of Install Confirmation	1-2	'00' - '7F' or '81 80' - '81 FF'	Mandatory
Install Confirmation	0-n	'xxxx...' - see section 11.1.6	Conditional

Table 11-54: INSTALL Response Data Field

11.5.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'65'	'81'	Memory failure
'6A'	'80'	Incorrect parameters in data field
'6A'	'84'	Not enough memory space
'6A'	'88'	Referenced data not found

Table 11-55: INSTALL Error Conditions

11.6 LOAD Command

11.6.1 Definition and Scope

This section defines the structure of the Load File transmitted in the LOAD command data field for loading a Load File. The runtime environment internal handling or storage of the Load File is beyond the scope of this Specification.

Multiple LOAD commands may be used to transfer a Load File to the card. The Load File is divided into smaller components for transmission. Each LOAD command shall be numbered starting at '00'. The LOAD command numbering shall be strictly sequential and increments by one. The card shall be informed of the last block of the Load File.

After receiving the last block of the Load File, the card shall execute the internal processes necessary for the Load File and any additional processes identified in the INSTALL [for load] command that preceded the LOAD commands.

11.6.2 Command Message

The LOAD command message is coded according to the following table.

Code	Value	Meaning
CLA	'80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'E8'	LOAD
P1	'xx'	Reference control parameter P1
P2	'xx'	Block number
Lc	'xx'	Length of data field
Data	'xxxx..'	Load data (and MAC if present)
Le	'00'	

Table 11-56: LOAD Command Message Structure

11.6.2.1 Reference Control Parameter P1

The following table describes the coding of the reference control parameter P1 indicating whether the block contained in the command message is one in a sequence of blocks or the last block in the sequence.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	More blocks
1	-	-	-	-	-	-	-	Last block
-	X	X	X	X	X	X	X	RFU

Table 11-57: LOAD Command Reference Control Parameter P1

11.6.2.2 Reference Control Parameter P2 – Block Number

Reference control parameter P2 contains the block number, and shall be coded sequentially from '00' to 'FF'.

11.6.2.3 Data Field Sent in the Command Message

The data field of the command message contains a portion of the Load File.

A complete GlobalPlatform Load File is structured as defined in the following table:

Tag	Length	Name	Presence
'E2'	1-n	DAP Block	Conditional
'4F'	5-16	Security Domain AID	Conditional
'C3'	1-n	Load File Data Block Signature	Conditional
:	:	:	
:	:	:	
'E2'	1-n	DAP Block	Conditional
'4F'	5-16	Security Domain AID	Conditional
'C3'	1-n	Load File Data Block Signature	Conditional
'C4'	1-n	Load File Data Block	Mandatory

Table 11-58: Load File Structure

The data objects defined in Table 11-58 shall be coded according to ASN.1 Basic Encoding Rules, and in particular, the length fields (e.g. one byte for length values up to 127).

A DAP Block is optional within a Load File unless the associated Security Domain has the DAP Verification privilege or a Security Domain with the Mandated DAP Verification privilege is present.

A Load File may contain multiple DAP Blocks; each DAP Block being introduced by tag 'E2' and preceding the Load File Data Block.

11.6.3 Response Message

A data field shall always be returned in the response message. The content of the data field is only relevant in the case of Delegated Management i.e. if a last LOAD command is being issued to a Security Domain with the Delegated Management privilege, a Receipt may be present in the data field depending on the security policy of the Card Issuer.

11.6.3.1 Data Field Returned in the Response Message

If the LOAD command does not contain the last block in the sequence, a single byte of '00' shall be returned indicating that no additional data is present.

If the Issuer Security Domain processes the LOAD command containing the last block in the sequence, a single byte of '00' shall be returned indicating that no additional data is present.

For a LOAD command containing the last block in the sequence being issued to a Security Domain with the Delegated Management privilege, the data field may contain the confirmation of the load procedure. The overall length of the response message shall not exceed 256 bytes.

The following table describes the structure of the LOAD response data field. The length field for the Load Confirmation is coded according to ASN.1 BER-TLV (see ISO 8825-1).

Name	Length	Value	Presence
Length of Load Confirmation	1-2	'00' - '7F' or '81 80' - '81 FF'	Mandatory

Name	Length	Value	Presence
Load Confirmation	0-n	'xxxx...' - see section 11.1.6	Conditional

Table 11-59: LOAD Response Data Field

11.6.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'65'	'81'	Memory failure
'6A'	'84'	Not enough memory space

Table 11-60: LOAD Error Conditions

11.7 MANAGE CHANNEL Command

11.7.1 Definition and Scope

The MANAGE CHANNEL command is processed by the OPEN on cards that are aware of logical channels. It is used to open and close Supplementary Logical Channels. The Basic Logical Channel (channel number zero) can never be closed.

11.7.2 Command Message

The MANAGE CHANNEL command message shall be coded according to the following table.

Code	Value	Meaning
CLA	'00' - '03' or '40' - '4F'	See section 11.1.4
INS	'70'	MANAGE CHANNEL
P1	'xx'	Reference control parameter P1
P2	'xx'	Reference control parameter P2
Lc		Not present
Le		'01' if P1 = '00' and not present if P1 = '80'

Table 11-61: MANAGE CHANNEL Command Message

11.7.2.1 Reference Control Parameter P1

Reference control parameter P1 is used to indicate whether a Supplementary Logical Channel is being opened or closed.

The following values of the reference control parameter may apply:

'80' – Close the Supplementary Logical Channel identified in reference control parameter P2.

'00' – Open the next available Supplementary Logical Channel.

11.7.2.2 Reference Control Parameter P2

If a Supplementary Logical Channel is being closed (reference control parameter P1 is '80'), the reference control parameter P2 identifies the Supplementary Logical Channel to be closed (i.e. '01', '02' or '03'). For a card supporting further interindustry logical channels, the reference control parameter may identify further interindustry Supplementary Logical Channels (i.e. ‘04’ to ‘13’).

If a Supplementary Logical Channel is being opened (reference control parameter P1 is '00'), the reference control parameter P2 indicates that the next available Supplementary Logical Channel is being opened (i.e. '00').

A card supporting logical channel assignment by off-card entities may accept a reference control parameter P2 indicating the Supplementary Logical Channel number to be opened (i.e. '01', '02' or '03'). A card supporting further interindustry logical channel assignment by off-card entities may accept a reference control parameter P2 indicating the further interindustry Supplementary Logical Channel number to be opened (i.e. ‘04’ to ‘13’).

11.7.2.3 Data Field Sent in the Command Message

The data field of the command message is not present.

11.7.3 Response Message

11.7.3.1 Data Field Returned in the Response Message

The data field of the response message is only present if a Supplementary Logical Channel is being opened.

Depending on the number of logical channels supported by the card, the following values of the data field of the response message may apply:

'01', '02' or '03' – Supplementary Logical Channel opened.

For a card supporting further interindustry logical channels, and depending on the number of Supplementary Logical Channels supported by the card, the following values of the data field of the response message may apply:

'04' to '13' – Supplementary Logical Channel opened.

11.7.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may return the following warning:

SW1	SW2	Meaning
'62'	'00'	Logical Channel already closed

Table 11-62: MANAGE CHANNEL Warning Conditions

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'68'	'82'	Secure messaging not supported
'6A'	'81'	Function not supported e.g. card Life Cycle State is CARD_LOCKED

Table 11-63: MANAGE CHANNEL Error Conditions

11.8 PUT KEY Command

11.8.1 Definition and Scope

The PUT KEY command is used to either:

- Replace an existing key with a new key: The new key has the same or a different Key Version Number but the same Key Identifier as the key being replaced;
- Replace multiple existing keys with new keys: The new keys have the same or a different Key Version Number (identical for all new keys) but the same Key Identifiers as the keys being replaced;
- Add a single new key: The new key has a different combination Key Identifier / Key Version Number than that of the existing keys;
- Add multiple new keys: The new keys have different combinations of Key Identifiers / Key Version Number (identical to all new keys) than that of the existing keys;

When the key management operation requires multiple PUT KEY commands, chaining of the multiple PUT KEY commands is recommended to ensure integrity of the operation.

In this version of the Specification the public values of asymmetric keys are presented in clear text.

11.8.2 Command Message

The PUT KEY command message is coded according to the following table:

Code	Value	Meaning
CLA	'80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'D8'	PUT KEY
P1	'xx'	Reference control parameter P1
P2	'xx'	Reference control parameter P2
Lc	'xx'	Length of data field
Data	'xxxx..'	Key data (and MAC if present)
Le	'00'	

Table 11-64: PUT KEY Command Message

11.8.2.1 Reference Control Parameter P1

Reference control parameter P1 defines a Key Version Number and whether more PUT KEY commands will follow this one.

The Key Version Number identifies a key or group of keys that is already present on the card. A value of '00' indicates that a new key or group of keys is being added. (The new Key Version Number is indicated in the data field of the command message).

The Key Version Number is coded from '01' to '7F'.

The reference control parameter P1 of the PUT KEY command message is coded according to the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	Last (or only) command
1	-	-	-	-	-	-	-	More PUT KEY commands
-	X	X	X	X	X	X	X	Key Version Number

Table 11-65: PUT KEY Reference Control Parameter P1

11.8.2.2 Reference Control Parameter P2

Reference control parameter P2 defines a Key Identifier and whether one or multiple keys are contained in the data field.

When one key is contained in the command message data field, reference control parameter P2 indicates the Key Identifier of this key. When multiple keys are contained in the command message data field, reference control parameter P2 indicates the Key Identifier of the first key in the command data field. Each subsequent key in the command message data field has an implicit Key Identifier that is sequentially incremented by one, starting from this first Key Identifier.

The Key Identifier is coded from '00' to '7F'.

The reference control parameter P2 of the PUT KEY command message is coded according to the following table:

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	Single key
1	-	-	-	-	-	-	-	Multiple keys
-	X	X	X	X	X	X	X	Key Identifier

Table 11-66: PUT KEY Reference Control Parameter P2

11.8.2.3 Data Field Sent in the Command Message

The command message data field contains a new Key Version Number (coded from '01' to '7F') followed by one or multiple key data fields as represented in the following diagram (with optionally a second and third keys):

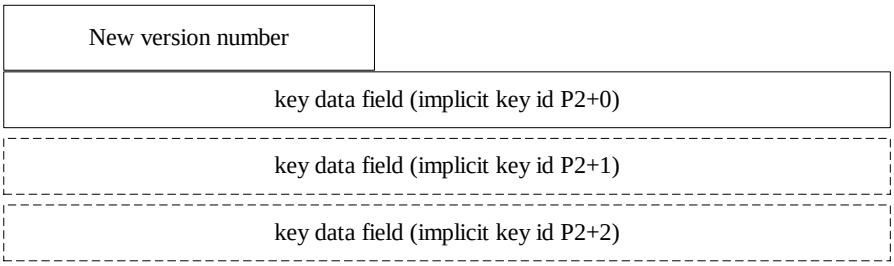

Table 11-67: Key Version Number Diagram

The new Key Version Number defines either:

- The version number of a new key or group of keys to be created on the card (Key Version Number indicated in P1 is set to zero); or
- The version number of a new key or group of keys that will replace an existing key or group of keys (Key Version Number indicated in P1 is different from zero).

If the data field contains multiple keys, the keys all share the same Key Version Number and the sequence in the command data field reflects the incremental sequence of the Key Identifiers.

The format of the key data field depends on the value of the first data element, Key Type.

11.8.2.3.1 Format 1

If the Key Type is coded on one byte (value other than 'FF'), the key data field is structured according to the following table:

Name	Length	Value	Presence
Key type	1-2	'00' - 'FE' - see section 11.1.8 - *Key Type Coding*	Mandatory
Length of key or key component	1-3	'01' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Mandatory
Key or key component data value	1-n	'xxxx...'	Mandatory
Length of key check value	1	'00' - '7F'	Mandatory
Key check value	0-n	'xxxx...'	Conditional

Table 11-68: Key Data Field - Basic

11.8.2.3.2 Format 2

If the Key Type is coded on two bytes (first byte = 'FF'), the key data field is structured according to the following table:

Name	Length	Value	Presence
Key type of the first or only key component	2	'FF xx' - see section 11.1.8 - *Key Type Coding*	Mandatory
Length of first or only key component	1-3	'01' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Mandatory
Key or first key component data value	1-n	'xxxx...'	Mandatory
	...		
Key type of the last key component, if more than one	2	'FF xx' - see section 11.1.8 - *Key Type Coding*	Conditional
Length of last key component	1-3	'01' - '80', or '81 80' - '81 FF', or '82 01 00' - '82 FF FF'	Conditional
Last key component data value	1-n	'xxxx...'	Conditional
Length of key check value	1	'00' - '7F'	Mandatory
Key check value (if present)	0-n	'xxxx...'	Conditional
Length of key usage	1	'00' - '7F'	Mandatory
Key usage	0-n	'xxxx...'	Conditional
Length of key access	1	'00' - '7F'	Mandatory
Key access	0-n	'xxxx...'	Conditional

Table 11-69: Key Data Field - Extended

The length of each key component is coded as defined for ASN.1 BER-TLV (see ISO 8825-1) except that the length 128 may also be coded on one byte as '80'.

11.8.2.3.3 Processing rules

When replacing keys, the new keys shall be presented to the card in the same format as they are already present on the card: in other words, it is not possible to change the size and the associated cryptographic algorithm of an existing key slot.

When using this command to load or replace secret or private keys, the key values shall be encrypted and the reference of the encrypting key and algorithm to be used is known implicitly according to the current context. Public key values may be presented in clear text.

When chaining is used to load or replace a key comprised of more than one component, the subsequent commands must refer to the same Key Identifier and the same Key Version Number as the first PUT KEY command used for the first key component. A key component shall not be split across two PUT KEY commands.

If the data field contains multiple keys or key components, the card must handle the multiple keys or key components in an atomic manner. When PUT KEY commands are chained (i.e. bit b8 of P1 set to 1), the card must handle the multiple key components transferred in the chain of PUT KEY commands (until and including the first PUT KEY command with bit b8 of P1 = 0) in an atomic manner.

The PUT KEY command creates or updates the Key Information Data which may later be returned in tag 'C0', structured as shown in Table 11-27 or Table 11-28.

The modulus component of an RSA key should be the first key component.

It is the responsibility of the receiving on-card entity to verify the key check value when present.

11.8.3 Response Message

11.8.3.1 Data Field Returned in the Response Message

The data field of the response message contains in clear text the Key Version Number followed by the key check value(s) not preceded by a length, if any, as presented in the command message data field. The personalization server may use the returned Key Version Number and key check value(s) to verify the correct loading of the key(s).

11.8.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'65'	'81'	Memory failure
'6A'	'84'	Not enough memory space
'6A'	'88'	Referenced data not found
'94'	'84'	Algorithm not supported
'94'	'85'	Invalid key check value

Table 11-70: PUT KEY Error Conditions

11.9 SELECT Command

11.9.1 Definition and Scope

The SELECT command is used for selecting an Application. The OPEN only processes SELECT commands indicating the SELECT [by name] option. All options other than SELECT [by name] shall be passed to the currently selected Security Domain or Application on the indicated logical channel.

11.9.2 Command Message

The SELECT command is coded according to the following table:

Code	Value	Meaning
CLA	'00' - '03' or '40' - '4F'	See section 11.1.4
INS	'A4'	SELECT
P1	'xx'	Reference control parameter P1
P2	'xx'	Reference control parameter P2
Lc	'xx'	Length of AID
Data	'xxxx..'	AID of Application to be selected
Le	'00'	

Table 11-71: SELECT Command Message

11.9.2.1 Reference Control Parameter P1

Reference control parameter P1 shall be coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	1	0	0	Select by name

Table 11-72: SELECT Reference Control Parameter P1

A Security Domain or Application may support other values as defined in ISO/IEC 7816-4.

11.9.2.2 Reference Control Parameter P2

Reference control parameter P2 shall be coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	0	0	0	0	0	0	0	First or only occurrence
0	0	0	0	0	0	1	0	Next occurrence

Table 11-73: SELECT Reference Control Parameter P2

OPEN or the underlying runtime environment may support other values as defined in ISO/IEC 7816-4.

11.9.2.3 Data Field Sent in the Command Message

The data field of the command shall contain the AID of the Application to be selected. The Lc and data field of the SELECT command may be omitted if the Issuer Security Domain is being selected. In this case, Le shall be set to '00' and the command is a case 2 command according to ISO/IEC 7816-4.

11.9.3 Response Message

11.9.3.1 Data Field Returned in the Response Message

The SELECT response data field consists of information specific to the selected Application.

The coding of the File Control Information for the Issuer Security Domain and Security Domains shall be according to the following table.

Tag	Description	Presence
'6F'	File Control Information (FCI template)	Mandatory
'84'	Application / file AID	Mandatory
'A5'	Proprietary data	Mandatory
'73'	Security Domain Management Data (see appendix H.3 for detailed coding)	Optional
'9F6E'	Application production Life Cycle data	Optional
'9F65'	Maximum length of data field in command message	Mandatory

Table 11-74: File Control Information

Additional data objects may be returned within the FCI template.

11.9.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may return the following warning condition when the Security Domain with the Final Application privilege is being selected.

SW1	SW2	Meaning
'62'	'83'	Card Life Cycle State is CARD_LOCKED

Table 11-75: SELECT Warning Condition

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'68'	'82'	Secure messaging not supported
'6A'	'81'	Function not supported e.g. card Life Cycle State is CARD_LOCKED
'6A'	'82'	Selected Application / file not found

Table 11-76: SELECT Error Conditions

11.10 SET STATUS Command

11.10.1 Definition and Scope

The SET STATUS command shall be used to modify the card Life Cycle State or the Application Life Cycle State.

11.10.2 Command Message

The SET STATUS command message is coded according to the following table.

Code	Value	Meaning
CLA	'80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'F0'	SET STATUS
P1	'xx'	Status type
P2	'xx'	State control
Lc	'xx'	Length of data field
Data	'xxxxx…'	AID of Application (and MAC if present)
Le		Not present

Table 11-77: SET STATUS Command Message

11.10.2.1 Reference Control Parameter P1 - Status Type

The status type of the SET STATUS command message indicates if the change in the Life Cycle State applies to the Issuer Security Domain, Supplementary Security Domains or an Application. The status type can also indicate that the command applies to a Security Domain and all its associated Applications: this only applies for transition to, and back from, the LOCKED state. The status type shall be coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
1	0	0	-	-	-	-	-	Indicate Issuer Security Domain
0	1	0	-	-	-	-	-	Indicate Application or Supplementary Security Domain
0	1	1	-	-	-	-	-	Indicate Security Domain and its associated Applications
-	-	-	X	X	X	X	X	RFU

Table 11-78: SET STATUS – Status Type

11.10.2.2 Reference Control Parameter P2 - State Control

The state control of the SET STATUS command message indicates the state transition required and shall be coded according to section 11.1.1 - *Life Cycle State Coding.*

For the Issuer Security Domain (which inherits the card Life Cycle State), the parameter shall be coded according to Table 11-6 and abide by the transitioning rules as diagrammed in Figure 5-1.

For a Security Domain setting its own Life Cycle State using this command, the only possible transitions are to the PERSONALIZED or LOCKED state: The parameter shall be coded according to Table 11-5.

For an Application setting its own Life Cycle State using this command, the parameter shall be coded according to Table 11-4 and abide by the transitioning rules as depicted in Figure 5-2.

For a Security Domain setting the Life Cycle State of another Application or Security Domain, the only possible transitions are to the LOCKED state and subsequently back to the previous state. The only relevant bit of this parameter would therefore be bit b8 (all other bits are ignored):

- b8 = 1 indicates a transition to the LOCKED state;
- b8 = 0 indicates a transition (from LOCKED) back to the previous state.

A request to transition a Security Domain or an Application to its current Life Cycle State shall be rejected.

11.10.2.3 Data Field Sent in the Command Message

The data field shall contain the AID of the target Application or Security Domain for which a Life Cycle change is requested. The on-card entity receiving the command shall be directly or indirectly associated with this target Application or Security Domain or shall have the relevant privilege. If reference control parameter P1 is '80' the content of the command data field shall be ignored. If the command relates to a Security Domain and all its associated Applications, the data field shall contain the AID of the Security Domain.

11.10.3 Response Message

11.10.3.1 Data Field Returned in the Response Message

The data field of the response message shall not be present.

11.10.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following error conditions.

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in command data
'6A'	'88'	Referenced data not found

Table 11-79: SET STATUS Error Conditions

11.11 STORE DATA Command

11.11.1 Definition and Scope

The STORE DATA command is used to transfer data to an Application or the Security Domain processing the command.

The Security Domain determines if the command is intended for itself or an Application depending on a previously received command. If a preceding command was an INSTALL [for personalize] command, the STORE DATA command is destined for an Application.

Multiple STORE DATA commands transfer data to the Application or Security Domain by breaking the data into smaller components for transmission. Each STORE DATA command is numbered starting at '00'. The STORE DATA command numbering shall be strictly sequential and increments by one. The Security Domain shall be informed of the last block.

11.11.2 Command Message

The STORE DATA command message shall be coded according to the following table.

Code	Value	Meaning
CLA	'80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'E2'	STORE DATA
P1	'xx'	Reference control parameter P1
P2	'xx'	Block number
Lc	'xx'	Length of data field
Data	'xxxxx…'	Application data and MAC (if present)
Le		Not present

Table 11-80: STORE DATA Command Message

11.11.2.1 Reference Control Parameter P1

Reference control parameter P1 shall be coded according to the following table.

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
0	-	-	-	-	-	-	-	More blocks
1	-	-	-	-	-	-	-	Last block
-	0	0	-	-	-	-	-	No general encryption information or non-encrypted data
-	0	1	-	-	-	-	-	Application dependent encryption of the data
-	1	0	-	-	-	-	-	RFU (encryption indicator)
-	1	1	-	-	-	-	-	Encrypted data
-	-	-	0	0	-	-	-	No general data structure information
-	-	-	0	1	-	-	-	DGI format of the command data field
-	-	-	1	0	-	-	-	BER-TLV format of the command data field
-	-	-	1	1	-	-	-	RFU (data structure information)

b8	b7	b6	b5	b4	b3	b2	b1	Meaning
-	-	-	-	-	X	X	X	RFU

Table 11-81: STORE DATA Reference Control Parameter P1

Bits b5 and b4 provide information on the data structure of the command message data field.

- b5 - b4 = 00 indicate that no general information on the data structure is provided, i.e. the data structure is Application dependent;
- b5 - b4 = 01 indicate that the command message data field is coded as one or more DGI structures, according to GlobalPlatform Systems Scripting Specification;
- b5 - b4 = 10 indicate that the command message data field is coded as one or more BER-TLV structures, according to ISO 8825.

Bits b7 and b6 provide information on the encryption of the value fields of the data structure present in the command message data field.

- b7 – b6 = 00 indicate that no general information on the data encryption is provided, i.e. the encryption (or non-encryption) of the data is Application dependent, or that the data value fields of all the data structures present in the current command message are not encrypted;
- b7 – b6 = 01 indicate that the encryption (or non-encryption) of the data structure value fields is Application dependent, e.g. when multiple data structures are present in the current command message, some may have encrypted data value fields and other data value fields may be non-encrypted;
- b7 – b6 = 11 indicate that the data value fields of all the data structures present in the current command message are encrypted.

The decryption of application-specific data is the responsibility of the Application.

11.11.2.2 Reference Control Parameter P2 - Block number

Reference control parameter P2 shall contain the block number coded sequentially from '00' to 'FF'. The Security Domain shall check the sequence of commands.

11.11.2.3 Data Field Sent in the Command Message

The data field shall contain data in a format expected by the Security Domain or the Application. If the data is intended for an Application, it is sent to the Application as described in section 7.3.3.

Three data structuring modes can be defined:

- Application dependent format applies when no information is available on the format of the incoming command data: bits b5-b4 of reference control parameter P1 are set to ‘00’. In this case, information on the encryption (or non-encryption) of the incoming command data is usually not available (parameter P1 bits b7-b6 set to ‘00’): the format and eventual encryption of the incoming command data are implicitly known by the Application;
- DGI formatting applies when all data structures that are present in the command data field are formatted as DGI structures (as defined in the Scripting Specification): bits b5-b4 of reference control parameter P1 are set to ‘01’. In this case, some information may be available on the encryption (or non-encryption) of the value fields of the DGI data structures: reference control parameter P1 bits b7-b6 are set accordingly;
- BER-TLV formatting applies when all data structures that are present in the command data field are formatted as BER-TLV structures (as defined in ISO 8825): bits b5-b4 of reference control parameter P1

are set to ‘10’. In this case, some information may be available on the encryption (or non-encryption) of the value fields of the TLV data structures: reference control parameter P1 bits b7-b6 are set accordingly.

If the overall length of the intended command message exceeds 255 bytes, the individual (or group of) data shall be sent in multiple consecutive STORE DATA commands. Whether the data format is a DGI or BER-TLV data structure, the following rules shall apply:

- The data structure length indicators shall always reflect the actual full length of the data structure value field;
- The data structure value field shall be truncated in the STORE DATA command message containing the data structure length indicator (e.g. at the maximum length of the command message);
- The subsequent STORE DATA command shall contain the remainder of the data structure value field (that may be followed by one or more data structure(s) in this same command message) – note: for very large data, more than one subsequent STORE DATA command message may be required for the remainder of the data structure value field;
- The Target Application or Security Domain shall use the last data structure length indicator of a STORE DATA command message to determine whether a subsequent STORE DATA command is expected to contain the remainder of the data structure value field.

The Issuer Security Domain shall support at least the following TLV coded data objects:

- Issuer Identification Number (tag '42')
- Card Image Number (tag '45')
- Issuer Security Domain AID (tag '4F')
- Card Data (tag '66')

A Supplementary Security Domain should support the following TLV coded data objects:

- Security Domain Provider Identification Number (tag '42')
- Security Domain Image Number (tag '45')
- Security Domain Management Data (tag '66')

When DGI formatting is supported, these data objects may be included in a DGI. In that case, the DGI ‘0070’ shall be used. Otherwise, these data objects shall be presented in their BER-TLV format.

11.11.3 Response Message

11.11.3.1 Data Field Returned in the Response Message

The data field of the response message shall not be present.

11.11.3.2 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or the following error condition.

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in command data

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table 11-82: STORE DATA Error Condition

Appendices

A GlobalPlatform API

A.1 GlobalPlatform on a Java Card

This section contains only the API required for GlobalPlatform 2.2.x Java Cards. Use of the Open Platform 2.0.1' API is still allowed for supporting older versions of Applications but is deprecated. It is defined in version 2.1.1 of this Specification.

The deprecated API and new API both access the same objects where applicable. While this may seem obvious for methods that have the same name across both classes (e.g. `setATRHistBytes()`, `setCardContentState()` and `getCardContentState()`) it shall also be noted as for example that an application that uses the `update()` method in the new API to change the value of the global PIN will affect the same global PIN of an application that uses the `setPIN()` method in the deprecated API to verify the global PIN.

GlobalPlatform Specific Requirements

In order to ensure the highest level of interoperability of GlobalPlatform implementations, GlobalPlatform also adopts the order defined in section 6.2 - *Java Card™ 2.2.x Virtual Machine Specification*.

Some of the minor modifications to the standard functionality defined in the *Java Card™ 2.1.1 Runtime Environment (JCRE) Specifications* and the *Java Card™ 2.1.1 Application Programming Interface* are no longer relevant with the Java Card™ 2.2.x specifications.

GPRegistryEntry objects shall be implemented as Shareable Interface Objects as defined in section 6.2.4 '*Shareable Interfaces*' of the *Java Card™ 2.2.x Runtime Environment (JCRE) Specification* in order to ensure shared access.

GlobalPlatform package AID

Each GlobalPlatform package AID will be a concatenation of a RID and a PIX. The AID value of the Java Card Export File for the new GlobalPlatform API (identical for both GlobalPlatform 2.1 and 2.1.1) based on the RID specified in appendix H - *GlobalPlatform Data Values and Card Recognition Data* is 'A00000015100'.

Installation

In section 3.1 - *The Method* `install` of the *Java Card™ 2.2.x Runtime Environment (JCRE) Specifications*, the parameters passed to the method are defined to be initialization parameters from the contents of the incoming byte array parameter.

This specification expands on this requirement and further defines the content of the Install Parameters. This expansion affects both the implementation of an OPEN and the behavior of a Java Card applet developed for a GlobalPlatform card. It does not affect the definition of the `install` method of the Class Applet of the *Java Card™ 2.2.x Application Programming Interface* specification.

The Install Parameters shall identify the following data, present in the INSTALL [for install] command (see section 11.5.2.3.2 - *Data Field for INSTALL [for install]*):

- The instance AID;
- The Privileges;

- The Application Specific Parameters[1].

The OPEN is responsible for ensuring that the parameters (`bArray, bOffset` and `bLength`) contain the following information:

The array, `bArray`, shall contain the following consecutive LV coded data:

- Length of the instance AID;
- The instance AID;
- Length of the Privileges;
- The Privileges;
- Length of the Application Specific Parameters;
- The Application Specific Parameters.

The byte, `bOffset`, shall contain an offset within the array pointing to the length of the instance AID.

The byte, `bLength`, shall contain a length indicating the total length of the above-defined data in the array.

The applet is required to utilize the instance AID as a parameter when invoking the `register (byte [] bArray, short bOffset, byte bLength)` method of the Class Applet of the *Java Card™ 2.2.x Application Programming Interface* specification.

T=0 Transmission Protocol

GlobalPlatform cards are intended to be functional in the widest range of environments (i.e. Card Acceptance Devices). Currently the *Java Card™ 2.2.x Runtime Environment (JCRE) Specifications* describe the behavior for case 2 commands (when using the T=0 protocol) in contradiction to EMV 2000. GlobalPlatform mandates that the JCRE shall handle this case of command in accordance with ISO/IEC 7816: An applet receiving a case 2 command builds the response and invokes the appropriate API to output the data. If the data is less than the data expected by the terminal, the OPEN will store the data and output a '6Cxx' response code and wait for the CAD to re-issue the command with the correct length. When the re-issued command is received the JCRE will manage the outputting of the stored data.

Atomicity

Unless otherwise specified all internal persistent objects of the GlobalPlatform API must conform to a transaction in progress.

All operations performed by this API, except the `Application.processData()` method shall be executed atomically. Objects used to enforce the implementation of velocity checking shall not conform to a transaction in progress.

Logical channels

The following logical channel restrictions apply to Java Card™ 2.2.x (see the *Java Card™ 2.2.x Runtime Environment (JCRE) Specifications* for more detail):

Selection of an Application on a logical channel as defined in section 6.3 - *Command Dispatch* will be unsuccessful if this same Application, or any other Application instantiated from code in the same package from which the

[1] While the APDU command contains Install Parameters representing TLV coded system and Application Specific Parameters, the application only requires knowledge of the Application Specific Parameters i.e. only LV of the TLV coded structure ‘C9’ are present as parameters.

Application being selected was instantiated, is currently selected on another logical channel but the application code does not implement the `MultiSelectable` interface.

Changing context from a Security Domain to an Application as defined in section 7.3.3 - *Personalization Support* will be unsuccessful if this same Application, or any other Application instantiated from code in the same package from which the Application being personalized was instantiated, is currently selected on another logical channel but the application code does not implement the `MultiSelectable` interface.

An Application that has the Card Reset privilege and is intended for a card that supports Supplementary Logical Channels should implement the `MultiSelectable` interface.

GlobalPlatform only defines the assignment of logical channel numbers by the card. Optionally and as defined in Java Card 2.2, a card may also support assignment of logical channel numbers by the terminal.

Cryptographic Algorithms

GlobalPlatform cards supporting RSA cryptography should support key sizes not defined as constants in the Key Builder class. More specifically support for key sizes being a multiple of 4 bytes (32 bits), and being within the allowed key lengths defined by the implementation, should be available.

Level of trust

The Java Card 2.2.x specifications assume that the RID of the AID of packages, applets and instances will be utilized to ensure a level of trust between these entities. In section 4.2.2 - *AID Usage* of the *Java Card™ 2.2.1 Application Programming Interface* it is defined that the RID of an AID of a component must match the RID of the AID of the package and in the definition of the `register (byte [] bArray, short bOffset, byte bLength)` method of the *Java Card™ 2.2.x Application Programming Interface* specification it is defined that an exception must be thrown if the RID portion of the AID bytes in the `bArray` parameter does not match the RID portion of the Java Card name of the applet.

From a real world implementation point of view, mandating that the RID of the instance AID must be the same as the RID of the component from which it was instantiated, is not practical. GlobalPlatform implementations shall not mandate that there be any link through the AID of an instance to its original package. It does however assume that all applications in the same package share the same level of trust.

Invocation of GlobalPlatform methods

The Application Programming Interface defined herein is accessible to any Java Card applet developed with the intention of being present on a GlobalPlatform card. One limitation does exist relating to the constructor of the applet and to the `install()` method of the Class Applet of the *Java Card™ 2.2.x Application Programming Interface.* As this specification does not define exactly when the instance of an applet becomes an entry in the card's GlobalPlatform Registry, an applet developer can only assume that this has occurred following the successful completion of the install method. To ensure interoperability, GlobalPlatform API methods that require access to the GlobalPlatform Registry entry of the applet invoking the method, shall not be invoked before registering the applet.

The following is a list of methods that may be invoked from within the constructor or the `install()` method:

- getCardState;
- getCVM;
- getService.

The required behavior of the card in the event that an Application incorrectly invokes a method of the org.globalplatform.GPSystem class other than those listed above is undefined. For example, an exception may be thrown and the processing of the `install()` method may be aborted.

Selection

On GlobalPlatform cards, Java Card applets are requested not to fail processing the select() method (e.g. throwing an exception) nor return false from the select() method. If an error occurs during the processing of the select() method, the selection process under progress is aborted by the JCRE and a Java Card specific error is returned to the off-card entity as specified in the Java Card Runtime Environment Specifications. Such situation leaves the corresponding logical channel open with no currently selected Application. Since no Application is currently selected, any subsequent command, other than a MANAGE CHANNEL or SELECT [by name] command, will be rejected. It is expected that the off-card entity will take appropriate action on such an error, e.g. select another Application, close the corresponding logical channel, reset or power off the card. The exception is where an applet may fail the select() method if it does not support the current card interface.

Class Hierarchy

- class java.lang.Object
  - class org.globalplatform.GPSystem

org.globalplatform

Class **GPSystem**

java.lang.Object
|
+-**org.globalplatform.GPSystem**

`public class` **GPSystem** `extends java.lang.Object`

This class exposes a subset of the behavior of the OPEN to the outside world. The OPEN implements and enforces a Card Issuer's security policy relating to these services. This OPEN class provides functionality at the same level as the JCRE i.e. the "system" context with special privileges. This class's public interface is composed of static methods visible to all applets importing the globalplatform package.

Field Summary

final static byte	APPLICATION_INSTALLED The applet Life Cycle State is INSTALLED (0x03).
final static byte	APPLICATION_LOCKED The applet Life Cycle State is LOCKED (0x80).
final static byte	APPLICATION_SELECTABLE The applet Life Cycle State is SELECTABLE (0x07).
final static byte	CARD_INITIALIZED The card is in the Life Cycle State of INITIALIZED (0x07).
final static byte	CARD_LOCKED The card is in the Life Cycle State of CARD_LOCKED (0x7F).
final static byte	CARD_OP_READY The card is in the Life Cycle State of OP_READY (0x01).
final static byte	CARD_SECURED The card is in the Life Cycle State of SECURED (0x0F).
final static byte	CARD_TERMINATED The card is in the Life Cycle State of TERMINATED (0xFF).
final static byte	CVM_GLOBAL_PIN Indicates that the `CVM` interface required is a global PIN (0x11).

final static byte	FAMILY_SECURE_CHANNEL Indicates the family of the Secure Channel Global Service Identifier (0x81).
final static byte	FAMILY_CVM Indicates the family of the CVM Global Service Identifier (0x82).
final static byte	FAMILY_USSM Indicates the family of the USSM Global Service Identifier (0xA0).
final static byte	GLOBAL_SERVICE_IDENTIFIER Indicates the generic Global Service Identifier (0x80).
final static byte	SECURITY_DOMAIN_PERSONALIZED The current Security Domain is in the Life Cycle State of PERSONALIZED (0x0F).

Method Summary

static byte	**getCardContentState**() This method returns the Life Cycle State of the current applet context.
static byte	**getCardState**() This method returns the Life Cycle State for the card.
static org.globalplatform.CVM	**getCVM**(byte bCVMIdentifier) This method returns a handle to the CVM interface.
static org.globalplatform.GPRegistryEntry	**getRegistryEntry**(javacard.framework.AID reqAID) This method returns a handle to the GPRegistryEntry interface.
static org.globalplatform.SecureChannel	**getSecureChannel**() This method returns a handle to the SecureChannel interface.
static org.globalplatform.GlobalService	**getService**(javacard.framework.AID serverAID, short sServiceName) This method returns a handle to the Global Services interface of a Global Services Application.
static boolean	**lockCard**() This method locks the card.

Method Summary

static boolean	**setATRHistBytes**(byte[] baBuffer, short sOffset, byte bLength) This method sets the historical bytes.
static boolean	**setCardContentState**(byte bState) This method sets the Application specific Life Cycle State of the current applet context.
static boolean	**terminateCard**() This method terminates the card.

Field Detail

APPLICATION_INSTALLED

`public static final byte` **APPLICATION_INSTALLED**

The current applet context is in the Life Cycle State of INSTALLED (0x03).

Note:

The Life Cycle State INSTALLED could be indicated along with another Application specific Life Cycle State e.g. a value of (0x07) indicates that the applet has been made selectable.

APPLICATION_LOCKED

`public static final byte` **APPLICATION_LOCKED**

The current applet context is in the Life Cycle State of LOCKED (0x7F).

APPLICATION_SELECTABLE

`public static final byte` **APPLICATION_SELECTABLE**

The current applet context is in the Life Cycle State of SELECTABLE (0x07).

Note:

The Life Cycle State SELECTABLE could be indicated along with another Application specific Life Cycle State.

CARD_INITIALIZED

`public static final byte` **CARD_INITIALIZED**

The card is in the Life Cycle State of CARD_INITIALIZED (0x07).

CARD_LOCKED

`public static final byte` **CARD_LOCKED**

The card is in the Life Cycle State of CARD_LOCKED (0x7F).

CARD_OP_READY

`public static final byte` **CARD_OP_READY**

The card is in the Life Cycle State of OP_READY (0x01).

CARD_SECURED

`public static final byte` **CARD_SECURED**

The card is in the Life Cycle State of CARD_SECURED (0x0F).

CARD_TERMINATED

`public static final byte` **CARD_TERMINATED**

The card is in the Life Cycle State of CARD_TERMINATED (0xFF).

CVM_GLOBAL_PIN

`public static final byte` **CVM_GLOBAL_PIN**

Indicates that the `CVM` interface required is a global PIN (0x11).

FAMILY_SECURE_CHANNEL

`public static final byte` **FAMILY_SECURE CHANNEL**

Indicates the family of the Secure Channel Global Service Identifier (0x81).

FAMILY_CVM

`public static final byte` **FAMILY_CVM**

Indicates the family of the CVM Global Service Identifier (0x82).

FAMILY_USSM

`public static final byte` **FAMILY_USSM**

Indicates the family of the USSM Global Service Identifier (0XA0).

GLOBAL_SERVICE_IDENTIFIER

`public static final byte` **GLOBAL_SERVICE_IDENTIFIER**

Indicates the generic Global Service Identifier (0x80).

SECURITY_DOMAIN_PERSONALIZED

`public static final byte` **SECURITY_DOMAIN_PERSONALIZED**

The current Security Domain is in the Life Cycle State of PERSONALIZED (0x0F).

Method Detail

getCardContentState

`public static byte` **getCardContentState**`()`

This method returns the Life Cycle State of the current applet context.

Notes:

- *The OPEN locates the entry of the current applet context in the GlobalPlatform Registry and retrieves the Life Cycle State.*

Returns:

the Life Cycle State of the current applet context.

See Also:

`APPLICATION_INSTALLED, APPLICATION_SELECTABLE, APPLICATION_LOCKED, SECURITY_DOMAIN_PERSONALIZED`

---

getCardState

`public static byte` **getCardState**`()`

This method returns the Life Cycle State of the card.

Returns:

the Life Cycle State of the card

See Also:

`CARD_OP_READY, CARD_INITIALIZED, CARD_SECURED, CARD_LOCKED, CARD_TERMINATED`

getCVM

```
public static org.globalplatform.CVM getCVM(byte bCVMIdentifier)
```

This method returns a handle to the appropriate `CVM` interface of a CVM Application.

Notes

- *OPEN searches the GlobalPlatform Registry for the CVM Application that has the Global Service privilege and is uniquely registered with FAMILY_CVM and this `bCVMIdentifier` or is uniquely registered for the entire `FAMILY_CVM`;*
- *OPEN invokes the `Applet.getShareableInterfaceObject()` method of the corresponding CVM Application . OPEN provides the following parameters to the `Applet.getShareableInterfaceObject()` method: '`clientAID`' is set to the AID of the current applet context (i.e. the requesting on-card entity) and '`parameter`' is set to GLOBAL_SERVICE_IDENTIFIER;*
- *The CVM Application returns the reference of the Shareable Interface Object implementing the GlobalService interface. OPEN invokes the `GlobalService.getServiceInterface()` method with the following parameters: '`GPRegistryEntry`' is set to the GPRegistryEntry interface of the current applet context (i.e. the requesting on-card entity), '`sServiceName`' first byte is set to FAMILY_CVM and second byte to `bCVMIdentifier`, `baBuffer` is set to null (indicating no buffer), `sOffset` and `sLength` are set to zero;*
- *The CVM Application returns the reference of the Shareable Interface Object implementing the CVM interface corresponding to `bCVMIdentifier`. This handle is then returned by OPEN to the requesting applet.*

Parameters:

`bCVMIdentifier` - identifies the required `CVM` interface.

Returns:

the `CVM` interface object reference or

`Null` if there is no matching CVM Interface in the GlobalPlatform Registry.

See Also:

`CVM_GLOBAL_PIN`

getRegistryEntry

```
public static org.globalplatform.GPRegistryEntry
getRegistryEntry(javacard.framework.AID reqAID)
```

This method allows an Application (e.g. a CVM) to access the GP Registry entry of another Application. If no AID is input, this method provides the GP Registry entry of the requesting Application. It returns a handle to the GPRegistryEntry interface.

Notes:

- *The OPEN verifies that the requesting Application has the Global Registry privilege, or is the Security Domain associated with the Application being investigated, or is the investigated Application itself.*

Parameters:

`reqAID` the AID object of the investigated Application, or `null`

Returns:

The `GPRegistryEntry` interface reference corresponding to the input AID, or

`null` if there is no Application in the GP Registry that corresponds to the input AID or if the requesting Application is not authorized to access the corresponding GP Registry entry.

---

getSecureChannel

`public static org.globalplatform.SecureChannel` **getSecureChannel**`()`

This method returns a handle to the `SecureChannel` interface.

Notes:

- *The OPEN locates the entry of the current applet context in the GlobalPlatform Registry to determine the Application's associated Security Domain, regardless of the Security Domain Life Cycle State;*
- *This method may be implemented using a similar mechanism as described for the getCVM method.*

Returns:

the `SecureChannel` interface object reference.

---

getService

```
public static org.globalplatform.GlobalService getService(javacard.framework.AID
                                        serverAID, short sServiceName)
```

This method returns a handle to the Global Services interface of a Global Services Application.

Notes:

- *The `serverAID` parameter is optional: if not known by the requesting on-card entity, it is set to null;*
- *The `sServiceName` parameter is mandatory. When the `serverAID` is Null, the OPEN searches the GlobalPlatform Registry for the Global Services Application corresponding to this unique `sServiceName or` to this entire service family. When the `serverAID` is not Null, the OPEN checks that the `sServiceName` or to this entire service family is recorded in the GP Registry for this `serverAID`;*
- *OPEN invokes the `Applet.getShareableInterfaceObject()` method of the corresponding Global Services Application. OPEN provides the following parameters to the*

`Applet.getShareableInterfaceObject()` method: `clientAID` is set to the AID of the current applet context (i.e. the requesting on-card entity) and `parameter` is set to GLOBAL_SERVICE_IDENTIFIER;

- *The Global Services Application returns the reference of the Shareable Interface Object implementing the `GlobalService` interface. This handle to the `GlobalService` interface is then returned by OPEN to the requesting applet.*

Parameters:

`serverAID` the requested Global Services Application AID or null

`sServiceName` the requested service name

Returns:

the Global Service interface reference or

`null` if there is no matching Global Services Application.

---

lockCard

`public static boolean` **lockCard**`()`

This method locks the card.

Notes:

- *The OPEN locates the entry of the current applet context in the GlobalPlatform Registry and verifies that the Application has the Card Lock privilege.*

Returns:

`true` if card locked, `false` otherwise

---

setATRHistBytes

```
public static boolean setATRHistBytes(byte[] baBuffer,
                                       short sOffset,
                                       byte bLength)
```

For contact cards according to ISO/IEC 7816-4 and Type A contactless cards according to ISO/IEC 14443-3, this method sets the historical bytes. The sequence of bytes will be visible on a subsequent power-up or reset.

Notes:

- *The OPEN locates the entry of the current applet context in the GlobalPlatform Registry and verifies that the Application has the Card Reset privilege for the current card I/O interface;*
- *The OPEN is responsible for synchronizing the length of historical bytes in Format Character T0 of the ATR.*

Parameters:

`baBuffer` - the source byte array containing the historical bytes. Must be a global array.

`sOffset` - offset of the historical bytes within the source byte array.

`bLength` - the number of historical bytes.

Returns:

`true` if historical bytes set, `false` if the Application does not have the required privilege

---

setCardContentState

`public static boolean` **setCardContentState**`(byte bState)`

This method sets the Application specific Life Cycle State of the current applet context. Application specific Life Cycle States range from 0x07 to 0x7F as long as the 3 low order bits are set.

Notes:

- *The OPEN shall reject any transition request to the Life Cycle State INSTALLED;*
- *The OPEN shall reject any transition request from the Life Cycle State LOCKED;*
- *The OPEN locates the entry of the current applet context in the GlobalPlatform Registry and modifies the value of the Life Cycle State.*

Parameters:

`bState` - the application specific state.

Returns:

`true` if the operation is successful, `false` otherwise

See Also:

`SECURITY_DOMAIN_PERSONALIZED, APPLICATION_LOCKED`

---

terminateCard

`public static boolean` **terminateCard**`()`

This method terminates the card.

Notes:

- *The OPEN locates the entry of the current applet context in the GlobalPlatform Registry and verifies that the Application has the Card Terminate privilege.*

Returns:

`true` if card terminated, `false` otherwise

Interface Hierarchy

- interface javacard.framework.Shareable
  - interface org.globalplatform.Application
  - interface org.globalplatform.CVM
  - interface org.globalplatform.GlobalService
  - interface org.globalplatform.GPRegistryEntry
  - interface org.globalplatform.SecureChannel
  - interface org.globalplatform.SecureChannelx

org.globalplatform

Interface **Application**

`public interface` **Application** `extends javacard.framework.Shareable`

This defines the interface that represents an applet method accessible through the OPEN to the Application's associated Security Domain. This interface must be implemented by the Applet class that will use the additional functionality that allows a Security Domain to pass data to the applet.

Method Summary

void	**processData**(byte[] baBuffer, short sOffset, short sLength) This method processes Application specific data received from another entity on the card.

Method Detail

processData

```
public void processData(byte[] baBuffer,
                        short sOffset,
                        short sLength)
```

This method processes Application specific data received from another entity on the card. If this other entity is the Application's associated Security Domain, this data is the APDU buffer.

Notes:

- *During the invocation the Java Card VM performs a context switch;*
- *The applet is responsible for managing the atomicity of its own data;*
- *As this method can be invoked by a Security Domain immaterial of the Application Life Cycle State, it is the responsibility of the applet to ensure that its current Life Cycle State is valid for this operation;*
- *As the applet is not the selected Application, it should not use the `CLEAR_ON_DESELECT` when creating transient arrays.*

Parameters:

`baBuffer` - the source byte array containing the data expected by the applet. This buffer must be global.

`sOffset` - starting offset within source byte array.

`sLength` - length of data.

Throws:

exceptions thrown are Application specific.

---

org.globalplatform

Interface **CVM**

`public interface CVM extends javacard.framework.Shareable`

This defines the interface of a Global Services Application implementing one or more Cardholder Verification Methods. This class offers basic Cardholder Verification Method services (e.g. CVM verification, CVM state interrogation) to any of the Applications present on the card, while some of the services (e.g. unblock CVM, change CVM value) are restricted to Applications with the privilege to change the CVM values. Prior to using this interface, an Application is required to obtain a handle to the CVM services. The CVM application shall expose the GlobalService interface object(s) through applet.getShareableInterfaceObject() according to this specification.

Field Summary

final static short	**CVM_FAILURE** The CVM value comparison failed (-1).
final static short	**CVM_SUCCESS** The CVM value comparison was successful (0).
final static byte	**FORMAT_ASCII** The CVM value is formatted as ASCII bytes (0x01).
final static byte	**FORMAT_BCD** The CVM value is formatted as numerical digits, coded on a nibble (4 bits) and left justified (0x02).
final static byte	**FORMAT_HEX** The CVM value is formatted as hexadecimal (binary) data (0x03).

Method Summary

boolean	**blockState**( ) This method sets the state of the CVM to BLOCKED.
byte	**getTriesRemaining**( ) This method returns the number of tries remaining for the CVM.
boolean	**isActive**( ) This method indicates whether the CVM is present and activated.

boolean	**isBlocked**() This method indicates whether the CVM is currently BLOCKED.
boolean	**isSubmitted**() This method indicates whether an attempt has been made to compare the CVM value.
boolean	**isVerified**() This method indicates whether a successful comparison of the CVM value has occurred (CVM state of VALIDATED).
boolean	**resetAndUnblockState**() This method resets the state of the CVM from BLOCKED to ACTIVE.
boolean	**resetState**() This method resets the state of the CVM to ACTIVE.
boolean	**setTryLimit**(byte bTryLimit) This method sets the maximum number of tries for the CVM.
boolean	**update**(byte[] baBuffer, short sOffset, byte bLength, byte bFormat) This method changes the value of the CVM.
short	**verify**(byte[] baBuffer, short sOffset, byte bLength, byte bFormat) This method compares the CVM value with the value passed as a parameter.

Field Detail

CVM_FAILURE

`public static final short` **CVM_FAILURE**

The CVM value comparison failed (-1).

CVM_SUCCESS

`public static final short` **CVM_SUCCESS**

The CVM value comparison was successful (0).

FORMAT_ASCII

`public static final byte` **FORMAT_ASCII**

The CVM value is formatted as ASCII bytes (0x01).

Note:

- *If the CVM value is stored in a format other than ASCII, it is the responsibility of the interface to convert to the expected format.*

FORMAT_BCD

`public static final byte` **FORMAT_BCD**

The CVM value is formatted as numerical digits, coded on a nibble (4 bits) and left justified (0x02).

Note:

- *If the CVM value is stored in a format other than BCD, it is the responsibility of the interface to convert to the expected format;*
- *If the length of the CVM value is uneven, the right most nibble of the CVM value shall be high values ('F').*

FORMAT_HEX

`public static final byte` **FORMAT_HEX**

The CVM value is formatted as hexadecimal (binary) data (0x03).

Note:

- *If the CVM value is stored in a format other than HEX, it is the responsibility of the interface to convert to the expected format.*

Method Detail

blockState

`public boolean` **blockState**`()`

This method sets the state of the CVM to BLOCKED.

Note:

- *The CVM Application shall verify the CVM Management privilege using the GPRegistryEntry interface of the invoking applet.*

Returns:

`true` if the CVM state was set to BLOCKED, `false` otherwise.

---

getTriesRemaining

`public byte` **getTriesRemaining**`()`

This method returns the number of tries remaining for the CVM. This indicates the number of times the CVM value can be incorrectly presented prior to the CVM reaching the state of BLOCKED.

Returns:

Tries remaining.

---

isActive

`public boolean` **isActive**`()`

This method indicates whether the CVM is present and activated. If active the CVM could be in any one of the following states: ACTIVE, INVALID_SUBMISSION, VALIDATED or BLOCKED).

Returns:

`true` if the CVM state is either ACTIVE, INVALID_SUBMISSION, VALIDATED or BLOCKED, `false` otherwise.

---

isBlocked

`public boolean` **isBlocked**`()`

This method indicates whether the CVM is currently BLOCKED.

Returns:

`true` if the CVM state is BLOCKED, `false` otherwise.

---

isSubmitted

`public boolean` **isSubmitted**`()`

This method indicates whether an attempt has been made to compare the CVM value.

Note:

- *This method does not differentiate whether the CVM value has been successfully verified or not i.e. CVM states of VALIDATED or INVALID_SUBMISSION.*

Returns:

`true` if the CVM state is INVALID_SUBMISSION or VALIDATED, `false` otherwise.

---

isVerified

`public boolean` **isVerified**`()`

This method indicates whether a successful comparison of the CVM value has occurred (CVM state of VALIDATED).

Returns:

`true` if the CVM state is VALIDATED, `false` otherwise.

---

resetAndUnblockState

`public boolean` **resetAndUnblockState**`()`

This method resets the state of the CVM from BLOCKED to ACTIVE.

Notes:

- *The CVM Application shall verify the CVM Management privilege using the GPRegistryEntry interface of the invoking applet;*
- *The CVM Retry Counter is reset when unblocking the CVM.*

Returns:

`true` if the CVM state was reset to ACTIVE, `false` otherwise.

---

resetState

`public boolean` **resetState**`()`

This method resets the state of the CVM to ACTIVE.

Notes:

- *The state of the CVM can only be set to ACTIVE from the states INVALID_SUBMISSION or VALIDATED.*
- *The state of the CVM cannot be set to ACTIVE from the state BLOCKED.*

Returns:

`true` if the CVM state was reset, `false` otherwise.

---

setTryLimit

`public boolean` **setTryLimit**`(byte bTryLimit)`

This method sets the Retry Limit for the CVM.

Notes:

- *The CVM Application shall verify the CVM Management privilege using the GPRegistryEntry interface of the invoking applet;*
- *The CVM Retry Counter is reset when setting the maximum number of tries;*

- *The CVM state is reset to ACTIVE when changing the maximum number of tries. When setting the maximum number of tries before the CVM state is ACTIVE, the CVM state transitions to ACTIVE only if the CVM value is already set.*

Parameters:

`bTryLimit` - the Retry Limit for the CVM.

Returns:

`true` if the Retry Limit was set, `false` otherwise.

---

update

```
public boolean update(byte[] baBuffer,
                      short sOffset,
                      byte bLength,
                      byte bFormat)
```

This method changes the content of the CVM value.

Notes:

- *The CVM Application shall verify the CVM Management privilege using the GPRegistryEntry interface of the invoking applet;*
- *The invoking applet is responsible for specifying the format of the CVM value;*
- *The CVM Retry Counter is reset when changing the CVM value;*
- *The CVM state is reset to ACTIVE when changing the CVM value. When setting the CVM value before the CVM state is ACTIVE, the CVM state transitions to ACTIVE only if the Retry Limit is already set;*
- *Data presented always replaces the previous data regardless of its format or length. The CVM shall remember the format, length, and value of the CVM data. The CVM may (or may not) do editing checks on the data and reject the CVM update if the data fails the editing checks (e.g. reject data that is presented as BCD that is not numerical).*

Parameters:

baBuffer - the source byte array containing the CVM value. This buffer must be global.

sOffset - the offset of the CVM value within source byte array

bLength - the length of the CVM value

bFormat - the format of the CVM value

Returns:

`true` if the CVM value was changed, `false` otherwise.

---

verify

```
public short verify(byte[] baBuffer,
                    short sOffset,
                    byte bLength,
                    byte bFormat)
```

This method compares the stored CVM value with the CVM value passed as a parameter.

Notes:

- *If the value passed as a parameter is not in the same format as the CVM is stored, the value passed as a parameter must be converted prior to comparing;*
- *If HEX format is presented for CVM verification and ASCII or BCD format was used for updating the CVM value, the comparison fails;*
- *If HEX format is presented for CVM verification and HEX format was used for updating the CVM value, the comparison succeeds when the length and the data value match exactly;*
- *If BCD or ASCII format is presented for CVM verification and HEX format was used for updating the CVM value, the comparison fails;*
- *If ASCII format is presented for CVM verification and BCD format was used for updating the CVM value, the comparison fails if the ASCII characters presented for verification are not all numerical (zero to nine). If all the ASCII characters are numerical, format conversion occurs and the comparison succeeds when the length and the data value match exactly;*
- *If BCD format is presented for CVM verification and ASCII format was used for updating the CVM value, the comparison fails if the CVM value contains non-numerical ASCII characters. If the CVM value contains only numerical ASCII characters, format conversion occurs and the comparison succeeds when the length and the data value match exactly;*
- *If the comparison is successful, the Retry Counter must be reset and the CVM state must be set to VALIDATED;*
- *If the comparison is unsuccessful, the Retry Counter must be updated and the CVM state must be set to INVALID_SUBMISSION;*
- *The Retry Counter object and the CVM states VALIDATED and INVALID_SUBMISSION shall not conform to a transaction in progress, i.e. they shall not revert to a previous value if a transaction in progress is aborted;*
- *If the maximum number of tries has been reached, the CVM state must be set to BLOCKED.*

Parameters:

`baBuffer` - the source byte array containing the CVM. This buffer must be global.

`sOffset` - the offset of the CVM value within source byte array

`bLength` - the length of the CVM value

`bFormat` - the format of the CVM value

Returns:

value indicating whether the comparison was successful or not. Values other than `CVM_SUCCESS` (0) or `CVM_FAILURE` (-1) are Reserved for Future Use.

---

org.globalplatform

Interface **GlobalService**

`public interface` **GlobalService** `extends javacard.framework.Shareable`

This defines the interface for requesting a Global Services Application to provide its actual service interface. The Global Services Application uses this interface to check the validity of the request presented by an on-card entity.

Field Summary

final static byte	KEY_ACCESS_ANY Key access indicating key may be used by the Security Domain and any associated application (0x00).
final static byte	KEY_ACCESS_SECURITY_DOMAIN Key access indicating key may be used by the Security Domain but not by any associated application (0x01).
final static byte	KEY_ACCESS_APPLICATION Key access indicating key may be used by any associated application but not by the Security Domain (0x02).
final static byte	KEY_TYPE_AES Key type indicating AES (0x88).
final static byte	KEY_TYPE_3DES Key type indicating Triple DES reserved for specific implementations (0x81).
final static byte	KEY_TYPE_3DES_CBC Key type indicating Triple DES in CBC mode (0x82).
final static byte	KEY_TYPE_DES Key type indicating DES with ECB/CBC implicitly known (0x80).
final static byte	KEY_TYPE_DES_CBC Key type indicating DES in CBC mode (0x84).
final static byte	KEY_TYPE_DES_ECB Key type indicating DES in ECB mode (0x83).
final static byte	KEY_TYPE_EXTENDED Key type indicating extended key format (0xFF).

final static byte	KEY_TYPE_HMAC_SHA1 Key type indicating HMAC SHA1, length of HMAC implicitly known (0x90).
final static byte	KEY_TYPE_HMAC_SHA1_160 Key type indicating HMAC SHA1, length of HMAC is 160 bits (0x91).
final static byte	KEY_TYPE_RSA_PRIVATE_CRT_P Key type indicating RSA Private Key Chinese Remainder p component (0xA4).
final static byte	KEY_TYPE_RSA_PRIVATE_CRT_PQ Key type indicating RSA Private Key Chinese Remainder pq component (0xA6).
final static byte	KEY_TYPE_RSA_PRIVATE_CRT_Q Key type indicating RSA Private Key Chinese Remainder q component (0xA5).
final static byte	KEY_TYPE_RSA_PRIVATE_CRT_DP1 Key type indicating RSA Private Key Chinese Remainder dp1 component (0xA7).
final static byte	KEY_TYPE_RSA_PRIVATE_CRT_DQ1 Key type indicating RSA Private Key Chinese Remainder dq1 component (0xA8).
final static byte	KEY_TYPE_RSA_PRIVATE_EXPONENT Key type indicating RSA Private exponent (0xA3).
final static byte	KEY_TYPE_RSA_PRIVATE_MODULUS Key type indicating RSA Private Key modulus (0xA2).
final static byte	KEY_TYPE_RSA_PUBLIC_EXPONENT Key type indicating RSA Public Key exponent (0xA0).
final static byte	KEY_TYPE_RSA_PUBLIC_MODULUS Key type indicating RSA Public Key modulus (0xA1).
final static byte	KEY_USAGE_COMPUTATION_DECIPHERMENT Key usage indicating computation and decipherment (0x40).
final static byte	KEY_USAGE_CONFIDENTIALITY Key usage indicating sensitive data confidentiality (0x08).
final static byte	KEY_USAGE_CRYPTOGRAPHIC_AUTHORIZATION Key usage indicating cryptographic authorization (0x01).
final static byte	KEY_USAGE_CRYPTOGRAPHIC_CHECKSUM Key usage indicating cryptographic checksum e.g. MAC (0x04).

final static byte	KEY_USAGE_DIGITAL_SIGNATURE Key usage indicating Digital Signature (0x02).
final static byte	KEY_USAGE_SM_COMMAND Key usage indicating Secure Messaging in command data field (0x10).
final static byte	KEY_USAGE_SM_RESPONSE Key usage indicating Secure Messaging in response data field (0x20).
final static byte	KEY_USAGE_VERIFICATION_ENCIPHERMENT Key usage indicating verification and encipherment (0x80).

Method Summary

javacard.framework.shareable	**getServiceInterface**(GPRegistryEntry clientRegistryEntry, short sServiceName, byte[ ] baBuffer, short sOffset, short sLength) This method returns a handle to the requested service interface.

Field Detail

KEY_ACCESS_ANY

`public static final byte` **KEY_ACCESS_ANY**

Key access indicating key may be used by the Security Domain and any associated application (0x00).

KEY_ACCESS_SECURITY_DOMAIN

`public static final byte` **KEY_ACCESS_SECURITY_DOMAIN**

Key access indicating key may be used by the Security Domain but not by any associated application (0x01).

KEY_ACCESS_APPLICATION

`public static final byte` **KEY_ACCESS_APPLICATION**

Key access indicating key may be used by any associated application but not by the Security Domain (0x02).

KEY_TYPE_AES

`public static final byte` **KEY_TYPE_AES**

Key type indicating AES (0x88).

KEY_TYPE_3DES

`public static final byte` **KEY_TYPE_3DES**

Key type indicating Triple DES reserved for specific implementations (0x81).

KEY_TYPE_3DES_CBC

`public static final byte` **KEY_TYPE_3DES_CBC**

Key type indicating Triple DES in CBC mode (0x82).

KEY_TYPE_DES

`public static final byte` **KEY_TYPE_DES**

Key type indicating DES with ECB/CBC implicitly known (0x80).

KEY_TYPE_DES_CBC

`public static final byte` **KEY_TYPE_DES_CBC**

Key type indicating DES in CBC mode (0x84).

KEY_TYPE_DES_ECB

`public static final byte` **KEY_TYPE_DES_ECB**

Key type indicating DES in ECB mode (0x83).

KEY_TYPE_EXTENDED

`public static final byte` **KEY_TYPE_EXTENDED**

Key type indicating extended key format (0xFF).

KEY_TYPE_HMAC_SHA1

`public static final byte` **KEY_TYPE_HMAC_SHA1**

Key type indicating HMAC SHA1, length of HMAC implicitly known (0x90).

KEY_TYPE_HMAC_SHA1_160

`public static final byte` **KEY_TYPE_HMAC_SHA1_160**

Key type indicating HMAC SHA1, length of HMAC is 160 bits (0x91).

KEY_TYPE_RSA_PRIVATE_CRT_P

`public static final byte` **KEY_TYPE_RSA_PRIVATE_CRT_P**

Key type indicating RSA Private Key Chinese Remainder p component (0xA4).

KEY_TYPE_RSA_PRIVATE_CRT_PQ

`public static final byte` **KEY_TYPE_RSA_PRIVATE_CRT_PQ**

Key type indicating RSA Private Key Chinese Remainder pq component (0xA6).

KEY_TYPE_RSA_PRIVATE_CRT_Q

`public static final byte` **KEY_TYPE_RSA_PRIVATE_CRT_Q**

Key type indicating RSA Private Key Chinese Remainder q component (0xA5).

KEY_TYPE_RSA_PRIVATE_CRT_DP1

`public static final byte` **KEY_TYPE_RSA_PRIVATE_CRT_DP1**

Key type indicating RSA Private Key Chinese Remainder dp1 component (0xA7).

KEY_TYPE_RSA_PRIVATE_CRT_DQ1

`public static final byte` **KEY_TYPE_RSA_PRIVATE_CRT_DQ1**

Key type indicating RSA Private Key Chinese Remainder dq1 component (0xA8).

KEY_TYPE_RSA_PRIVATE_EXPONENT

`public static final byte` **KEY_TYPE_RSA_PRIVATE_EXPONENT**

Key type indicating RSA Private exponent (0xA3).

KEY_TYPE_RSA_PRIVATE_MODULUS

`public static final byte` **KEY_TYPE_RSA_PRIVATE_MODULUS**

Key type indicating RSA Private Key modulus (0xA2).

KEY_TYPE_RSA_PUBLIC_EXPONENT

`public static final byte` **KEY_TYPE_RSA_PUBLIC_EXPONENT**

Key type indicating RSA Public Key exponent (0xA0).

KEY_TYPE_RSA_PUBLIC_MODULUS

`public static final byte` **KEY_TYPE_RSA_PUBLIC_MODULUS**

Key type indicating RSA Public Key modulus (0xA1).

KEY_USAGE_COMPUTATION_DECIPHERMENT

`public static final byte` **KEY_USAGE_COMPUTATION_DECIPHERMENT**

Key usage indicating computation and decipherment (0x40).

KEY_USAGE_CONFIDENTIALITY

`public static final byte` **KEY_USAGE_CONFIDENTIALITY**

Key usage indicating sensitive data confidentiality (0x08).

KEY_USAGE_CRYPTOGRAPHIC_AUTHORIZATION

`public static final byte` **KEY_USAGE_CRYPTOGRAPHIC_AUTHORIZATION**

Key usage indicating cryptographic authorization (0x01).

KEY_USAGE_CRYPTOGRAPHIC_CHECKSUM

`public static final byte` **KEY_USAGE_CRYPTOGRAPHIC_CHECKSUM**

Key usage indicating cryptographic checksum e.g. MAC (0x04).

KEY_USAGE_DIGITAL_SIGNATURE

`public static final byte` **KEY_USAGE_DIGITAL_SIGNATURE**

Key usage indicating Digital Signature (0x02).

KEY_USAGE_SM_COMMAND

`public static final byte` **KEY_USAGE_SM_COMMAND**

Key usage indicating Secure Messaging in command data field (0x10).

KEY_USAGE_SM_RESPONSE

`public static final byte` **KEY_USAGE_SM_RESPONSE**

Key usage indicating Secure Messaging in response data field (0x20).

KEY_USAGE_VERIFICATION_ENCIPHERMENT

`public static final byte` **KEY_USAGE_VERIFICATION_ENCIPHERMENT**

Key usage indicating verification and encipherment (0x80).

Method Detail

getServiceInterface

`public javacard.framework.shareable` **getServiceInterface**`(GPRegistryEntry clientRegistryEntry, short sServiceName, byte[ ] baBuffer, short sOffset, short sLength)`

This method returns a handle to the requested service interface of a Global Services Application.

Notes:

- *The Global Services Application verifies the validity of the request according to its own security policies for this `sServiceName` and based on the identity of the requesting on-card entity and its characteristics as registered in the `clientRegistryEntry`;*
- *On a valid service request, the Global Service Application returns the reference of the Shareable Interface Object implementing the actual service: this Shareable Interface Object may either be this Global Service interface or another Java Card shareable interface;*
- *Depending on its own security policy, the Global Service Application may reject an invalid service request by either throwing an exception or just returning `Null`.*

Parameters:

`clientRegistryEntry`	the GP Registry entry reference of the requesting on-card entity
`sServiceName`	the requested service name
`baBuffer`	the source byte array containing additional parameters of the service request
`sOffset`	offset of the additional request parameters within the source byte array
`sLength`	length of the additional request parameters

Returns:

the specific service interface reference or null

Throws:

`javacard.framework.ISOException` - with the following reason codes:

- `ISO7816.SW_CONDITIONS_NOT_SATISFIED`
- `ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED`

org.GlobalPlatform

Interface **GPRegistryEntry**

`public interface` **GPRegistryEntry** `extends javacard.framework.Shareable`

This defines the interface corresponding to the GPRegistryEntry of a single Application.

Field Summary

final static byte	PRIVILEGE_AUTHORIZED_MANAGEMENT Privilege number for Authorized Management (0x09).
final static byte	PRIVILEGE_CARD_LOCK Privilege number for Card Lock (0x03).
final static byte	PRIVILEGE_CARD_RESET Privilege number for Card Reset (0x05).
final static byte	PRIVILEGE_CARD_TERMINATE Privilege number for Card Terminate (0x04).
final static byte	PRIVILEGE_CVM_MANAGEMENT Privilege number for CVM Management (0x06).
final static byte	PRIVILEGE_DAP_VERIFICATION Privilege number for DAP verification (0x01).
final static byte	PRIVILEGE_DELEGATED_MANAGEMENT Privilege number for Delegated Management (0x02).
final static byte	PRIVILEGE_FINAL_APPLICATION Privilege number for Final Application (0x0E).
final static byte	PRIVILEGE_GLOBAL_DELETE Privilege number for Global Delete (0x0B).
final static byte	PRIVILEGE_GLOBAL_LOCK Privilege number for Global Lock (0x0C).
final static byte	PRIVILEGE_GLOBAL_REGISTRY Privilege number for Global Registry (0x0D).

final static byte	PRIVILEGE_GLOBAL_SERVICE Privilege number for Global Service (0x0F).
final static byte	PRIVILEGE_MANDATED_DAP Privilege number for Mandated DAP verification (0x07).
final static byte	PRIVILEGE_RECEIPT_GENERATION Privilege number for Receipt Generation (0x10).
final static byte	PRIVILEGE_SECURITY_DOMAIN Privilege number for application being a Security Domain (0x00).
final static byte	PRIVILEGE_TOKEN_VERIFICATION Privilege number for Token Verification (0x0A).
final static byte	PRIVILEGE_TRUSTED_PATH Privilege number for Trusted Path (0x08).

Method Summary

void	**deregisterService**(short sServiceName) This method deregisters a service name.
javacard.framework.AID	**getAID**() This method provides the AID of the GlobalPlatform Registry entry.
short	**getPrivileges**(byte[] baBuffer, short sOffset) This method retrieves the Privileges for the GlobalPlatform Registry entry.
byte	**getState**() This method retrieves the Life Cycle State for the GlobalPlatform Registry entry.
boolean	**isAssociated**(javacard.framework.AID SDAID) This method checks the association between an Application and a Security Domain.
boolean	**isPrivileged**(byte bPrivilege) This method checks the privilege of an Application.
void	**registerService**(short sServiceName) This method registers a unique service name.

boolean	**setState** `(byte bState)` This method sets the Life Cycle state of an Application.

Field Detail

PRIVILEGE_AUTHORIZED_MANAGEMENT

`public static final byte` **PRIVILEGE_AUTHORIZED_MANAGEMENT**

Privilege number for Authorized Management (0x00).

PRIVILEGE_CARD_LOCK

`public static final byte` **PRIVILEGE_CARD_LOCK**

Privilege number for Card Lock (0x03).

PRIVILEGE_CARD_RESET

`public static final byte` **PRIVILEGE_CARD_RESET**

Privilege number for Card Reset (0x05).

PRIVILEGE_CARD_TERMINATE

`public static final byte` **PRIVILEGE_CARD_TERMINATE**

Privilege number for Card Terminate (0x04).

PRIVILEGE_CVM_MANAGEMENT

`public static final byte` **PRIVILEGE_CVM_MANAGEMENT**

Privilege number for CVM Management (0x06).

PRIVILEGE_DAP_VERIFICATION

`public static final byte` **PRIVILEGE_DAP_VERIFICATION**

Privilege number for DAP verification (0x01).

PRIVILEGE_DELEGATED_MANAGEMENT

`public static final byte` **PRIVILEGE_DELEGATED_MANAGEMENT**

Privilege number for Delegated Management (0x06).

PRIVILEGE_FINAL_APPLICATION

`public static final byte` **PRIVILEGE_FINAL_APPLICATION**

Privilege number for Final Application (0x0E).

PRIVILEGE_GLOBAL_DELETE

`public static final byte` **PRIVILEGE_GLOBAL_DELETE**

Privilege number for Global Delete (0x0B).

PRIVILEGE_GLOBAL_LOCK

`public static final byte` **PRIVILEGE_GLOBAL_LOCK**

Privilege number for Global Lock (0x0C).

PRIVILEGE_GLOBAL_REGISTRY

`public static final byte` **PRIVILEGE_GLOBAL_REGISTRY**

Privilege number for Global Registry (0x0D).

PRIVILEGE_GLOBAL_SERVICE

`public static final byte` **PRIVILEGE_GLOBAL_SERVICE**

Privilege number for Global Service (0x0F).

PRIVILEGE_MANDATED_DAP

`public static final byte` **PRIVILEGE_MANDATED_DAP**

Privilege number for Mandated DAP verification (0x07).

PRIVILEGE_RECEIPT_GENERATION

`public static final byte` **PRIVILEGE_RECEIPT_GENERATION**

Privilege number for Receipt Generation (0x07).

PRIVILEGE_SECURITY_DOMAIN

`public static final byte` **PRIVILEGE_SECURITY_DOMAIN**

Privilege number for application being a Security Domain (0x00).

PRIVILEGE_TOKEN_VERIFICATION

`public static final byte` **PRIVILEGE_TOKEN_VERIFICATION**

Privilege number for Token Verification (0x0A).

PRIVILEGE_TRUSTED_PATH

`public static final byte` **PRIVILEGE_TRUSTED_PATH**

Privilege number for Trusted Path (0x08).

Method Detail

deregisterService

`public void` **deregisterService**`(short sServiceName)`

This method allows a Global Services Application (e.g. a CVM Application) to deregister a service name.

Notes:

- *The OPEN checks that the requesting on-card entity has the Global Service Privilege and is associated with this registry entry;*
- *The OPEN checks that the service name is registered as unique for the requesting on-card entity.*

Parameters:

`serviceName` the unique service name to deregister

Throws:

`javacard.framework.ISOException` - with the following reason codes:

- `ISO7816.SW_CONDITIONS_NOT_SATISFIED`

---

getAID

`public javacard.framework.AID` **getAID**`()`

This method returns the Application's AID registered in the current GlobalPlatform Registry's entry.

Returns:

the AID object

---

getPrivileges

`public short` **getPrivileges** (`byte[] baBuffer, short sOffset`)

This method returns all the Privileges bytes registered in the current GlobalPlatform registry entry.

Parameters:

`baBuffer` The byte array where Privileges bytes are to be stored

`sOffset` The offset in baBuffer at which to begin the Privileges bytes

Returns:

sOffset + Length of the Privileges.

Throws:

`java.lang.ArrayIndexOutOfBoundsException` - if storing the Privileges bytes would cause access outside array bounds or the sOffset is negative.

---

getState

`public byte` **getState** ()

This method returns the Life Cycle State registered in the current GlobalPlatform Registry entry.

Returns:

The Life Cycle State as defined in section 11.1.1.

---

isAssociated

`public boolean` **isAssociated**(`javacard.framework.AID SDAID`)

This method allows to verify if the entity whose AID is provided in the input parameters is registered as the associated Security Domain of this GPRegistryEntry.

Notes:

- *The OPEN determines if the SDAID is registered in the current GlobalPlatform Registry's entry as the associated Security Domain.*

Parameters:

`SDAID` the AID object of the investigated Security Domain

Returns:

`True` if the GP Registry references the Security Domain as being associated with this GPRegistryEntry, or

`False` otherwise.

---

isPrivileged

`public boolean` **isPrivileged**`(byte bPrivilege)`

This method allows an Application (e.g. a CVM Application) to verify if a given Privilege is registered in this GPRegistryEntry (e.g. check the CVM Management privilege of another Application invoking the CVM.update() method).

Parameters:

`bPrivilege`	the privilege number to verify, as defined in Table 6-1.

Returns:

`True` if at least the referenced Privilege is registered in the GP Registry entry, or

`False` if the referenced Privilege is not registered in the GP Registry entry.

---

registerService

`public void` **registerService**`(short sServiceName)`

This method allows a Global Services Application (e.g. a CVM Application) to register a unique service name.

Notes:

- *The OPEN checks that the requesting on-card entity has the Global Service Privilege and is associated with the current GlobalPlatform Registry entry;*
- *The OPEN checks that the requested service name matches with (one of) the Service Parameter(s) recorded in the current GlobalPlatform Registry entry;*
- *The OPEN checks that the service name is not already registered as unique by any other entry in the GlobalPlatform Registry.*

Parameters:

`sServiceId`	the unique service identifier to register

Throws:

`javacard.framework.ISOException` - with the following reason codes:

- `ISO7816.SW_CONDITIONS_NOT_SATISFIED`

---

setState

```
public boolean setState(byte bState)
```

This method allows the Life Cycle state of this GPRegistryEntry to be transitioned to the requested target state.

Notes:

- *A transition request to the Life Cycle State INSTALLED shall be rejected;*
- *A transition request to Life Cycle state other than APPLICATION_LOCKED and APPLICATION_UNLOCKED shall be accepted only if the invoking Application corresponds to this GPRegistryEntry;*
- *An Application shall be able to lock and shall not be able to unlock itself;*
- *Only an Application with GlobalLock privilege or the directly or indirectly associated Security Domain of this GPRegistryEntry shall be able to lock or unlock this GPRegistry Entry;*
- *This method shall fail if this GPRegistryEntry corresponds to the Issuer Security Domain.*

Parameters:

`bState` the target state for this GPRegistryEntry

Returns:

`True` if the transition is successful, or

`False` otherwise.

---

org.globalplatform

Interface **SecureChannel**

`public interface` **SecureChannel** `extends javacard.framework.Shareable`

This defines an interface to be used by an Application that may want to delegate the handling of entity authentication and APDU security to its associated Security Domain. This interface is designed to offer interoperability to the Application in that it requires no knowledge of the mechanisms used to perform the authentication or of the scheme used for APDU security and shall allow an Application to interface correctly with a Security Domain immaterial of the mechanisms or schemes used. Prior to using this interface, an Application is required to obtain a handle to its associated Security Domain's `SecureChannel` interface object by invoking the GP`System.getSecureChannel()` method. The `SecureChannel` interface shall only be exposed through the GP`System.getSecureChannel()` method.

In all cases where the Application passes the APDU buffer as a parameter to the Security Domain, the class byte of the APDU shall not be modified. This ensures that the Security Domain knows the logical channel number. If the card supports logical channels, it is the responsibility of the Security Domain to correctly manage the logical channel information when processing the APDU.

See also SecureChannelx interface.

Field Summary

final static byte	**ANY_AUTHENTICATED** The off-card entity has been authenticated but not as the provider of this Application (0x40)
final static byte	**AUTHENTICATED** The off-card entity has been authenticated as the provider of this Application (0x80).
final static byte	**C_DECRYPTION** The unwrap method will decrypt incoming command data (0x02).
final static byte	**C_MAC** The unwrap method will verify the MAC on an incoming command (0x01).
final static byte	**NO_SECURITY_LEVEL** Entity Authentication has not occurred (0x00).
final static byte	**R_ENCRYPTION** The wrap method will encrypt the outgoing response data (0x20).
final static byte	**R_MAC** The wrap method will generate a MAC for outgoing response data (0x10).

Method Summary

short	**decryptData**(byte[] baBuffer, short sOffset, short sLength) This method is used to decrypt data located in the input buffer.
short	**encryptData**(byte[] baBuffer, short sOffset, short sLength) This method is used to encrypt data located in the input buffer.
byte	**getSecurityLevel**() This method is used to determine whether the Security Domain has performed authentication and to determine what level of security will be applied by the `wrap()` and `unwrap()` methods.
short	**processSecurity**(javacard.framework.APDU apdu) Processes security related APDU commands.
void	**resetSecurity**() This method is used to reset information relating to the current Secure Channel Session.
short	**unwrap**(byte[] baBuffer, short sOffset, short sLength) This method is used to process and verify the secure messaging of an incoming command according to the security level.
short	**wrap**(byte[] baBuffer, short sOffset, short sLength) This method is used to apply additional security processing to outgoing response data and status bytes according to the security level.

Field Detail

ANY_AUTHENTICATED

`public static final byte` **ANY_AUTHENTICATED**

The off-card entity has been authenticated but not as the provider of this Application (0x40).

Note:

- *Entity authentication and the level of security that will be applied by the wrap and unwrap methods are not necessarily related. A Security Domain, by default, could verify the MAC on any command passed as a parameter in the `unwrap()` method without entity authentication previously having occurred*

AUTHENTICATED

`public static final byte` **AUTHENTICATED**

The off-card entity has been authenticated as the provider of this Application (0x80).

Note:

- *Entity authentication and the level of security that will be applied by the wrap and unwrap methods are not necessarily related. A Security Domain, by default, could verify the MAC on any command passed as a parameter in the `unwrap()` method without entity authentication previously having occurred*

C_DECRYPTION

`public static final byte` **C_DECRYPTION**

The `unwrap` method will decrypt incoming command data (0x02).

Note:

- *Command data decryption could be indicated along with entity authentication and one or more levels of security.*

C_MAC

`public static final byte` **C_MAC**

The `unwrap` method will verify the MAC on an incoming command (0x01).

Note:

- *MAC verification could be indicated along with entity authentication and one or more levels of security e.g. a value of '03' indicates that while entity authentication has not occurred, the `unwrap()` method will decrypt the command data of incoming commands and verify the MAC on incoming commands.*

NO_SECURITY_LEVEL

`public static final byte` **NO_SECURITY_LEVEL**

Entity authentication has not occurred (0x00).

Notes:

- *This indicates that no Secure Channel Session is active, or that a security error has occurred during the current Secure Channel Session, or that the previous Secure Channel Session was aborted during the same Application Session;*
- *Entity authentication and the level of security that will be applied by the `wrap` and `unwrap` methods are not necessarily related. A Security Domain, by default, could verify the MAC on any command passed as a parameter in the `unwrap()` method without entity authentication previously having occurred;*

- *The `wrap` and `unwrap` methods will not apply any cryptographic processing to command or response data.*

R_ENCRYPTION

```
public static final byte R_ENCRYPTION
```

The `wrap` method will encrypt the outgoing response data (0x20).

Note:

- *Response data encryption could be indicated along with entity authentication and one or more levels of security.*

R_MAC

```
public static final byte R_MAC
```

The `wrap` method will generate a MAC for the outgoing response data (0x10).

Note:

- *MAC generation could be indicated along with entity authentication and one or more levels of security e.g. a value of '91' indicates that entity authentication has occurred and that the `unwrap()` method will verify the MAC on incoming commands and that the `wrap()` method will generate a MAC on outgoing response data*

Method Detail

decryptData

```
public short decryptData(byte[] baBuffer,
                         short sOffset,
                         short sLength)
                         throws javacard.framework.ISOException
```

This method is used to decrypt data located in the input buffer.

Notes:

- *Processing this method shall comply to the rules of the Secure Channel Protocol implemented by the Security Domain;*
- *The Security Domain implicitly knows the key used for decryption;*
- *The Security Domain is implicitly aware of any padding that may be present in the decrypted data according to the Secure Channel Protocol and the eventual padding is discarded;*
- *The clear text data replaces the ciphered data within the byte array;*
- *The applet is responsible for checking the integrity of the decrypted data;*

- *A Secure Channel Session shall be opened and a sensitive data decryption key shall be available;*
- *This method fails if a Secure Channel Session is not open (i.e. Session Security Level = NO_SECURITY_LEVEL) or if the corresponding session keys are not available.*

Parameters:

`baBuffer` - the source byte array. This buffer must be global.

`sOffset` - offset within the source byte array to start the decryption.

`sLength` - the number of bytes to decrypt.

Returns:

The length of the clear text data.

Throws:

`javacard.framework.ISOException` - with the following reason codes:

- `ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED` if a Secure Channel Session has not been opened.
- `ISO7816.SW_WRONG_LENGTH` if the length of data to be decrypted is not valid.

---

encryptData

```
public short encryptData(byte[] baBuffer,
                         short sOffset,
                         short sLength)
                  throws java.lang.ArrayIndexOutOfBoundsException
                         javacard.framework.ISOException
```

This method is used to encrypt data located in the input buffer.

Notes:

- *Processing this method shall comply to the rules of the Secure Channel Protocol implemented by the Security Domain;*
- *The Security Domain is implicitly aware of any padding that must be applied to the clear text data prior to encryption according to the Secure Channel Protocol;*
- *The Security Domain implicitly knows the key used for encryption;*
- *The ciphered data replaces the clear text data within the byte array;*
- *A Secure Channel Session shall be opened and a sensitive data encryption key shall be available;*
- *This method fails if a Secure Channel Session is not open (i.e. Session Security Level = NO_SECURITY_LEVEL) or if the corresponding session keys are not available.*

Parameters:

`baBuffer` - the source byte array. This buffer must be global.

`sOffset` - offset within the source byte array to start the encryption.

`sLength` - the number of bytes to encrypt.

Returns:

The length of the encrypted data.

Throws:

- `java.lang.ArrayIndexOutOfBoundsException` - if enciphering would cause access outside array bounds.
- ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED if a Secure Channel Session is not open.

---

getSecurityLevel

```
public byte getSecurityLevel()
```

This method returns, from the requester's standpoint, the Current Security Level coded as a bit-map according to Table 10-1 indicating whether entity authentication has occurred and what level of security is currently applicable to command and response messages processed by the `unwrap()` and `wrap()` methods.

Notes:

- *Applets must invoke this method to ensure that application specific security requirements have been previously met or will be enforced by the Security Domain;*
- *More than one level of security may be active and these may change during a Secure Channel Session e.g. an `R_MAC` session may be initiated during a `C_MAC` session;*
- *The Current Security Level may be different at the same time for an Application invoking the method and its associated Security Domain depending on the Secure Channel Protocol and the authenticated off-card entity's Application Provider ID (e.g. it may be ANY_AUTHENTICATED for the application and AUTHENTICATED for its associated Security Domain).*

Returns:

- The Current Security Level.

---

processSecurity

```
public short processSecurity(javacard.framework.APDU apdu)
                      throws javacard.framework.ISOException
```

Processes security related APDU commands.

This method is used by an applet to process APDU commands that possibly relate to the security mechanism used by the Security Domain. As the intention is to allow an Application to be associated with a Security Domain without having any knowledge of the security mechanisms used by the Security Domain, the applet assumes that APDU commands that it does not recognize are part of the security mechanism and will be recognized by the Security Domain. The applet can either invoke this method prior to determining if it recognizes the command or only invoke this method for commands it does not recognize.

The method sets the compulsory Session Security Level that is established at Secure Channel initiation and which is required for the whole Secure Channel Session. On successful initialization of the Secure Channel Session, the Current Security Level is set to the same value as the compulsory Session Security Level. The Current Security

Level is updated (R-MAC or not) on the successful processing of the BEGIN R-MAC SESSION / END R-MAC SESSION commands.

Notes:

- *Processing this method shall comply to the rules of the Secure Channel Protocol implemented by the Security Domain;*
- *The method is responsible for receiving the data field of commands that are recognized;*
- *The applet is responsible for recognizing commands that the method refused to process ('6E00' and '6D00');*
- *The applet is responsible for outputting status bytes returned due to the processing of instructions recognized by the method;*
- *If response data is present, this data will be placed in the APDU buffer at offset `ISO7816.OFFSET_CDATA`. The return code indicates the length and the applet is responsible for outputting this data.*

Parameters:

`apdu` - the incoming APDU object

Returns:

the number of bytes to be output

Throws:

`javacard.framework.ISOException` - with the following reason codes (other security mechanism related status bytes may be returned):

- `ISO7816.SW_CLA_NOT_SUPPORTED` if class byte is not recognized by the method;
- `ISO7816.SW_INS_NOT_SUPPORTED` if instruction byte is not recognized by the method.

---

resetSecurity

`public void` **resetSecurity**()

This method is used to reset all information relating to the current Secure Channel Session. It resets both the compulsory Session Security Level and the Current Security Level to NO_SECURITY_LEVEL, terminates the current Secure Channel Session and erases all session keys.

Notes:

- *It is strongly recommended that applets using the services of a Security Domain invoke this method in the `Applet.deselect()` method;*
- *The Security Domain will reset all information relating to the current Secure Channel Session i.e. all Secure Channel session keys, state information and security level information will be erased;*
- *This method shall not fail if no Secure Channel Session is open (i.e. Session Security Level = NO_SECURITY_LEVEL).*

---

unwrap

```
public short unwrap(byte[] baBuffer,
                    short sOffset,
                    short sLength)
             throws javacard.framework.ISOException
```

This method is used to process and verify the secure messaging of an incoming command according to the Current Security Level and Session Security Level of the current Secure Channel Session.

Notes:

- *Processing this method shall comply to the rules of the Secure Channel Protocol implemented by the Security Domain;*
- *If `NO_SECURITY_LEVEL`, `AUTHENTICATED` or `ANY_AUTHENTICATED` only is indicated, when complying to the Secure Channel Protocol rules, this method will not attempt any secure messaging processing on the incoming command, the incoming command will remain as is within the incoming APDU object and the returned length of the "unwrapped" data is set to the value of the sLength parameter, otherwise a security error is returned;*
- *If the class byte does not indicate secure messaging (according to ISO/IEC 7816-4), this method will not attempt any secure messaging processing on the incoming command and the incoming command will remain as is within the incoming APDU object. When complying with the Secure Channel Protocol rules, the returned length of the "unwrapped" data is set to the value of the sLength parameter otherwise a security error is returned;*
- *The applet is responsible for receiving the data field of the command;*
- *Correct secure messaging processing of the `unwrap()` will result in the incoming command being reformatted within the incoming APDU object with all data relating to the secure messaging removed. A reformatted case 1 or case 2 command will include an Lc byte set to zero;*
- *If R_MAC is indicated in the Current Security Level of the Secure Channel Session, once secure messaging processing of the incoming command has successfully completed, R-MAC computation on the reformatted command (i.e. after all the data relating to secure messaging has been removed) will be initiated. If no secure messaging processing was required for the incoming command, R-MAC computation will be initiated on the unmodified incoming command, appended with a Lc byte of zero in event of a case 1 or case 2 command;*
- *Incorrect processing of the `unwrap()` will result in the information relating to the current Secure Channel being reset;*
- *This method does not fail if a Secure Channel Session is not open (that is, when Session Security Level = NO_SECURITY_LEVEL) or if the corresponding session keys are not available;*
- *The method fails if the secure messaging of the incoming command is not successfully verified or does not match the Current Security Level (as defined either as the compulsory Session Security Level set at initialization of the Secure Channel Session or as the Current Security Level set by the `setSecurityLevel()` method). If the method fails, the Current Security Level is reset to NO_SECURITY_LEVEL, but not the compulsory Session Security Level.*

Parameters:

`baBuffer` - the source of the data to be wrapped. This buffer must be global

`sOffset` - the offset within the source buffer of the data to unwrap

`sLength` - the length of the data to unwrap

Returns:

the length of the unwrapped data i.e. the length of the command header and data field

Throws:

`javacard.framework.ISOException` - with the following reason code (other security mechanism related status bytes may be returned):

- *ISO7816.SW_SECURITY_STATUS_NOT_SATISFIED if secure messaging verification failed.*
- *ISO7816.SW_CLA_NOT_SUPPORTED if class byte is not recognized by the method.*

---

wrap

```
public short wrap(byte[] baBuffer,
                  short sOffset,
                  short sLength)
           throws java.lang.ArrayIndexOutOfBoundsException,
                  javacard.framework.ISOException.
```

This method applies secure messaging to the current outgoing response according to the Current Security Level and Session Security Level of the Secure Channel Session.

Notes:

- *Processing this method shall comply to the rules of the Secure Channel Protocol implemented by the Security Domain;*
- *This method attempts secure messaging processing of the current outgoing response when the Current Security Level indicates R_MAC and/or R_ENCRYPTION;*
- *If the Current Security Level does not indicate `R_MAC` and/or `R_ENCRYPTION`, when complying with the Secure Channel Protocol rules, this method will do no processing and the outgoing response message will remain as is in the APDU object. The returned length of the "wrapped" data is set to the value of the `sLength` parameter minus 2 (indicating the status bytes are no longer present at the end of the returned data);*
- *The applet is responsible for appending the expected status bytes at the end of the response data in order for them to be protected by secure messaging;*
- *The returned data does not include the status bytes;*
- *The method fails if the secure messaging of the incoming command is not successfully verified or does not match the Current Security Level (as defined either as the compulsory Session Security Level set at initialization of the Secure Channel Session or as the Current Security Level set by the setSecurityLevel() method). If the method fails, the Current Security Level is reset to NO_SECURITY_LEVEL, but not the compulsory Session Security Level;*
- *This method does not fail if a Secure Channel Session is not open (that is, when Session Security Level = NO_SECURITY_LEVEL) or if the corresponding session keys are not available.*

Parameters:

`baBuffer` - the source of the data to be wrapped, with the status bytes included at the end. This buffer must be global.

`sOffset` - the offset within the source buffer of the data to wrap

`sLength` - the length of the data to wrap

Note:

- The length Li of the Response Data Field is automatically calculated by the Security Domain and prepended to the Response Data Field for computation of the R-MAC.

Returns:

new length of wrapped data (i.e. the length of the response data field to be transmitted)

Throws:

`javacard.framework.ISOException` – if security mechanism related status bytes might be returned

`java.lang.ArrayIndexOutOfBoundsException` - if wrapping would cause access of data outside array bounds

---

org.globalplatform

Interface **SecureChannelx**

`public interface` **SecureChannelx** `extends org.globalplatform.SecureChannel`

This defines an interface which extends `SecureChannel` Interface and includes a supplementary method. See SecureChannel interface for a description of the underlying interface. Prior to using this interface, an Application is required to obtain a handle to its associated Security Domain's `SecureChannelx` interface object by invoking the `GPSystem.getSecureChannel()` method and casting the returned object to type `SecureChannelx`. The `SecureChannelx` Interface shall be implemented by a Security Domain compliant to this version of the specification and the corresponding object reference shall be exposed through the `GPSystem.getSecureChannel()` method.

Method Summary

void	**setSecurityLevel**`(byte bSecurityLevel)` This method updates the Current Security Level for all subsequent invocations of the `wrap()` and `unwrap()` methods, except when a Secure Channel Session is not active or was aborted during the same Application session.

Method Detail

---

setSecurityLevel

`public void` **setSecurityLevel**`(byte bSecurityLevel)`

This method updates the Current Security Level for all subsequent invocations of the `wrap()` and `unwrap()` methods, except when a Secure Channel Session is not active or was aborted during the same Application session. Current Security Level is coded as a bit-map according to Table 10-1. The Current Security Level cannot be set below the compulsory Session Security Level, but only equal or above. The Current Security Level may be increased or decreased during a Secure Channel Session as long as it is at least equal to the compulsory Session Security Level.

Notes:

- *The method fails if a Secure Channel Session is not open, if the corresponding session keys are not available or if the Current Security Level is equal to NO_SECURITY_LEVEL.*

Parameters:

`SecurityLevel` - the Current Security Level to be set

Throws:

`javacard.framework.ISOException` - with the following reason codes:

- `ISO7816.SW_CONDITIONS_NOT_SATISFIED`

A.2 GlobalPlatform on MULTOS™

A.2.1 Glossary of Terms

Term	Definition
AAM	Application Abstract Machine, another term for runtime environment. The AAM is the virtual machine that executes MEL applications.
Application Load Certificate	A digital certificate that provides card issuers with the authority to load an Application onto a MULTOS smart card.
Public	The name of a memory segment provided by the MULTOS AAM to MEL Applications. *Public* is visible to all Applications on a MULTOS smart card. On receiving a command MULTOS places the command APDU in *Public* prior to invoking an Application; Applications place their response APDU in *Public* for transmission by MULTOS. The bottom of *Public* is pointed to by the address register *Public Base*, normally abbreviated to *PB*; the top of *Public* is pointed to by the address register *Public Top*, normally abbreviated to *PT*. The byte at the bottom of *Public* has the tagged address PB[0]; the byte at the top of *Public* has the tagged address PT[-1]. The top seventeen bytes of *Public* (i.e. the addresses PT[-17] … PT[-1]) are reserved for storing the command APDU header, response status bytes and other information related to APDU exchange.
Segment address	The sixteen-bit virtual address assigned to an item of data by the MULTOS AAM (q.v. tagged address).
Tagged address	The address of an item of data when represented using the syntax *Address register[Offset from address register]* (q.v. *segment address*).

Table A-1: Glossary of Terms

A.2.2 GlobalPlatform Specific Requirements

A.2.2.1 Environment

A GlobalPlatform on MULTOS smart card contains a minimum of two applications:

- A “Command Dispatcher” application. This application provides the command dispatch services specified in section 6.3. This application has the AID ‘A000000144475031’;
- A “Card Manager Services Provider” application. This application provides the Card Manager services specified in section 6.2. This application has the AID ‘A000000144475032’.

These two applications provide MEL Applications loaded by card issuers with a GlobalPlatform runtime environment.

A.2.2.2 Application Life Cycle States

When an Application is loaded onto a GlobalPlatform for MULTOS smart card its initial Life Cycle State is SELECTABLE. This reflects the fact that a MEL Application is able to receive commands from off-card entities as soon as it has been loaded onto a card.

When a Security Domain is loaded onto a GlobalPlatform for MULTOS smart card its initial Life Cycle State is SELECTABLE.

A.2.2.3 Privileges

When an Application is loaded onto a GlobalPlatform for MULTOS smart card it does not have any privileges. Once an Application has been loaded its privileges can be changed using the INSTALL [for registry update] command.

A.2.2.4 Registration

An Application should register itself with the Card Manager. This registration is performed using the `registerApplication()` function.

A.2.3 Accessing Card Manager Services

A.2.3.1 Introduction

This section describes the mechanism used by MEL Applications to access Card Manager services.

The description assumes the existence of a generic Card Manager service **S** having *N* input parameters $p_1 \ldots p_N$ and returning a value *r*. The interface to this service can be described using the following C programming language syntax:

`tr` **`S`** `(`tp_1 p_1`,` tp_2 p_2`, …,` tp_{N-1} p_{N-1}`,` tp_N p_N`);`

Parameters:

p_1 – A parameter of type tp_1. The size of this parameter is sp_1 bytes.

p_2 – A parameter of type tp_2. The size of this parameter is sp_2 bytes.

.

.

.

p_{N-1} - A parameter of type tp_{N-1}. The size of this parameter is sp_{N-1} bytes.

p_N - A parameter of type tp_N. The size of this parameter is sp_N bytes.

Returns:

A value of type `tr`. The size of this value is *sr* bytes. If the type represented by `tr` is `void` then *sr* is zero.

A.2.3.2 The “Card Manager services interface buffer”

A MEL Application must be able to exchange data with the Card Manager when requesting a service. Therefore, an area of the MULTOS *Public* memory segment is reserved in order to allow:

- A MEL Application requesting Card Manager services to specify both the requested service and the value of any associated parameters;

- The Card Manager to return the *SCode* status condition for the requested service and any return value.

Figure A-1 : *Public* and the "Card Manager services interface buffer" illustrates the relationship of this reserved area of *Public*, the “Card Manager services interface buffer”, to the rest of *Public*.

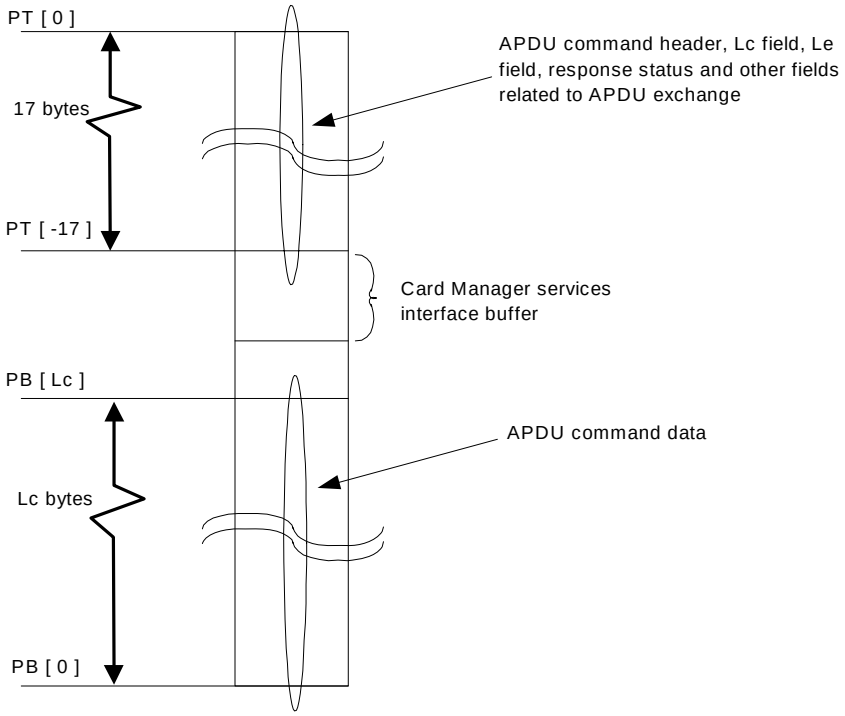

Figure A-1 : *Public* and the "Card Manager services interface buffer"

When a MEL Application requests a Card Manager service the “Card Manager services interface buffer” is structured according to the following table.

Name	Description	Address	Source	Size (bytes)
Service	The Card Manager service requested by the Application.	PT[-19]	Application	2
SCode	The status condition reported by the Card Manager as a result of its attempt to provide the service requested by the Application.	PT[-21]	Card Manager	2
p_N	The value of the *N*th parameter associated with the service.	PT[-21 - sp_N]	Application	sp_N

Name	Description	Address	Source	Size (bytes)
p_{N-1}	The value of the *N-1*th parameter associated with the service.	PT[-21 - sp_N - sp_{N-1}]	Application	sp_{N-1}
. . .	. . .	. . .	. . .	. . .
p_2	The value of the second parameter associated with the service.	PT[-21 - sp_N - sp_{N-1} - … - sp_2]	Application	Sp_2
p_1	The value of the first parameter associated with the service.	PT[-21 - sp_N - sp_{N-1} - … - sp_2 - sp_1]	Application	Sp_1
r	The return value for the service.	PT[-21 - sr]	Card Manager	Sr

Table A-2 : Structure of the "Card Manager services interface buffer"

The field *Service* in the “Card Manager services interface buffer” identifies the Card Manager service requested by the MEL Application. The Card Manager services are exposed to an Application via the interfaces described in section A.2.4. The mapping of these interfaces to values of *Service* is shown in the following table.

Card Manager Service		*Service*
Module	**Function**	
Secure Channel	`decryptData()`	‘0101’
	`encryptData()`	‘0102’
	`getSecurityLevel()`	‘0103’
	`processSecurity()`	‘0104’
	`resetSecurity()`	‘0105’
	`unwrap()`	‘0106’
	`wrap()`	‘0107’
	`setSecurityLevel()`	‘0108’
GPSystem	`getCardContentState()`	‘0201’
	`getCardState()`	‘0202’
	`getCVM()`	‘0203’
	`lockCard()`	‘0204’
	`registerApplication()`	‘0205’
	`setATRHistBytes()`	‘0206’
	`setCardContentState()`	‘0207’
	`terminateCard()`	‘0208’

Card Manager Service		*Service*
Module	**Function**	
	`getRegistryEntry()`	'0209'
	`getService()`	'020A'
CVM	`blockState()`	'0301'
	`getTriesRemaining()`	'0302'
	`isActive()`	'0303'
	`isBlocked()`	'0304'
	`isSubmitted()`	'0305'
	`isVerified()`	'0306'
	`resetState()`	'0307'
	`resetAndUnblockState()`	'0308'
	`setTryLimit()`	'0309'
	`update()`	'030A'
	`verify()`	'030B'
GPRegistryEntry	`deregisterService()`	'0401'
	`getAID()`	'0402'
	`getPrivileges()`	'0403'
	`getState()`	'0404'
	`isAssociated()`	'0405'
	`isPrivileged()`	'0406'
	`registerService()`	'0407'
	`setState()`	'0408'
GlobalService	`getServiceInterface()`	'0501'

Table A-3 : Encoding of *Service*

A.2.3.3 Requesting Card Manager services

In order for a MEL Application to request a Card Manager service the Application must:

- Populate the "Card Manager services interface buffer" with arguments that identify the requested Card Manager service and the value of any associated parameters $p_1 \ldots p_N$;
- Invoke the MULTOS *Delegate* primitive, specifying the AID of the "Card Manager Services Provider" application as the identifier of the delegate application;
- Where necessary, examine the value *SCode* returned in the "Card Manager services interface buffer";
- Where appropriate, examine the value *r* returned in the "Card Manager services interface buffer".

A.2.4 API

A.2.4.1 Programming Language

The API to the Card Manager services is specified using the syntax of the programming language *C* as defined by [9899]; the API is presented as a set of *C* function prototype declarators.

Within this API:

- The elements `bool`, `false` and `true` are assumed to have the definitions specified in the `<stdbool.h>` header;
- The element `NULL` is assumed to have the definition specified in the `<stddef.h>` header.

A.2.4.2 Header file

The function prototype declarators specified in this chapter shall be declared in a header `globalplatform.h`.

A.2.4.3 Representation of Types

The API makes some assumptions regarding the representation of types. These assumptions are stated in the following table.

Type, `T`	`sizeof (T)`	Other Assumptions
`bool`	1	–
`unsigned short int`	2	–
`AID`	–	Objects of this type do not contain any padding, either within or at the end of the object
`*AID`	2	The type "pointer to `AID`" represents a MULTOS AAM segment address.
`CVM_HANDLE`	–	Objects of this type do not contain any padding, either within or at the end of the object
`*CVM_HANDLE`	2	The type "pointer to `CVM_HANDLE`" represents a MULTOS AAM segment address.

Table A-4: Types

A.2.4.4 Location of APDUs

The API assumes that command and response APDUs reside in the MULTOS *Public* memory segment. This memory segment may be regarded as being defined as:

```
static unsigned char ucaPublic[PUBLICSIZE];
```

where:

- The element `PUBLICSIZE` represents the size in bytes of the MULTOS *Public* memory segment for a particular MULTOS implementation;
- The expression `&ucaPublic[0]` has the value of the MULTOS AAM segment address corresponding to the tagged address *PB[0]*.

A.2.4.5 Module Summary

The following modules implement the individual functions:

- Status
- SecureChannel
- GPSystem
- CVM
- GPRegistryEntry
- GlobalService

A.2.4.6 Status

This module defines an interface to allow an Application to interrogate the status condition of the most recent API function invocation.

Function Summary

SCode

```
unsigned short int SCode(void);
```

This function returns the status condition of the API function most recently invoked by the Application.

SetSCode

```
void SetSCode(unsigned short int usStatusCode);
```

This function may be used by an Application or API function to set the status condition returned by a subsequent invocation of the `SCode()` function.

Enumeration Detail

ISO7816

```
enum ISO7816 {
SW_CLA_NOT_SUPPORTED = 0x6E00,
SW_COMMAND_NOT_ALLOWED = 0x6986,
SW_CONDITIONS_NOT_SATISFIED = 0x6985,
SW_CORRECT_LENGTH_00 = 0x6C00,
SW_DATA_INVALID = 0x6984,
SW_FILE_FULL = 0x6A84,
SW_FILE_INVALID = 0x6983,
SW_FILE_NOT_FOUND = 0x6A82,
SW_FUNC_NOT_SUPPORTED = 0x6A81,
SW_INCORRECT_P1P2 = 0x6A86,
SW_INS_NOT_SUPPORTED = 0x6D00,
SW_LOGICAL_CHANNEL_NOT_SUPPORTED = 0x6881,
SW_NO_ERROR = 0x9000,
```

```
SW_RECORD_NOT_FOUND = 0x6A83,
SW_SECURE_MESSAGING_NOT_SUPPORTED = 0x6882,
SW_SECURITY_STATUS_NOT_SATISFIED = 0x6982,
SW_UNKNOWN = 0x6F00,
SW_WARNING_STATE_UNCHANGED = 0x6200,
SW_WRONG_DATA = 0x6A80,
SW_WRONG_LENGTH = 0x6700,
SW_WRONG_P1P2 = 0x6B00
};
```

This enumeration defines enumeration constants used to represent ISO/IEC 7816 response status values defined in this specification.

Function Detail

SCode

```
unsigned short int SCode(void);
```

This function returns the status condition of the API function most recently invoked by the Application.

Notes:

- An invocation of this function does not affect the status condition returned by subsequent invocations of this function;
- The status condition returned by this function may have the same value as an enumeration constant of the enumeration `ISO7816`.

Returns:

If the Application has not invoked an API function since it was selected then `SW_NO_ERROR` otherwise the status condition of the API function most recently invoked by the Application.

SetSCode

```
void SetSCode(unsigned short int usStatusCode);
```

This function may be used by an Application or API function to set the status condition returned by a subsequent invocation of the `SCode()` function.

Parameter:

`usStatusCode` – Status condition.

A.2.4.7 SecureChannel

This module defines an interface to be used by an Application that may want to delegate the handling of entity authentication and APDU security to its associated Security Domain. This interface is designed to offer interoperability to the Application in that it requires no knowledge of the mechanisms used to perform the authentication or of the scheme used for APDU security and shall allow an Application to interface correctly with a Security Domain immaterial of the mechanisms or schemes used.

If the card supports logical channels, it is the responsibility of the Security Domain to correctly manage the logical channel information when processing the APDU.

Constant Summary

```
#define ANY_AUTHENTICATED 0x40
```

The off-card entity has been authenticated but not as the provider of this Application.

```
#define AUTHENTICATED 0x80
```

The off-card entity has been authenticated as the provider of this Application.

```
#define C_DECRYPTION 0x02
```

The `unwrap()` function will decrypt incoming command data.

```
#define C_MAC 0x01
```

The `unwrap()` function will verify the MAC on an incoming command.

```
#define NO_SECURITY_LEVEL 0x00
```

Entity Authentication has not occurred.

```
#define R_ENCRYPTION 0x20
```

The `wrap()` function will encrypt the outgoing response data.

```
#define R_MAC 0x10
```

The `wrap()` function will generate a MAC for outgoing response data.

Function Summary

```
unsigned short int decryptData(unsigned short int usOffset, unsigned short
int usLength);
```

This function is used to decrypt data located in the MULTOS *Public* memory segment.

```
unsigned short int encryptData(unsigned short int usOffset, unsigned short
int usLength);
```

This function is used to encrypt data located in the MULTOS *Public* memory segment.

```
unsigned char getSecurityLevel(void);
```

This function is used to determine whether the Security Domain has performed authentication and to determine what level of security will be applied by the `wrap()` and `unwrap()` functions.

```
unsigned short int processSecurity(void);
```

Processes security related APDU commands.

```
void resetSecurity(void);
```

This function is used to reset information relating to the current Secure Channel Session.

```
void setSecurityLevel(unsigned char ucSecurityLevel);
```

This function is used to set the level of security to be applied by the `wrap()` and `unwrap()` functions for the next command - response pair.

```
unsigned short int unwrap(unsigned short int usOffset, unsigned short int
usLength);
```

This function is used to process and verify the secure messaging of an incoming command according to the security level.

```
unsigned short int wrap(unsigned short int usOffset, unsigned short int
usLength);
```

This function is used to apply additional security processing to outgoing response data and status bytes according to the security level.

Constant Detail

ANY_AUTHENTICATED

```
#define ANY_AUTHENTICATED 0x40
```

Note:

- Entity authentication and the level of security that will be applied by the `wrap()` and `unwrap()` functions are not necessarily related. A Security Domain, by default, could verify the MAC on any command passed as a parameter in the `unwrap()` function without entity authentication previously having occurred.

AUTHENTICATED

```
#define AUTHENTICATED 0x80
```

The off-card entity has been authenticated as the provider of this Application.

Note:

- Entity authentication and the level of security that will be applied by the `wrap()` and `unwrap()` functions are not necessarily related. A Security Domain, by default, could verify the MAC on any command passed as a parameter in the `unwrap()` function without entity authentication previously having occurred.

C_DECRYPTION

```
#define C_DECRYPTION 0x02
```

The `unwrap()` function will decrypt incoming command data.

Note:

- Command data decryption could be indicated along with entity authentication and one or more levels of security.

C_MAC

```
#define C_MAC 0x01
```

The `unwrap()` function will verify the MAC on an incoming command.

Note:

- MAC verification could be indicated along with entity authentication and one or more levels of security e.g. a value of '03' indicates that while entity authentication has not occurred, the `unwrap()` function will decrypt the command data of incoming commands and verify the MAC on incoming commands.

R_ENCRYPTION

```
#define R_ENCRYPTION 0x20
```

The `wrap()` function will encrypt the outgoing response data.

Note:

Response data encryption could be indicated along with entity authentication and one or more levels of security.

R_MAC

```
#define R_MAC 0x10
```

The `wrap()` function will generate a MAC for the outgoing response data.

Note:

- MAC generation could be indicated along with entity authentication and one or more levels of security e.g. a value of '91' indicates that entity authentication has occurred and that the `unwrap()` function will verify the MAC on incoming commands and that the `wrap()` function will generate a MAC on outgoing response data.

NO_SECURITY_LEVEL

```
#define NO_SECURITY_LEVEL 0x00
```

Entity authentication has not occurred.

Notes:

- This indicates that no Secure Channel Session is active, or that a security error has occurred during the current Secure Channel Session, or that the previous Secure Channel Session was aborted during the same Application Session);
- Entity authentication and the level of security that will be applied by the wrap and unwrap functions are not necessarily related. A Security Domain, by default, could verify the MAC on any command passed as a parameter in the `unwrap()` function without entity authentication previously having occurred;
- The `wrap()` and `unwrap()` functions will not apply any cryptographic processing to command or response data.

Function Detail

processSecurity

```
unsigned short int processSecurity(void);
```

This function processes security related APDU commands.

This function is used by an Application to process APDU commands that possibly relate to the security mechanism used by the Security Domain. As the intention is to allow an Application to be associated with a Security Domain without having any knowledge of the security mechanisms used by the Security Domain, the Application assumes that APDU commands that it does not recognize are part of the security mechanism and will be recognized by the Security Domain. The Application can either invoke this function prior to determining if it recognizes the command or only invoke this function for commands it does not recognize.

The function sets the compulsory Session Security Level that is established at Secure Channel initiation and which is required for the whole Secure Channel Session. On successful initialization of the Secure Channel Session, the Current Security Level is set to the same value as the compulsory Session Security Level. The Current Security Level is updated (R-MAC or not) on the successful processing of the BEGIN R-MAC SESSION / END R-MAC SESSION commands.

Notes:

- The processing performed by this function shall comply with the rules of the Secure Channel Protocol implemented by the Security Domain;

- The function is responsible for receiving the data field of commands that are recognized;
- The Application is responsible for recognizing commands that the function refused to process ('6E00' and '6D00');
- The Application is responsible for outputting status bytes returned due to the processing of instructions recognized by the function;
- If response data is present, this data will be placed in the MULTOS *Public* memory segment at offset zero. The return code indicates the length and the Application is responsible for outputting this data.

Returns:

The number of bytes to be output.

`SCode()`:

May return one of the following status codes (other security mechanism related status codes may be returned):

- `SW_CLA_NOT_SUPPORTED` if class byte is not recognized by the function;
- `SW_INS_NOT_SUPPORTED` if instruction byte is not recognized by the function;
- `SW_NO_ERROR` if the command was processed successfully by the function.

`wrap`

```
unsigned short int wrap(unsigned short int usOffset, unsigned short int
usLength);
```

This function applies secure messaging to the current outgoing response according to the Current Security Level and Session Security Level of the Secure Channel Session.

Notes:

- The processing performed by this function shall comply with the rules of the Secure Channel Protocol implemented by the Security Domain;
- This function attempts secure messaging processing of the current outgoing response when the Current Security Level indicates R_MAC and/or R_ENCRYPTION;
- If the Current Security Level does not indicate R_MAC and/or R_ENCRYPTION, when complying with the Secure Channel Protocol rules, this function will do no processing and the outgoing response message will remain as is in the MULTOS *Public* memory segment. The returned length of the "wrapped" data is set to the value of the `usLength` parameter minus 2 (indicating the status words are no longer present at the end of the returned data);
- The Application is responsible for appending the expected status words at the end of the response data in order for them to be protected by secure messaging;
- The returned data does not include the status words;
- The function fails if the secure messaging of the incoming command is not successfully verified or does not match the Current Security Level (as defined either as the compulsory Session Security Level set at initialization of the Secure Channel Session or as the Current Security Level set by the `setSecurityLevel()` function). If the function fails, the Current Security Level is reset to NO_SECURITY_LEVEL, but not the compulsory Session Security Level;
- • This function does not fail if a Secure Channel Session is not open (that is, when Session Security Level = NO_SECURITY_LEVEL) or if the corresponding session keys are not available;

- Normally the MULTOS *Public* memory segment address range *PB[0]…PT[-17]* is available for outgoing response data. However, for this function the address range is restricted to *PB[0]…PT[-23]*.

Parameters:

`usOffset` - The offset within the MULTOS *Public* memory segment of the data to wrap.

`usLength` - The length of the data to wrap.

Returns:

New length of wrapped data (i.e. the length of the response data field to be transmitted).

`SCode()`:

May return one of the following:

- A security mechanism related status;
- `SW_NO_ERROR` if the data was processed successfully by the `wrap()` function.

unwrap

```
unsigned short int unwrap(unsigned short int usOffset, unsigned short int
usLength);
```

This function is used to process and verify the secure messaging of an incoming command according to the Current Security Level and Session Security Level of the current Secure Channel Session.

Notes:

- The processing performed by this function shall comply with the rules of the Secure Channel Protocol implemented by the Security Domain;
- If `NO_SECURITY_LEVEL`, `AUTHENTICATED` or `ANY_AUTHENTICATED` only is indicated, this function will not attempt any secure messaging processing on the incoming command and the incoming command will remain as is within the MULTOS *Public* memory segment. The returned length of the "unwrapped" data is set to the value of the `usLength` parameter;
- If the class byte does not indicate secure messaging (according to ISO/IEC 7816-4), this function will not attempt any secure messaging processing on the incoming command and the incoming command will remain as is within the MULTOS *Public* memory segment. When complying with the Secure Channel Protocol rules, the returned length of the "unwrapped" data is set to the value of the `usLength` parameter otherwise a security error is returned;
- The Application is responsible for receiving the data field of the command;
- Correct secure messaging processing of the `unwrap()` will result in the incoming command being reformatted within the MULTOS *Public* memory segment with all data relating to the secure messaging removed. A reformatted case 1 or case 2 command will include an Lc byte set to zero;
- If `R_MAC` is indicated in the Current Security Level of the Secure Channel Session, once secure messaging processing of the incoming command has successfully completed, R-MAC computation on the reformatted command (i.e. after all the data relating to secure messaging has been removed) will be initiated. If no secure messaging processing was required for the incoming command, R-MAC computation will be initiated on the unmodified incoming command, appended with a Lc byte of zero in event of a case 1 or case 2 command;
- Incorrect processing of the `unwrap()` will result in the information relating to the current Secure Channel being reset;

- Normally the MULTOS *Public* memory segment address range *PB[0]…PT[-17]* is available for incoming command data. However, for this function the address range is restricted to *PB[0]…PT[-23]*.

Parameters:

`usOffset` - The offset within the MULTOS *Public* memory segment of the data to unwrap.

`usLength` - The length of the data to unwrap.

Returns:

The length of the unwrapped data i.e. the length of the command data.

`SCode()`:

May return one of the following status codes (other security mechanism related status codes may be returned):

- `SW_SECURITY_STATUS_NOT_SATISFIED` if secure messaging verification failed;
- `SW_CLA_NOT_SUPPORTED` if class byte is not recognized by the function;
- `SW_NO_ERROR` if the command was processed successfully by the function.

`decryptData`

```
unsigned short int decryptData(unsigned short int usOffset, unsigned short int usLength);
```

This function is used to decrypt data located in the MULTOS *Public* memory segment.

Notes:

- The processing performed by this function shall comply with the rules of the Secure Channel Protocol implemented by the Security Domain;
- The Security Domain implicitly knows the key used for decryption;
- The Security Domain is implicitly aware of any padding that may be present in the decrypted data according to the Secure Channel Protocol and the eventual padding is discarded;
- The clear text data replaces the encrypted data within the MULTOS *Public* memory segment;
- The Application is responsible for checking the integrity of the decrypted data;
- No part of either the encrypted data or the decrypted data should reside outside the MULTOS *Public* memory segment address range *PB[0]… PT[-23]*.

Parameters:

`usOffset` - Offset within the MULTOS *Public* memory segment to start the decryption.

`usLength` - The number of bytes to decrypt.

Returns:

The length of the clear text data.

`SCode()`:

May return one of the following status codes:

- `SW_SECURITY_STATUS_NOT_SATISFIED` if a Secure Channel Session has not been opened;

- `SW_WRONG_LENGTH` if the length of data to be decrypted is not valid;
- `SW_NO_ERROR` if the encrypted data was processed successfully by the `decryptData()` function.

`encryptData`

```
unsigned short int encryptData(unsigned short int usOffset, unsigned short int usLength);
```

This function is used to encrypt data located in the MULTOS *Public* memory segment.

Notes:

- The processing performed by this function shall comply with the rules of the Secure Channel Protocol implemented by the Security Domain;
- The Security Domain is implicitly aware of any padding that must be applied to the clear text data prior to encryption according to the Secure Channel Protocol;
- The Security Domain implicitly knows the key used for encryption;
- The encrypted data replaces the clear text data within the MULTOS *Public* memory segment;
- No part of either the clear text data or the encrypted data should reside outside the MULTOS *Public* memory segment address range *PB[0]… PT[-23]*.

Parameters:

`usOffset` - Offset within the MULTOS *Public* memory segment to start the encryption.

`usLength` - The number of bytes to encrypt.

Returns:

The length of the encrypted data.

`SCode()`:

May return one of the following status codes:

- `SW_SECURITY_STATUS_NOT_SATISFIED` if a Secure Channel Session has not been opened.
- `SW_NO_ERROR` if the data was processed successfully by the `encryptData()` function.

`resetSecurity`

```
void resetSecurity(void);
```

This function is used to reset all information relating to the current Secure Channel Session. It resets both the compulsory Session Security Level and the Current Security Level to NO_SECURITY_LEVEL, terminates the current Secure Channel Session and erases all session keys.

Notes:

- When an Application using the services of a Security Domain is de-selected the OPEN will invoke the `resetSecurity()` function on behalf of the Application;
- The Security Domain will reset all information relating to the current Secure Channel Session i.e. all Secure Channel Session keys, state information and Security Level information will be erased;

- This function shall not fail if no Secure Channel Session is open (i.e. Session Security Level = NO_SECURITY_LEVEL).

`setSecurityLevel`

```
void setSecurityLevel(unsigned char ucSecurityLevel);
```

This function updates the Current Security Level for all subsequent invocations of the `wrap()` and `unwrap()` functions, except when a Secure Channel Session is not active or was aborted during the same Application session. Current Security Level is coded as a bit-map according to Table 10-1. The Current Security Level cannot be set below the compulsory Session Security Level, but only equal or above. The Current Security Level may be increased or decreased during a Secure Channel Session as long as it is at least equal to the compulsory Session Security Level.

Notes:

- The function fails if a Secure Channel Session is not open, if the corresponding session keys are not available or if the Current Security Level is equal to NO_SECURITY_LEVEL.

Parameters:

`ucSecurityLevel` - The Current Security Level to be set.

`SCode():`

May return the following status code:

- `SW_CONDITIONS_OF_USE_NOT_SATISFIED`

`getSecurityLevel`

```
unsigned char getSecurityLevel(void);
```

This function returns the Current Security Level coded as a bit-map according to section 10.6 indicating whether entity authentication has occurred and what level of security is currently applicable to command and response messages processed by the `wrap()` and `unwrap()` functions.

Notes:

- Applications must invoke this function to ensure that Application specific security requirements have been previously met or will be enforced by the Security Domain;
- More than one level of security may be active and these may change during a Secure Channel Session, e.g. an R_MAC session may be initiated during a C_MAC session.

Returns:

The Current Security Level.

A.2.4.8 GPSystem

The System module exposes a subset of the behavior of the OPEN to the outside world. The OPEN implements and enforces a Card Issuer's security policy relating to these services. This OPEN module provides functionality at the same level as the run-time environment i.e. the "system" context with special privileges.

Constant Summary

```
#define APPLICATION_INSTALLED 0x03
```

The current Application is in the Life Cycle State of INSTALLED.

```
#define APPLICATION_LOCKED 0x80
```

The current Application is in the Life Cycle State of LOCKED.

```
#define APPLICATION_UNLOCKED 0x7F
```

The current Application is in its previous State.

```
#define APPLICATION_SELECTABLE 0x07
```

The current Application is in the Life Cycle State of SELECTABLE.

```
#define CARD_OP_READY 0x01
```

The card is in the Life Cycle State of OP_READY.

```
#define CARD_INITIALIZED 0x07
```

The card is in the Life Cycle State of INITIALIZED.

```
#define CARD_LOCKED 0x7F
```

The card is in the Life Cycle State of CARD_LOCKED.

```
#define CARD_SECURED 0x0F
```

The card is in the Life Cycle State of SECURED.

```
#define CARD_TERMINATED 0xFF
```

The card is in the Life Cycle State of TERMINATED.

```
#define CVM_GLOBAL_PIN 0x11
```

Indicates that the CVM interface required is a global PIN.

```
#define FAMILY_SECURE_CHANNEL 0x81
```

Indicates the family of the Secure Channel Global Service Identifier.

```
#define FAMILY_CVM 0x82
```

Indicates the family of the CVM Global Service Identifier.

```
#define FAMILY_USSM 0xA0
```

Indicates the family of the USSM Global Service Identifier.

```
#define FAMILY_SERVICE_IDENTIFIER 0x80
```

Indicates the family of the Secure Channel Global Service Identifier.

```
#define SECURITY_DOMAIN_PERSONALIZED 0x0F
```

The current Security Domain is in the Life Cycle State of PERSONALIZED.

Type Summary

AID

```
typedef struct tagAID {
      unsigned char     ApplicationIdentifierLength;
      char        ApplicationIdentifier[];
```

```
} AID;
```

This type represents an ISO/IEC 7816-4 AID.

CVM_HANDLE

```
typedef struct tagCVM_HANDLE {
      unsigned char     CVMIdentifierLength;
      char        CVMIdentifier[];
} CVM_HANDLE;
```

This type represents a handle to a CVM interface.

GPRE_HANDLE

```
typedef unsigned short GPRE_HANDLE;
```

This type represents a handle to a GlobalPlatform Registry entry.

SERVICE_HANDLE

```
typedef unsigned short SERVICE_HANDLE;
```

This type represents a handle to the services interface of a Global Services Application.

Function Summary

```
unsigned char getCardContentState(void)
```

This function returns the Life Cycle State of the current Application.

```
unsigned char getCardState(void)
```

This function returns the Life Cycle State for the card.

```
CVM_HANDLE *getCVM(unsigned char ucCVMIdentifier)
```

This function returns a handle to a CVM Application.

```
GPRE_HANDLE getRegistryEntry(const AID *reqAID)
```

This function returns a handle to a GlobalPlatform Registry entry.

```
AID *getService(const AID *serverAID, unsigned short usServiceName)
```

This function returns the AID of a Global Services Application.

```
bool lockCard(void);
```

This function locks the card.

```
bool registerApplication(const AID *pSDIdentifier);
```

This function registers a MEL Application with the Card Manager.

```
bool setATRHistBytes(unsigned short int usOffset, unsigned char ucLength);
```

This function sets the historical bytes, for contact cards according to ISO/IEC 7816-4 and for Type A contactless cards according to ISO/IEC 14443-3.

```
bool setCardContentState(unsigned char ucState);
```

This function sets the Application specific Life Cycle State of the current Application.

```
bool terminateCard(void);
```

This function terminates the card.

Constant Detail

APPLICATION_INSTALLED

```
#define APPLICATION_INSTALLED 0x03
```

The current Application is in the Life Cycle State of INSTALLED.

Note:

- The Life Cycle State INSTALLED could be indicated along with another Application specific Life Cycle State e.g. a value of '07' indicates that the Application has been made selectable.

APPLICATION_LOCKED

```
#define APPLICATION_LOCKED 0x80
```

The current Application is in the Life Cycle State of LOCKED.

APPLICATION_UNLOCKED

```
#define APPLICATION_UNLOCKED 0x7F
```

The current Application is in its previous State.

APPLICATION_SELECTABLE

```
#define APPLICATION_SELECTABLE 0x07
```

The current Application is in the Life Cycle State of SELECTABLE.

Note:

- The Life Cycle State SELECTABLE could be indicated along with another Application specific Life Cycle State.

SECURITY_DOMAIN_PERSONALIZED

```
#define SECURITY_DOMAIN_PERSONALIZED 0x0F
```

The current Security Domain is in the Life Cycle State of PERSONALIZED.

CARD_OP_READY

```
#define CARD_OP_READY 0x01
```

The card is in the Life Cycle State of OP_READY.

CARD_INITIALIZED

```
#define CARD_INITIALIZED 0x07
```

The card is in the Life Cycle State of CARD_INITIALIZED.

CARD_SECURED

```
#define CARD_SECURED 0x0F
```

The card is in the Life Cycle State of CARD_SECURED.

CARD_LOCKED

```
#define CARD_LOCKED 0x7F
```

The card is in the Life Cycle State of CARD_LOCKED.

CARD_TERMINATED

```
#define CARD_TERMINATED 0xFF
```

The card is in the Life Cycle State of TERMINATED.

CVM_GLOBAL_PIN

```
#define CVM_GLOBAL_PIN 0x11
```

Indicates that the CVM interface required is a global PIN.

```
#define FAMILY_SECURE_CHANNEL 0x81
```

Indicates the family of the Secure Channel Global Service Identifier.

```
#define FAMILY_CVM 0x82
```

Indicates the family of the CVM Global Service Identifier.

```
#define FAMILY_USSM 0xA0
```

Indicates the family of the USSM Global Service Identifier.

```
#define FAMILY_SERVICE_IDENTIFIER 0x80
```

Indicates the family of the Secure Channel Global Service Identifier.

```
#define SECURITY_DOMAIN_PERSONALIZED 0x0F
```

The current Security Domain is in the Life Cycle State of PERSONALIZED.

Type Detail

AID

```
typedef struct tagAID {
      unsigned char     ApplicationIdentifierLength;
      char        ApplicationIdentifier[];
} AID;
```

This type represents an ISO/IEC 7816-4 AID.

CVM_HANDLE

```
typedef struct tagCVM_HANDLE {
      unsigned char     CVMIdentifierLength;
      char        CVMIdentifier[];
} CVM_HANDLE;
```

This type represents a handle to a CVM interface.

GPRE_HANDLE

```
typedef unsigned short GPRE_HANDLE;
```

This type represents a handle to a GlobalPlatform Registry entry.

SERVICE_HANDLE

```
typedef unsigned short SERVICE_HANDLE;
```

This type represents a handle to the services interface of a Global Services Application.

Function Detail

getCardContentState

```
unsigned char getCardContentState(void);
```

This function returns the Life Cycle State of the current Application.

Note:

- The OPEN locates the entry of the current Application in the GlobalPlatform Registry and retrieves the Life Cycle State.

Returns:

The Life Cycle State of the current Application.

See Also:

`APPLICATION_INSTALLED`, `APPLICATION_SELECTABLE`, `APPLICATION_LOCKED`, `SECURITY_DOMAIN_PERSONALIZED`

getCardState

```
unsigned char getCardState(void);
```

This function returns the Life Cycle State for the card.

Returns:

The Life Cycle State of the card.

See Also:

`CARD_OP_READY`, `CARD_INITIALIZED`, `CARD_SECURED`, `CARD_LOCKED`, `CARD_TERMINATED`

getCVM

```
CVM_HANDLE *getCVM(unsigned char ucCVMIdentifier)
```

This function returns a handle to a CVM Application.

Notes:

- The OPEN searches the GlobalPlatform Registry for the CVM Application that has the Global Service privilege and is either:
  - Uniquely registered with service family `FAMILY_CVM` and service identifier `ucCVMIdentifier`; or
  - Registered for the entire service family `FAMILY_CVM`.
- If the search is successful then the OPEN invokes the `getServiceInterface()` function of the CVM Application the with the following parameters:
  - `hClientRegistryEntry` set to identify the GlobalPlatform Registry entry of the the requesting on-card entity;
  - `sServiceName` with first byte set to `FAMILY_CVM` (indicating the CVM family) and second byte set to `ucCVMIdentifier`;
  - `usOffset` and `usLength` set to zero.

- If the search and the invocation of the `getServiceInterface()` function are successful then the OPEN returns the AID of the corresponding CVM Application as a handle. If either is unsuccessful then the OPEN returns `NULL`.

Parameters:

`ucCVMIdentifier` - Identifies the required CVM interface.

Returns:

The handle to the required CVM Application or `NULL` if there is no matching CVM Application.

See Also:

`CVM_GLOBAL_PIN`

getRegistryEntry

```
GPRE_HANDLE getRegistryEntry(const AID *reqAID);
```

This function allows an Application (e.g. a CVM) to access the GlobalPlatform Registry entry of another Application. If no AID is input, this function provides the GlobalPlatform Registry entry of the requesting Application. It returns a handle to the GlobalPlatform Registry entry.

Notes:

- The OPEN verifies that the requesting Application has the Global Registry privilege, or is the Security Domain associated to the Application being investigated, or is the investigated Application itself.

Parameters:

`reqAID` - The AID of the investigated Application, or `NULL`.

Returns:

The handle to the GlobalPlatform Registry entry corresponding to the input AID, or `NULL` if there is no Application in the GlobalPlatform Registry that corresponds to the input AID or if the requesting Application is not authorized to access the corresponding GlobalPlatform Registry entry.

getService

```
AID *getService(const AID *serverAID, unsigned short usServiceName);
```

This function returns the AID of a Global Services Application.

Notes:

- The `serverAID` parameter is optional: if not known by the requesting on-card entity then it is set to `NULL`;
- The `usServiceName` parameter is mandatory. When `serverAID` is `NULL` the OPEN searches the GlobalPlatform Registry for the Global Services Application corresponding to either the unique service `usServiceName` or to this entire service family. When `serverAID` is not `NULL` the OPEN searches the GlobalPlatform Registry for a Global Services Application `serverAID` where `usServiceName` either:
  - Matches exactly with a service name recorded for the Global Services Application `serverAID`; or
  - Corresponds to an entire service family recorded for the Global Services Application `serverAID`.
- If the search is successful then the OPEN returns the AID of the corresponding Global Services Application. If the search is unsuccessful then the OPEN returns `NULL`.

Parameters:

`serverAID` - The requested Global Services Application AID.

`usServiceName` - The requested service name.

Returns:

The AID of the Global Services Application or `NULL` if there is no matching Global Services Application.

`lockCard`

```
bool lockCard(void);
```

This function locks the card.

Note:

- The OPEN locates the entry of the current Application in the GlobalPlatform Registry and verifies that the Application has the Card Lock privilege.

Returns:

`true` if card locked, `false` otherwise.

`terminateCard`

```
bool terminateCard(void);
```

This function terminates the card.

Note:

- The OPEN locates the entry of the current Application in the GlobalPlatform Registry and verifies that the Application has the Card Terminate privilege.

Returns:

`true` if card terminated, `false` otherwise.

`setATRHistBytes`

```
bool setATRHistBytes(unsigned short int usOffset, unsigned char ucLength);
```

This function sets the historical bytes, for contact cards according to ISO/IEC 7816-4 and for Type A contactless cards according to ISO/IEC 14443-3. The sequence of bytes will be visible on a subsequent power-up or reset.

Notes:

- The OPEN locates the entry of the current Application in the GlobalPlatform Registry and verifies that the Application has the Card Reset privilege for the current card interface;
- The OPEN is responsible for synchronizing the length of historical bytes in Format Character T0 of the ATR.

Parameters:

`usOffset` - Offset of the historical bytes within the MULTOS *Public* memory segment.

`ucLength` - The number of historical bytes.

Returns:

`true` if historical bytes set, `false` if the Application does not have the required privilege.

`setCardContentState`

```
bool setCardContentState(unsigned char ucState);
```

This function sets the Application specific Life Cycle State of the current Application. Application specific Life Cycle States range from '07' to '80' as long as the three low order bits are set.

Notes:

- The OPEN shall reject any transition request to the Life Cycle States INSTALLED;
- The OPEN shall reject any transition request from the Life Cycle State LOCKED;
- The OPEN locates the entry of the current Application in the GlobalPlatform Registry and modifies the value of the Life Cycle State.

Parameters:

`ucState` - The Application specific state.

Returns:

`true` if the operation is successful, `false` otherwise.

See Also:

`SECURITY_DOMAIN_PERSONALIZE, APPLICATION_LOCKED`

`registerApplication`

`bool` **`registerApplication`**`(const AID *pSDIdentifier);`

This function registers a MEL Application with the Card Manager.

Note:

- An Application should invoke the `registerApplication()` function prior to invoking any other Card Manager functions. An Application need only invoke the `registerApplication()` function once during its Life Cycle;
- Unless and until an Application invokes the `registerApplication()` function the Card Manager will assume that the Application is associated with the Issuer Security Domain and does not have the Card Reset privilege.

Parameters:

`pSDIdentifier` - Identifies the Security Domain associated with the Application.

Returns:

`true` if the Application was registered successfully, `false` otherwise.

A.2.4.9 CVM

This defines the interface of a Global Services Application implementing one or more Cardholder Verification Methods. This module offers basic Cardholder Verification Method services (e.g. CVM verification, CVM state interrogation) to any of the Applications present on the card, while some of the services (e.g. unblock CVM, change CVM value) are restricted to Applications with the privilege to change the CVM values. Prior to using this interface, an Application is required to obtain a handle to the CVM services.

Constant Summary

`#define` **`CVM_FAILURE`** `-1`

The CVM value comparison failed.

```
#define CVM_SUCCESS 0
```

The CVM value comparison was successful.

```
#define FORMAT_ASCII 0x01
```

The CVM value is formatted as ASCII bytes.

```
#define FORMAT_BCD 0x02
```

The CVM value is formatted as numerical digits, coded on a nibble (four bits) and left justified.

```
#define FORMAT_HEX 0x03
```

The CVM value is formatted as hexadecimal (binary) data.

Function Summary

```
bool blockState(const CVM_HANDLE *hCVM);
```

This function sets the state of the CVM to BLOCKED.

```
unsigned char getTriesRemaining(const CVM_HANDLE *hCVM);
```

This function returns the number of tries remaining for the CVM.

```
bool isActive(const CVM_HANDLE *hCVM);
```

This function indicates whether the CVM is present and activated.

```
bool isBlocked(const CVM_HANDLE *hCVM);
```

This function indicates whether the CVM is currently BLOCKED.

```
bool isSubmitted(const CVM_HANDLE *hCVM);
```

This function indicates whether an attempt has been made to compare the CVM value.

```
bool isVerified(const CVM_HANDLE *hCVM);
```

This function indicates whether a successful comparison of the CVM value has occurred (CVM state of VALIDATED).

```
bool resetState(const CVM_HANDLE *hCVM);
```

This function resets the state of the CVM to ACTIVE.

```
bool resetAndUnblockState(const CVM_HANDLE *hCVM);
```

This function resets the state of the CVM from BLOCKED to ACTIVE.

```
bool setTryLimit(const CVM_HANDLE *hCVM, unsigned char ucTryLimit);
```

This function sets the maximum number of tries for the CVM.

```
bool update(const CVM_HANDLE *hCVM, unsigned char[] ucaBuffer, unsigned short
int usOffset, unsigned char ucLength, unsigned char ucFormat);
```

This function changes the value of the CVM.

```
unsigned short int verify(const CVM_HANDLE *hCVM, unsigned short int
usOffset, unsigned char ucLength, unsigned char ucFormat);
```

This function compares the CVM value with the value passed as a parameter.

Constant Detail

CVM_FAILURE

```
#define CVM_FAILURE -1
```

The CVM value comparison failed.

CVM_SUCCESS

```
#define CVM_SUCCESS 0
```

The CVM value comparison was successful.

FORMAT_ASCII

```
#define FORMAT_ASCII 0x01
```

The CVM value is formatted as ASCII bytes.

Note:

- If the CVM value is stored in a format other than ASCII, it is the responsibility of the interface to convert to the expected format.

FORMAT_BCD

```
#define FORMAT_BCD 0x02
```

The CVM value is formatted as numerical digits, coded on a nibble (four bits) and left justified.

Note:

- If the CVM value is stored in a format other than BCD, it is the responsibility of the interface to convert to the expected format;
- If the length of the CVM value is uneven, the right most nibble of the CVM value shall be high values ('F').

FORMAT_HEX

```
#define FORMAT_HEX 0x03
```

The CVM value is formatted as hexadecimal (binary) data.

Note:

- If the CVM value is stored in a format other than HEX, it is the responsibility of the interface to convert to the expected format.

Function Detail

isActive

```
bool isActive(const CVM_HANDLE *hCVM)
```

This function indicates whether the CVM is present and activated. If active the CVM could be in any one of the following states: ACTIVE, INVALID_SUBMISSION, VALIDATED or BLOCKED.

Parameters:

`hCVM` - The handle to the CVM services.

Returns:

`true` if the CVM state is either ACTIVE, INVALID_SUBMISSION, VALIDATED or BLOCKED, `false` otherwise).

isSubmitted

```
bool isSubmitted(const CVM_HANDLE *hCVM);
```

This function indicates whether an attempt has been made to compare the CVM value.

Note:

- This function does not differentiate whether the CVM value has been successfully verified or not i.e. CVM states of VALIDATED or INVALID_SUBMISSION.

Parameters:

`hCVM` - The handle to the CVM services.

Returns:

`true` if the CVM state is (at least) INVALID_SUBMISSION, `false` otherwise.

isVerified

```
bool isVerified(const CVM_HANDLE *hCVM);
```

This function indicates whether a successful comparison of the CVM value has occurred (CVM state of VALIDATED).

Parameters:

`hCVM` - The handle to the CVM services.

Returns:

`true` if the CVM state is VALIDATED, `false` otherwise.

isBlocked

```
bool isBlocked(const CVM_HANDLE *hCVM);
```

This function indicates whether the CVM is currently BLOCKED.

Parameters:

`hCVM` - The handle to the CVM services.

Returns:

`true` if the CVM state is BLOCKED, `false` otherwise.

getTriesRemaining

```
unsigned char getTriesRemaining(const CVM_HANDLE *hCVM)
```

This function returns the number of tries remaining for the CVM. This indicates the number of times the CVM value can be incorrectly presented prior to the CVM reaching the state of BLOCKED.

Parameters:

`hCVM` - The handle to the CVM services.

Returns:

Tries remaining.

update

```
bool update(const CVM_HANDLE *hCVM, unsigned short int usOffset, unsigned
char ucLength, unsigned char ucFormat)
```

This function changes the content of the CVM value.

Notes:

- The CVM Application verifies that the client Application has the CVM Management privilege;
- The client Application is responsible for specifying the format of the CVM value;
- The CVM Retry Counter is reset when changing the CVM value;
- The CVM state is reset to ACTIVE when changing the CVM value;
- Data presented always replaces the previous data regardless of its format or length. The CVM shall remember the format, length, and value of the CVM data. The CVM may (or may not) do editing checks on the data and reject the CVM update if the data fails the editing checks (e.g. reject data that is presented as BCD that is not numerical).

Parameters:

`hCVM` - The handle to the CVM services.

`usOffset` - The offset of the CVM value within the MULTOS *Public* memory segment.

`ucLength` - The length of the CVM value.

`ucFormat` - The format of the CVM value.

Returns:

`true` if the CVM value was changed, `false` otherwise.

`resetState`

```
bool resetState(const CVM_HANDLE *hCVM);
```

This function resets the state of the CVM to ACTIVE.

Notes:

- The state of the CVM can only be set to ACTIVE from the states INVALID_SUBMISSION or VALIDATED;
- The state of the CVM cannot be set to ACTIVE from the state BLOCKED.

Parameters:

`hCVM` - The handle to the CVM services.

Returns:

`true` if the CVM state was reset, `false` otherwise.

`blockState`

```
bool blockState(const CVM_HANDLE *hCVM);
```

This function sets the state of the CVM to BLOCKED.

Note:

- The CVM Application verifies that the client Application has the CVM Management privilege.

Parameters:

`hCVM` - The handle to the CVM services.

Returns:

`true` if the CVM state was set to BLOCKED, `false` otherwise.

`resetAndUnblockState`

```
bool resetAndUnblockState(const CVM_HANDLE *hCVM);
```

This function resets the state of the CVM from BLOCKED to ACTIVE.

Notes:

- The CVM Application verifies that the client Application has the CVM Management privilege;
- The CVM Retry Counter is reset when unblocking the CVM.

Parameters:

`hCVM` - The handle to the CVM services.

Returns:

`true` if the CVM state was reset to ACTIVE, `false` otherwise.

`setTryLimit`

```
bool setTryLimit(const CVM_HANDLE *hCVM, unsigned char ucTryLimit)
```

This function sets the Retry Limit for the CVM.

Notes:

- The CVM Application verifies that the client Application has the CVM Management privilege;
- The CVM Retry Counter is reset when setting the maximum number of tries;
- The CVM state is reset to ACTIVE when setting the maximum number of tries.

Parameters:

`hCVM` - The handle to the CVM services.

`ucTryLimit` - The Retry Limit for the CVM.

Returns:

`true` if the Retry Limit was set, `false` otherwise.

`verify`

```
unsigned short int verify(const CVM_HANDLE *hCVM, unsigned short int
usOffset, unsigned char ucLength, unsigned char ucFormat)
```

This function compares the stored CVM value with the CVM value passed as a parameter.

Notes:

- If the value passed as a parameter is not in the same format as the CVM is stored, the value passed as a parameter must be converted prior to comparing;
- If HEX format is presented for CVM verification and ASCII or BCD format was used for updating the CVM value, the comparison fails;
- If HEX format is presented for CVM verification and HEX format was used for updating the CVM value, the comparison succeeds when the length and the data value match exactly;
- If BCD or ASCII format is presented for CVM verification and HEX format was used for updating the CVM value, the comparison fails;

- If ASCII format is presented for CVM verification and BCD format was used for updating the CVM value, the comparison fails if the ASCII characters presented for verification are not all numerical (zero to nine). If all the ASCII characters are numerical, format conversion occurs and the comparison succeeds when the length and the data value match exactly;
- If BCD format is presented for CVM verification and ASCII format was used for updating the CVM value, the comparison fails if the CVM value contains non-numerical ASCII characters. If the CVM value contains only numerical ASCII characters, format conversion occurs and the comparison succeeds when the length and the data value match exactly;
- If the comparison is successful, the Retry Counter must be reset and the CVM state must be set to VALIDATED;
- If the comparison is unsuccessful, the Retry Counter must be updated and the CVM state must be set to INVALID_SUBMISSION;
- The Retry Counter object and the CVM states VALIDATED and INVALID_SUBMISSION shall not conform to a transaction in progress, i.e. they shall not revert to a previous value if a transaction in progress is aborted;
- If the maximum number of tries has been reached, the CVM state must be set to BLOCKED.

Parameters:

`hCVM` - The handle to the CVM services.

`usOffset` – The offset of the CVM value within the MULTOS *Public* memory segment.

`ucLength` - The length of the CVM value.

`ucFormat` - The format of the CVM value.

Returns:

Value indicating whether the comparison was successful or not. Values other than `CVM_SUCCESS` or `CVM_FAILURE` are Reserved for Future Use.

A.2.4.10 GPRegistryEntry

Constant Summary

`#define` **`PRIVILEGE_AUTHORIZED_MANAGEMENT 0x09`**

Privilege number for Authorized Management.

`#define` **`PRIVILEGE_CARD_LOCK 0x03`**

Privilege number for Card Lock.

`#define` **`PRIVILEGE_CARD_RESET 0x05`**

Privilege number for Card Reset.

`#define` **`PRIVILEGE_CARD_TERMINATE 0x04`**

Privilege number for Card Terminate.

`#define` **`PRIVILEGE_CVM_MANAGEMENT 0x06`**

Privilege number for CVM Management.

`#define` **`PRIVILEGE_DAP_VERIFICATION 0x01`**

Privilege number for DAP Verification.

```
#define PRIVILEGE_DELEGATED_MANAGEMENT 0x02
```

Privilege number for Delegated Management.

```
#define PRIVILEGE_FINAL_APPLICATION 0x0E
```

Privilege number for Final Application.

```
#define PRIVILEGE_GLOBAL_DELETE 0x0B
```

Privilege number for Global Delete.

```
#define PRIVILEGE_GLOBAL_LOCK 0x0C
```

Privilege number for Global Lock.

```
#define PRIVILEGE_GLOBAL_REGISTRY 0x0D
```

Privilege number for Global Registry.

```
#define PRIVILEGE_GLOBAL_SERVICE 0x0F
```

Privilege number for Global Service.

```
#define PRIVILEGE_MANDATED_DAP 0x07
```

Privilege number for Mandated DAP verification.

```
#define PRIVILEGE_RECEIPT_GENERATION 0x10
```

Privilege number for Receipt Generation.

```
#define PRIVILEGE_SECURITY_DOMAIN  0x00
```

Privilege number for application is a Security Domain.

```
#define PRIVILEGE_TOKEN_VERIFICATION 0x0A
```

Privilege number for Token Verification.

```
#define PRIVILEGE_TRUSTED_PATH 0x08
```

Privilege number for Trusted Path.

Function Summary

```
void deregisterService(GPRE_HANDLE hGPRE, unsigned short int usServiceName);
```

This function deregisters a service name.

```
AID *getAID(GPRE_HANDLE hGPRE);
```

This function provides the AID of the GlobalPlatform Registry entry.

```
unsigned short int getPrivileges(GPRE_HANDLE hGPRE,unsigned char *ucBuffer,
unsigned short int usOffset);
```

This function retrieves the Privileges for the GlobalPlatform Registry entry.

```
unsigned char getState(GPRE_HANDLE hGPRE);
```

This function retrieves the Life Cycle State for the GlobalPlatform Registry entry.

```
bool isAssociated(GPRE_HANDLE hGPRE, const AID *SDAID);
```

This function checks the association between an Application and a Security Domain.

```
bool isPrivileged(GPRE_HANDLE hGPRE, unsigned char ucPrivilege);
```

This function checks the privilege of an Application.

```
void registerService(GPRE_HANDLE hGPRE, unsigned short int usServiceName);
```

This function registers a unique service name.

```
boolean setState(GPRE_HANDLE hGPRE, unsigned char ucState);
```

This method sets the Life Cycle state of an Application.

Constant Detail

PRIVILEGE_AUTHORIZED_MANAGEMENT

```
#define PRIVILEGE_AUTHORIZED_MANAGEMENT 0x09
```

Privilege number for Authorized Management.

PRIVILEGE_CARD_LOCK

```
#define PRIVILEGE_CARD_LOCK 0x03
```

Privilege number for Card Lock.

PRIVILEGE_CARD_RESET

```
#define PRIVILEGE_CARD_RESET 0x05
```

Privilege number for Card Reset.

PRIVILEGE_CARD_TERMINATE

```
#define PRIVILEGE_CARD_TERMINATE 0x04
```

Privilege number for Card Terminate.

PRIVILEGE_CVM_MANAGEMENT

```
#define PRIVILEGE_CVM_MANAGEMENT 0x06
```

Privilege number for CVM Management.

PRIVILEGE_DAP_VERIFICATION

```
#define PRIVILEGE_DAP_VERIFICATION 0x01
```

Privilege number for DAP Verification.

PRIVILEGE_DELEGATED_MANAGEMENT

```
#define PRIVILEGE_DELEGATED_MANAGEMENT 0x02
```

Privilege number for Delegated Management.

PRIVILEGE_FINAL_APPLICATION

```
#define PRIVILEGE_FINAL_APPLICATION 0x0E
```

Privilege number for Final Application.

PRIVILEGE_GLOBAL_DELETE

```
#define PRIVILEGE_GLOBAL_DELETE 0x0B
```

Privilege number for Global Delete.

PRIVILEGE_GLOBAL_LOCK

```
#define PRIVILEGE_GLOBAL_LOCK 0x0C
```

Privilege number for Global Lock.

PRIVILEGE_GLOBAL_REGISTRY

```
#define PRIVILEGE_GLOBAL_REGISTRY 0x0D
```

Privilege number for Global Registry.

PRIVILEGE_GLOBAL_SERVICE

```
#define PRIVILEGE_GLOBAL_SERVICE 0x0F
```

Privilege number for Global Service.

PRIVILEGE_MANDATED_DAP

```
#define PRIVILEGE_MANDATED_DAP 0x07
```

Privilege number for Mandated DAP verification.

PRIVILEGE_RECEIPT_GENERATION

```
#define PRIVILEGE_RECEIPT_GENERATION 0x10
```

Privilege number for Receipt Generation.

PRIVILEGE_SECURITY_DOMAIN

```
#define PRIVILEGE_SECURITY_DOMAIN  0x00
```

Privilege number for application is a Security Domain.

PRIVILEGE_TOKEN_VERIFICATION

```
#define PRIVILEGE_TOKEN_VERIFICATION 0x0A
```

Privilege number for Token Verification.

PRIVILEGE_TRUSTED_PATH

```
#define PRIVILEGE_TRUSTED_PATH 0x08
```

Privilege number for Trusted Path.

Function Detail

deregisterService

```
void deregisterService(GPRE_HANDLE hGPRE, unsigned short int usServiceName);
```

This function allows a Global Services Application (e.g. a CVM Application) to deregister a service name.

Notes:

- The OPEN checks that the requesting on-card entity has the Global Service Privilege and is associated with this registry entry;
- The OPEN checks that the service name is registered as unique for the requesting on-card entity.

Parameters:

`hGPRE` - The handle to the GlobalPlatform Registry entry.

`usServiceName` - The unique service name to deregister.

`SCode()`:

May return the following status code:

- SW_CONDITIONS_NOT_SATISFIED

`getAID`

```
AID *getAID(GPRE_HANDLE hGPRE);
```

This function returns the Application's AID registered in the current GlobalPlatform Registry's entry.

Parameters:

`hGPRE` - The handle to the GlobalPlatform Registry entry.

Returns:

The Application's AID.

`getPrivileges`

```
unsigned short int getPrivileges(GPRE_HANDLE hGPRE, unsigned char *ucBuffer,
unsigned short int usOffset);
```

This function returns all the Privileges bytes registered in the current GlobalPlatform Registry entry.

Parameters:

`hGPRE` - The handle to the GlobalPlatform Registry entry.

`ucBuffer` - The byte array where Privileges bytes are to be stored.

`usOffset` - The offset in `ucBuffer` at which to begin the Privileges bytes.

Returns:

The offset in `ucBuffer` + Length of the Privileges where the Privileges bytes end.

`getState`

```
unsigned char getState(GPRE_HANDLE hGPRE);
```

This function returns the Life Cycle State registered in the current GlobalPlatform Registry entry.

Parameters:

`hGPRE` - The handle to the GlobalPlatform Registry entry.

Returns:

The Life Cycle State as defined in section 11.1.1.

`isAssociated`

```
bool isAssociated(GPRE_HANDLE hGPRE, const AID *SDAID);
```

This function verifies wether the Application whose AID is provided in the input parameters is the Security Domain associated with the entity whose Global Platform Registry entry is provided in the input parameters.

Notes:

- The OPEN determines if `SDAID` is registered in the current GlobalPlatform Registry's entry as the associated Security Domain.

Parameters:

`hGPRE` - The handle to the GlobalPlatform Registry entry.

`SDAID` - The AID object of the investigated Security Domain.

Returns:

`true` if the GlobalPlatform Registry references an association with the Security Domain, `false` otherwise

`isPrivileged`

```
bool isPrivileged(GPRE_HANDLE hGPRE, unsigned char ucPrivilege);
```

This function allows an Application (e.g. a CVM Application) to verify if a given Privilege is registered in the current GlobalPlatform Registry entry of an Application (e.g. check the CVM Management privilege of another Application invoking the `CVM.update()` function).

Parameters:

`hGPRE` - The handle to the GlobalPlatform Registry entry.

`ucPrivilege` - The privilege number to verify, as defined in Table 6-1.

Returns:

`true` if at least the referenced Privilege is registered in the GlobalPlatform Registry entry, `false` if the referenced Privilege is not registered in the GP Registry entry.

`registerService`

```
void registerService(GPRE_HANDLE hGPRE, unsigned short int usServiceName);
```

This function allows a Global Services Application (e.g. a CVM Application) to register a unique service name.

Notes:

- The OPEN checks that the requesting on-card entity has the Global Service Privilege and is associated with the current GlobalPlatform Registry entry;
- The OPEN checks that the requested service name matches with (one of) the Service Parameter(s) recorded in the current GlobalPlatform Registry entry;
- The OPEN checks that the service name is not already registered as unique by any other entry in the GlobalPlatform Registry.

Parameters:

`hGPRE` - The handle to the GlobalPlatform Registry entry.

`usServiceName` - The unique service name to register.

`SCode()`:

May return one of the following status codes (other security mechanism related status codes may be returned):

- SW_CONDITIONS_NOT_SATISFIED

setState

```
bool setState(GPRE_HANDLE hGPRE, unsigned char ucState)
```

This method allows the Life Cycle state of this GPRegistryEntry to be transitioned to the requested target state.

Notes:

- A transition request to the Life Cycle State INSTALLED shall be rejected;
- A transition request to Life Cycle state other than APPLICATION_LOCKED and APPLICATION_UNLOCKED shall be accepted only if the invoking Application corresponds to this GPRegistryEntry;
- An Application shall be able to lock and shall not be able to unlock itself;
- Only an Application with Global Lock privilege or the directly or indirectly associated Security Domain of this GPRegistryEntry shall be able to lock or unlock this GPRegistry Entry;
- This method shall fail if this GPRegistryEntry corresponds to the Issuer Security Domain.

Parameters:

`hGPRE` - The handle to the GlobalPlatform Registry Entry

`ucState` - the target state for this GlobalPlatform Registry Entry

Returns:

`true` if the transition is successful, or

`false` otherwise.

A.2.4.11 GlobalService

This defines the interface for requesting a Global Services Application to provide its actual service interface. The Global Services Application uses this interface to check the validity of the request presented by an on-card entity.

Constant Summary

```
#define KEY_ACCESS_ANY -1
```

Key access indicating key may be used by the Security Domain and any associated application.

```
#define KEY_ACCESS_SECURITY_DOMAIN 0x01
```

Key access indicating key may be used by the Security Domain but not by any associated application .

```
#define KEY_ACCESS_APPLICATION 0x02
```

Key access indicating key may be used by any associated application but not by the Security Domain.

```
#define KEY_TYPE_AES 0x88
```

Key type indicating AES.

```
#define KEY_TYPE_3DES 0x81
```

Key type indicating Triple DES reserved for specific implementations.

```
#define KEY_TYPE_3DES_CBC 0x82
```

Key type indicating Triple DES in CBC mode.

```
#define KEY_TYPE_DES 0x80
```

Key type indicating DES with ECB/CBC implicitly known.

```
#define KEY_TYPE_DES_CBC 0x84
```

Key type indicating DES in CBC mode.

```
#define KEY_TYPE_DES_ECB 0x83
```

Key type indicating DES in ECB mode.

```
#define KEY_TYPE_EXTENDED 0xFF
```

Key type indicating extended key format.

```
#define KEY_TYPE_HMAC_SHA1 0x90
```

Key type indicating HMAC SHA1, length of HMAC implicitly known.

```
#define KEY_TYPE_HMAC_SHA1_160 0x91
```

Key type indicating HMAC SHA1, length of HMAC is 160 bits.

```
#define KEY_TYPE_RSA_PRIVATE_CRT_P 0xA4
```

Key type indicating RSA Private Key Chinese Remainder p component.

```
#define KEY_TYPE_RSA_PRIVATE_CRT_PQ 0xA6
```

Key type indicating RSA Private Key Chinese Remainder pq component.

```
#define KEY_TYPE_RSA_PRIVATE_CRT_Q 0xA5
```

Key type indicating RSA Private Key Chinese Remainder q component.

```
#define KEY_TYPE_RSA_PRIVATE_CRT_DP1 0xA7
```

Key type indicating RSA Private Key Chinese Remainder dp1 component.

```
#define KEY_TYPE_RSA_PRIVATE_CRT_DQ1 0xA8
```

Key type indicating RSA Private Key Chinese Remainder dq1 component.

```
#define KEY_TYPE_RSA_PRIVATE_CRT_DQ1 0xA8
```

Key type indicating RSA Private Key Chinese Remainder dq1 component.

```
#define KEY_TYPE_RSA_PRIVATE_EXPONENT 0xA3
```

Key type indicating RSA Private Key exponent .

```
#define KEY_TYPE_RSA_PRIVATE_MODULUS 0xA2
```

Key type indicating RSA Public Key modulus.

```
#define KEY_USAGE_COMPUTATION_DECIPHERMENT 0x40
```

Key usage indicating computation and decipherment.

```
#define KEY_USAGE_CONFIDENTIALITY 0x08
```

Key usage indicating sensitive data confidentiality .

```
#define KEY_USAGE_CRYPTOGRAPHIC_AUTHORIZATION 0x01
```

Key usage indicating cryptographic authorization.

```
#define KEY_USAGE_CRYPTOGRAPHIC_CHECKSUM 0x04
```

Key usage indicating cryptographic checksum e.g. MAC.

```
#define KEY_USAGE_DIGITAL_SIGNATURE 0x02
```

Key usage indicating Digital Signature.

```
#define KEY_USAGE_SM_COMMAND 0x10
```

Key usage indicating Secure Messaging in command data field.

```
#define KEY_USAGE_SM_RESPONSE 0x20
```

Key usage indicating Secure Messaging in response data field.

```
#define KEY_USAGE_VERIFICATION_ENCIPHERMENT 0x80
```

Key usage indicating verification and encipherment.

Function Summary

```
SERVICE_HANDLE getServiceInterface(GPRE_HANDLE hClientRegistryEntry, unsigned
short usServiceName, unsigned short usOffset, unsigned short usLength);
```

This function returns a handle to the requested service interface.

Constant Detail

KEY_ACCESS_ANY

```
#define KEY_ACCESS_ANY -1
```

Key access indicating key may be used by the Security Domain and any associated application.

KEY_ACCESS_SECURITY_DOMAIN

```
#define KEY_ACCESS_SECURITY_DOMAIN 0x01
```

Key access indicating key may be used by the Security Domain but not by any associated application .

KEY_ACCESS_APPLICATION

```
#define KEY_ACCESS_APPLICATION 0x02
```

Key access indicating key may be used by any associated application but not by the Security Domain.

KEY_TYPE_AES

```
#define KEY_TYPE_AES 0x88
```

Key type indicating AES.

KEY_TYPE_3DES

```
#define KEY_TYPE_3DES 0x81
```

Key type indicating Triple DES reserved for specific implementations.

KEY_TYPE_3DES_CBC

```
#define KEY_TYPE_3DES_CBC 0x82
```

Key type indicating Triple DES in CBC mode.

KEY_TYPE_DES

```
#define KEY_TYPE_DES 0x80
```

Key type indicating DES with ECB/CBC implicitly known.

KEY_TYPE_DES_CBC

```
#define KEY_TYPE_DES_CBC 0x84
```

Key type indicating DES in CBC mode.

KEY_TYPE_DES_ECB

```
#define KEY_TYPE_DES_ECB 0x83
```

Key type indicating DES in ECB mode.

KEY_TYPE_EXTENDED

```
#define KEY_TYPE_EXTENDED 0xFF
```

Key type indicating extended key format.

KEY_TYPE_HMAC_SHA1

```
#define KEY_TYPE_HMAC_SHA1 0x90
```

Key type indicating HMAC SHA1, length of HMAC implicitly known.

KEY_TYPE_HMAC_SHA1_160

```
#define KEY_TYPE_HMAC_SHA1_160 0x91
```

Key type indicating HMAC SHA1, length of HMAC is 160 bits.

KEY_TYPE_RSA_PRIVATE_CRT_P

```
#define KEY_TYPE_RSA_PRIVATE_CRT_P 0xA4
```

Key type indicating RSA Private Key Chinese Remainder p component.

KEY_TYPE_RSA_PRIVATE_CRT_PQ

```
#define KEY_TYPE_RSA_PRIVATE_CRT_PQ 0xA6
```

Key type indicating RSA Private Key Chinese Remainder pq component.

KEY_TYPE_RSA_PRIVATE_CRT_Q

```
#define KEY_TYPE_RSA_PRIVATE_CRT_Q 0xA5
```

Key type indicating RSA Private Key Chinese Remainder pq component.

KEY_TYPE_RSA_PRIVATE_CRT_DP1

```
#define KEY_TYPE_RSA_PRIVATE_CRT_DP1 0xA7
```

Key type indicating RSA Private Key Chinese Remainder dp1 component.

KEY_TYPE_RSA_PRIVATE_CRT_DQ1

```
#define KEY_TYPE_RSA_PRIVATE_CRT_DQ1 0xA8
```

Key type indicating RSA Private Key Chinese Remainder dq1 component.

KEY_TYPE_RSA_PRIVATE_CRT_DQ1

```
#define KEY_TYPE_RSA_PRIVATE_CRT_DQ1 0xA8
```

Key type indicating RSA Private Key Chinese Remainder dq1 component.

KEY_TYPE_RSA_PRIVATE_EXPONENT

```
#define KEY_TYPE_RSA_PRIVATE_EXPONENT 0xA3
```

Key type indicating RSA Public Key exponent .

KEY_TYPE_RSA_PRIVATE_MODULUS

```
#define KEY_TYPE_RSA_PRIVATE_MODULUS 0xA2
```

Key type indicating RSA Public Key modulus.

KEY_USAGE_COMPUTATION_DECIPHERMENT

```
#define KEY_USAGE_COMPUTATION_DECIPHERMENT 0x40
```

Key usage indicating computation and decipherment.

KEY_USAGE_CONFIDENTIALITY

```
#define KEY_USAGE_CONFIDENTIALITY 0x08
```

Key usage indicating sensitive data confidentiality .

KEY_USAGE_CRYPTOGRAPHIC_AUTHORIZATION

```
#define KEY_USAGE_CRYPTOGRAPHIC_AUTHORIZATION 0x01
```

Key usage indicating cryptographic authorization.

KEY_USAGE_CRYPTOGRAPHIC_CHECKSUM

```
#define KEY_USAGE_CRYPTOGRAPHIC_CHECKSUM 0x04
```

Key usage indicating cryptographic checksum e.g. MAC.

KEY_USAGE_DIGITAL_SIGNATURE

```
#define KEY_USAGE_DIGITAL_SIGNATURE 0x02
```

Key usage indicating Digital Signature.

KEY_USAGE_SM_COMMAND

```
#define KEY_USAGE_SM_COMMAND 0x10
```

Key usage indicating Secure Messaging in command data field.

KEY_USAGE_SM_RESPONSE

```
#define KEY_USAGE_SM_RESPONSE 0x20
```

Key usage indicating Secure Messaging in response data field.

KEY_USAGE_VERIFICATION_ENCIPHERMENT

```
#define KEY_USAGE_VERIFICATION_ENCIPHERMENT 0x80
```

Key usage indicating verification and encipherment.

Function Detail

getServiceInterface

```
SERVICE_HANDLE getServiceInterface(GPRE_HANDLE hClientRegistryEntry, unsigned short usServiceName, unsigned short usOffset, unsigned short usLength);
```

This function returns a handle to the requested service interface of a Global Services Application.

Notes:

- This function is implemented by each Global Services Application, not by the OPEN. However, the function is specified within this specification in order to allow each Global Services Application to present a consistent interface to client applications;

- Client applications should invoke this function using a similar mechanism to that described in section A.2.3.3, except that the MULTOS *Delegate* primitive is invoked specifying the AID of the Global Services Application instead of the AID of the "Card Manager Services Provider" application;
- The Global Services Application verifies the validity of the request according to its own security policies for this `usServiceName` and based on the identity of the requesting on-card entity and its characteristics as registered in the GlobalPlatform Registry entry identified by the handle `hClientRegistryEntry`;
- The Global Services Application may reject an invalid service request and return `NULL`.

Parameters:

`hClientRegistryEntry` - The handle to the GlobalPlatform Registry entry of the requesting on-card entity.

`usServiceName` - The requested service name.

`usOffset` - The offset of the additional parameters of the service request within the MULTOS *Public* memory segment.

`usLength` - Length of the additional parameters of the service request.

Returns:

The specific service interface reference or `NULL`.

`SCode()`:

May return one of the following status codes (other security mechanism related status codes may be returned):

- SW_CONDITIONS_NOT_SATISFIED
- SW_SECURITY_STATUS_NOT_SATISFIED

B Algorithms (Cryptographic and Hashing)

A GlobalPlatform card may support many types of security functions for use by applications. This appendix contains examples of several of the cryptographic algorithms and the hashing method that are possible for GlobalPlatform.

B.1 Data Encryption Standard (DES)

The Data Encryption Standard (DES) is a symmetric cryptographic algorithm that requires the use of the same secret key to encrypt and decrypt data. In its simplest form it uses an 8-byte key to encrypt an 8-byte block of data and the same 8-byte key to decrypt and retrieve the original clear text.

Triple DES uses a compound operation of DES encryption and decryption. Triple DES as used in this specification uses keying option 2 as defined in ISO/IEC 18033-3.

B.1.1 Encryption/Decryption

For encryption two variants are defined.

B.1.1.1 CBC Mode

Triple DES in CBC mode, as defined in [ANSI X9.52] and [ISO 10116], is used with an Initial Chaining Value equal to '00 00 00 00 00 00 00 00'.

B.1.1.2 ECB Mode

Triple DES in ECB mode, as defined in [ANSI X9.52] and [ISO 10116], is used.

B.1.2 MACing

The chaining data encryption methods are defined in [ISO 9797].

B.1.2.1 Full Triple DES MAC

The full triple DES MAC is as defined in [ISO 9797-1] as MAC Algorithm 3 with output transformation 3, without truncation, and with triple DES taking the place of the block cipher.

B.1.2.2 Single DES Plus Final Triple DES MAC

This is also known as the Retail MAC. It is as defined in [ISO 9797-1] as MAC Algorithm 3 with output transformation 3, without truncation, and with DES taking the place of the block cipher.

B.2 Hashing Algorithms

A hash is a one-way digest operation over an arbitrary length of data that returns a fixed length hash value. A hash is not a cryptographic algorithm and it only provides integrity of the data. It does not provide authentication or confidentiality.

B.2.1 Secure Hash Algorithm (SHA-1)

SHA-1 is defined in [ISO 10118-3] and FIPS PUB 180-1.

B.2.2 MULTOS Asymmetric Hash Algorithm

The asymmetric hash algorithm for GlobalPlatform on MULTOS is described in MULTOS document [MAO-DOC-REF-009].

B.3 Public Key Cryptography Scheme 1 (PKCS#1)

Unlike DES, which uses a shared secret key, public key cryptography employs the use of a Private Key (kept secret by one entity) and a Public Key. One public key algorithm used for GlobalPlatform is RSA (Rivest / Shamir / Adleman).

The process of generating an RSA signature involves using the PKCS#1 standard. The signature scheme is RSA SSA-PKCS1-v1_5 as defined in PKCS#1. This is applied to a digest of the data being signed, generated by the SHA-1 algorithm. The resultant signature is the same size as the public key modulus.

The process of verifying a signature uses the public key applied to the signature (providing the hash) and the comparison of this hash with the hash of the data being verified.

B.4 DES Padding

Unless specified to the contrary, padding prior to performing a DES operation across a block of data is achieved in the following manner:

- Append an '80' to the right of the data block;
- If the resultant data block length is a multiple of 8 bytes, no further padding is required;
- Append binary zeroes to the right of the data block until the data block length is a multiple of 8 bytes.

If the data is being padded with the intention of generating a MAC, the padding is discarded following the DES operation.

C Secure Content Management

This appendix defines the different methods that shall be used to secure the changes and/or modifications to the content of a GlobalPlatform card. At this point in time, these methods are the only ones specified for GlobalPlatform cards. Depending on the implementation of the card, any subset of these methods may be supported.

C.1 Keys

In order to perform the Card Content management operations described in the subsequent sub-sections different keys are required.

C.1.1 Token and Receipt Keys

The Security Domain with Token Verification privilege and the Security Domain with Receipt Generation privilege shall have knowledge of different keys if Delegated Management is supported. These keys are used to secure the management of the card and its content.

Key	Usage	Minimum Length	Remark
Token	Token verification (RSA Public Key)	1024 bits	Delegated Management only
Receipt	Optional Receipt generation (DES)	16 bytes	Delegated Management only
Receipt	Optional Receipt generation (RSA)	1024 bits	Delegated Management only

Table C-1: Token and Receipt Keys

It is recommended that RSA keys are a multiple of 32 bits in length above the minimum length.

C.1.1.1 Token key

If the card supports Delegated Management, the Security Domain with Token Verification privilege shall have an RSA Public Key to be used to verify a Token. Token verification and the corresponding delegated management operation shall fail if the Token verification key is not present.

C.1.1.2 Receipt Key

If the card supports Delegated Management and specifically Receipt generation, the Security Domain with Receipt Generation privilege shall either have an unambiguously recognized DES key or private RSA key to be used to generate Receipts. Receipt generation and the corresponding delegated management operation shall fail if the Receipt generation key is not present.

C.1.2 DAP Verification Keys

If the Security Domain supports DAP Verification, the Security Domain shall have either an RSA public key or a DES key to verify Load File Data Block Signatures.

Key	Usage	Minimum Length	Remark
DAP Verification	Load File Data Block Signature verification (RSA Public Key)	1024 bits	DAP Verification only

Key	Usage	Minimum Length	Remark
DAP Verification	Load File Data Block Signature verification (DES Key)	16 Bytes	DAP Verification only

Table C-2: Additional Security Domain Key

It is recommended that RSA keys are a multiple of 32 bits in length above the minimum length.

C.2 Load File Data Block Hash

The Load File Data Block Hash is a field that is required to be present in the INSTALL [for load] command if Delegated Management loading is occurring and/or the Load File contains one or more DAP Blocks. The purpose of this field is to provide a SHA-1 digest of the Load File Data Block and it is used to ensure that the contents of the Load File Data Block have not been modified in any way. While the actual hash is not secured in any manner, it is used in other secure operations that ensure that the Load File Data Block has not been modified and a new hash generated for the modified data. The OPEN shall verify the integrity of the Load File Data Block prior to creating an Executable Load File.

Figure C-1 details the Load File Data Block Hash calculation performed by the Security Domain, an Application Provider and a Controlling Authority.

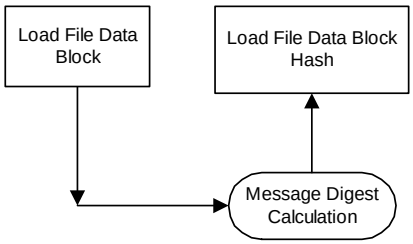

Figure C-1: Load File Data Block Hash Calculation

Padding of the data is as defined by the SHA-1. The digest is generated prior to the Load File Data Block being encapsulated in the TLV structure of the Load File (i.e. excluding tag 'C4' and its length).

When Delegated Management is occurring the Load Token is a signature of multiple fields including this hash and is proof that the Card Issuer generated the Token for the Load File Data Block linked to the hash.

If the Load File contains DAP Blocks, DAP Verification is the actual signing of this hash and is proof that the DAP Block was generated by an entity that verified the content of the Load File Data Block linked to the hash.

C.3 Load File Data Block Signature (DAP Verification)

The Load File Data Block Signature provides verification of the Load File Data Block prior to commencing with the processing of the actual Load File Data Block. The OPEN shall request the Security Domain linked to the Load File Data Block Signature to verify the signature.

The Load File Data Block Signature is a signature of the Load File Data Block Hash. Each Load File Data Block Signature is combined with its linked Security Domain AID in the TLV structured DAP Block. DAP Blocks are positioned in the beginning of the Load File.

Figure C-2 details the Load File Data Block Signature calculation performed by an Application Provider or Controlling Authority.

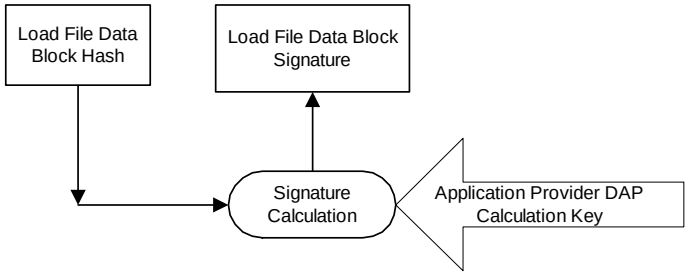

Figure C-2: Load File Data Block Signature Calculation

C.4 Tokens

Tokens are public key signatures, generated by the Card Issuer. Tokens are the proof that the Card Issuer has authorized the Card Content Management operation being performed and they are generated and verified according to this appendix. Tokens may be generated for use on multiple cards, depending on the Card Issuer's security policy.

A token is an RSA signature, being the decryption (with the token private key) of the SHA-1 message digest of the appropriate data. Padding of the data is as defined by the SHA-1 and PKCS#1 mechanisms. The signature scheme for tokens is as defined in appendix B.2.1 - *Secure Hash Algorithm (SHA-1)*.

C.4.1 Load Token

The Load Token is a signature authorizing the transmission of application code to the card. It allows for the verification of the load request. The OPEN shall request the Security Domain with Token Verification privilege to verify the Load Token.

The following figure shows the Load Token generation.

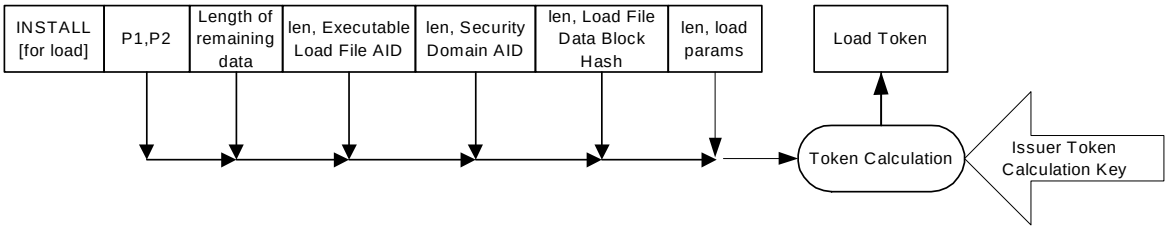

Figure C-3 Load Token Calculation

Generating a Load Token ensures the following:

- Only the application code (hash of the Load File Data Block) included in this signature may be loaded to the card;
- The AID of the Load File Data Block may only be that included in the signature;
- The Executable Load File (derived from the Load File Data Block) and all Executable Modules within the Executable Load File may only be associated with the Security Domain included in the signature;
- The issuer of the Load Token may only be the Security Domain Provider of the Security Domain with Token Verification Privilege (e.g. the Card Issuer).

The Load Token is an RSA signature of the following data that will be included in the actual INSTALL [for load] command to be transmitted to the card.

Name	Length
Reference control parameter P1	1
Reference control parameter P2	1
Length of the following data fields (including the length fields) - see note below	1
Load File AID length	1
Load File AID	5-16
Security Domain AID length	1
Security Domain AID	0 or 5-16
Length of the Load File Data Block Hash	1
Load File Data Block Hash	20
Load parameters field length	1-2
Load parameters field	Var

Table C-3: Data Elements Included in the Load Token

Note that 'Length of the following data fields' is the value of Lc of the INSTALL command, prior to a Token being added. It is coded on 1 byte with a value up to 255.

C.4.2 Install Token

The Install Token allows for the verification of the installation process. The token is a signature authorizing the installation to the card of an Application (Executable Module) contained in a previously loaded file (Executable Load File). The OPEN shall request the Security Domain with Token Verification privilege to verify the Install Token.

Figure C-4 shows the Install Token generation.

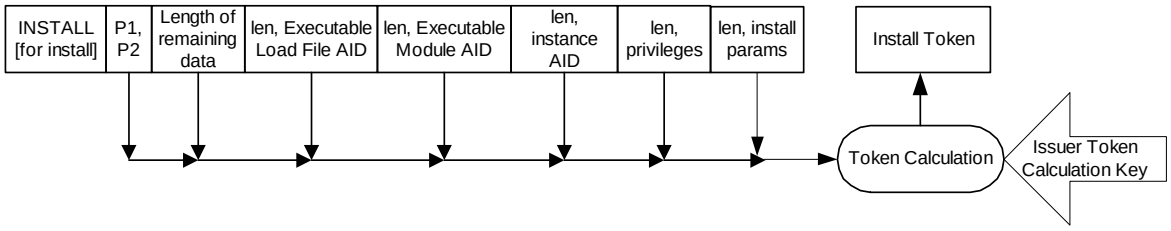

Figure C-4: Install Token Calculation

Generating an Install Token ensures the following:

- Only the application (Executable Module) included in this signature and present in the Executable Load File may be installed on the card;
- That only the instance AID included in this signature may be used as the AID to select the instance;
- The Application may only be installed with the privileges included in this signature;
- Only the parameters included in this signature may be used to install the Application;
- The issuer of the Install Token may only be the Security Domain Provider of the Security Domain with Token Verification Privilege (e.g. the Card Issuer).

The Install Token is an RSA signature of the following data that will be included in an INSTALL [for install] command.

Name	Length
Reference control parameter P1	1
Reference control parameter P2	1
Length of the following data fields (including the length fields) - see note below	1
Executable Load File AID length	1
Executable Load File AID	5-16
Executable Module AID length	1
AID within the Executable Load File	5-16
Instance AID length	1
Instance AID	5-16
Privileges length	1
Privileges (byte 1 - byte 2 – byte 3)	1 or 3
Install Parameters field length	1-2
Install Parameters (system and Application) field	Var

Table C-4: Data Elements Included in the Install Token

Note that 'Length of the following data fields' is the value of Lc of the INSTALL command, prior to a Token being added. It is coded on 1 byte with a value up to 255.

C.4.3 Make Selectable Token

The Make Selectable Token allows for the verification of the installation for make selectable process. The token is a signature authorizing the making selectable of an Application. The OPEN shall request the Security Domain with Token Verification privilege to verify the Make Selectable Token.

The following figure shows the Make Selectable Token generation.

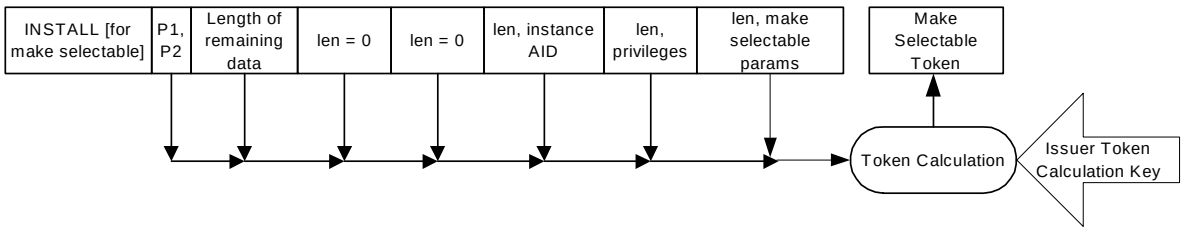

Figure C-5: Make Selectable Token Calculation

Generating a Make Selectable Token ensures the following:

- Only the Application included in this signature and present in the Executable Load File may be made selectable on the card;
- That only the instance AID included in this signature may be used as the AID to select the instance;
- The Application may only be made selectable with the privileges included in this signature;
- Only the parameters included in this signature may be used to make the Application selectable;
- The issuer of the Make Selectable Token may only be the Security Domain Provider of the Security Domain with Token Verification Privilege (e.g. the Card Issuer).

The Make Selectable Token is an RSA signature of the following data that will be included in an INSTALL [for make selectable] command.

Name	Length
Reference control parameter P1	1
Reference control parameter P2	1
Length of the following data fields (including the length fields) - see note below	1
Executable Load File AID length (= '00')	1
Executable Module AID length (= '00')	1
Instance AID length	1
Instance AID	5-16
Privileges length	1
Privileges (byte 1 - byte 2 - byte 3)	1 or 3
Make Selectable parameters field length	1
Make Selectable parameters field	0-n

Table C-5: Data Elements Included in the Make Selectable Token

Note that 'Length of the following data fields' is the value of Lc of the INSTALL command, prior to a Token being added. It is coded on 1 byte with a value up to 255.

C.4.4 Extradition Token

The Extradition Token allows for the verification of the extradition process. The token is a signature authorizing the extradition of an Application from one Security Domain to another. The OPEN shall request the Security Domain with Token Verification privilege to verify the Extradition Token.

The following figure shows Extradition Token generation.

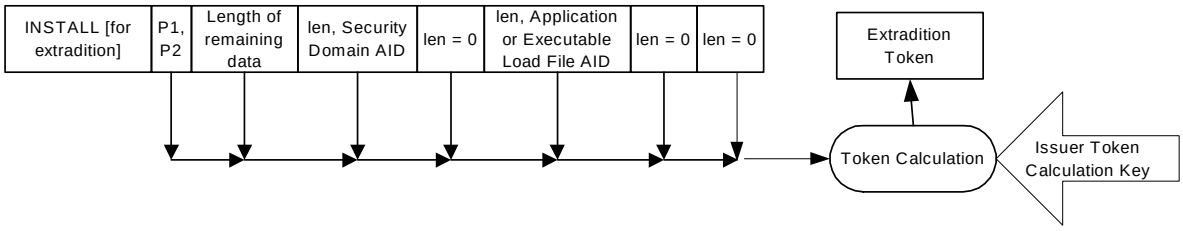

Figure C-6: Extradition Token Calculation

Generating an Extradition Token ensures the following:

- Only the Application or Executable Load File included in this signature may be extradited;
- The issuer of the Extradition Token may only be the Security Domain Provider of the Security Domain with Token Verification Privilege (e.g. the Card Issuer).

The Extradition Token is an RSA signature of the following data that will be included in an actual INSTALL [for extradition] command to be transmitted to the card.

Name	Length	Presence
Reference control parameter P1	1	Mandatory
Reference control parameter P2	1	Mandatory
Length of the following data fields (including the length fields) - see note below	1	Mandatory
Security Domain AID length	1	Mandatory
Security Domain AID	5-16	Conditional
Length = 0	1	Mandatory
Application or Executable Load File AID length	1	Mandatory
Application or Executable Load File AID	5-16	Mandatory
Length = 0	1	Mandatory
Length of Extradition Parameters field	1	Mandatory
Extradition Parameters field	0-n	Conditional

Table C-6: Data Elements Included in the Extradition Token

Note that 'Length of the following data fields' is the value of Lc of the INSTALL command, prior to a Token being added. It is coded on 1 byte with a value up to 255.

C.4.5 Registry Update Token

The Registry Update Token allows for the verification of the registry update process. The token is a signature authorizing the updating of the GlobalPlatform Registry data for an Application, and optionally the extradition of an Application from one Security Domain to another. The OPEN shall request the Security Domain with Token Verification privilege to verify the Registry Update Token.

Figure C-7 shows the Registry Update Token generation.

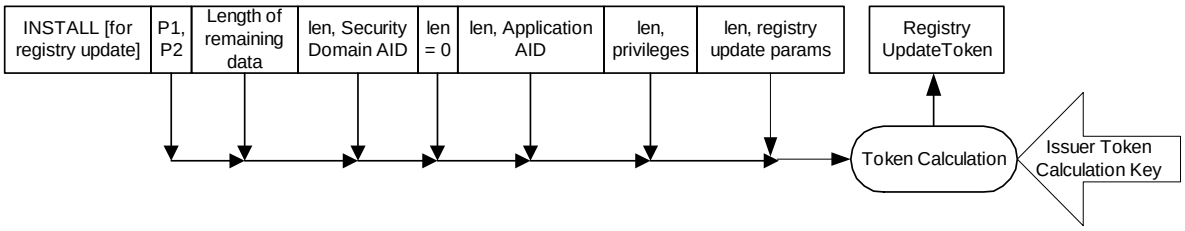

Figure C-7: Registry Update Token Calculation

Generating a Registry Update Token ensures the following:

- Only the Application included in this signature may have its GP Registry entry updated (and be extradited, in the case of extradition using registry update);
- The GP Registry entry of the Application may only be updated with the privileges and/or parameters included in this signature;
- The issuer of the Registry Update Token may only be the Security Domain Provider of the Security Domain with Token Verification Privilege (e.g. the Card Issuer).

The Registry Update Token is an RSA signature of the following data that will be included in an actual INSTALL [for registry update] command to be transmitted to the card.

Name	Length	Presence
Reference control parameter P1	1	Mandatory
Reference control parameter P2	1	Mandatory
Length of the following data fields (including the length fields) - see note below	1	Mandatory
Security Domain AID length	1	Mandatory
Security Domain AID	0 or 5-16	Conditional
Length = 0	1	Mandatory
Application AID length	1	Mandatory
Application AID	5-16	Conditional
Length of Privileges	1	Mandatory
Privileges (byte 1 - byte 2 - byte 3)	0-3	Conditional
Length of Registry Update Parameters	1	Mandatory
Registry Update Parameters	0-n	Conditional

Table C-7: Data Elements Included in the Registry Update Token

Note that 'Length of the following data fields' is the value of Lc of the INSTALL command, prior to a Token being added. It is coded on 1 byte with a value up to 255.

C.4.6 Delete Token

The Delete Token is a signature authorizing the deletion of card content from the card. It allows for the verification of the card content removal process The Token is a signature of selected fields within the DELETE command APDU and is appended to the DELETE command.

Figure C-8 shows the Delete Token generation.

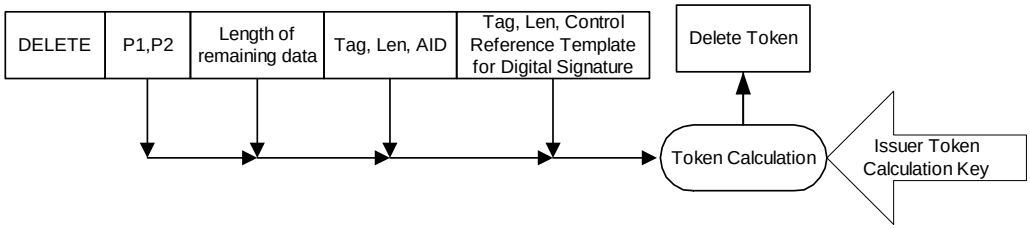

Figure C-8: Delete Token Calculation

Generating a Delete Token ensures the following:

- Only the Application or Executable Load File included in this signature may be deleted;
- The issuer of the Delete Token may only be the Security Domain Provider of the Security Domain with Token Verification Privilege (e.g. the Card Issuer).

The Delete Token is an RSA signature of the following data that will be included in the actual DELETE [card content] command to be transmitted to the card.

Name	Length
Reference control parameter P1	1
Reference control parameter P2	1
Length of the following data fields - see note below	1
Application or Executable Load File AID tag ('4F')	1
AID length	1
Application or Executable Load File AID	5-16
Control Reference Template for Digital Signature tag ('B6')	0 or 1
Control Reference Template for Digital Signature length	0-3
Control Reference Template for Digital Signature (Token) as defined in Table 11-22, comprising (in TLV format) Token Issuer id (tag '42'), Card Image Number (tag '45'), Application Provider id (tag '5F20') and Token identifier/number (tag '93')	0-n

Table C-8: Data Elements Included in the Delete Token

Note that 'Length of the following data fields' is the value of Lc of the INSTALL command, prior to a Token being added. It is coded on 1 byte with a value up to 255.

C.4.7 Load, Install and Make Selectable Token

The combined Load, Install and Make Selectable Token allows for the verification of the combined loading and installation process. The token is a signature authorizing the combined loading of an Executable Load File and installation to the card of an Application contained in a previously loaded file (Executable Load File). The OPEN shall request the Security Domain with Token Verification privilege to verify the combined Load, Install and Make Selectable Token.

Figure C-9 shows the combined Load, Install and Make Selectable Token generation.

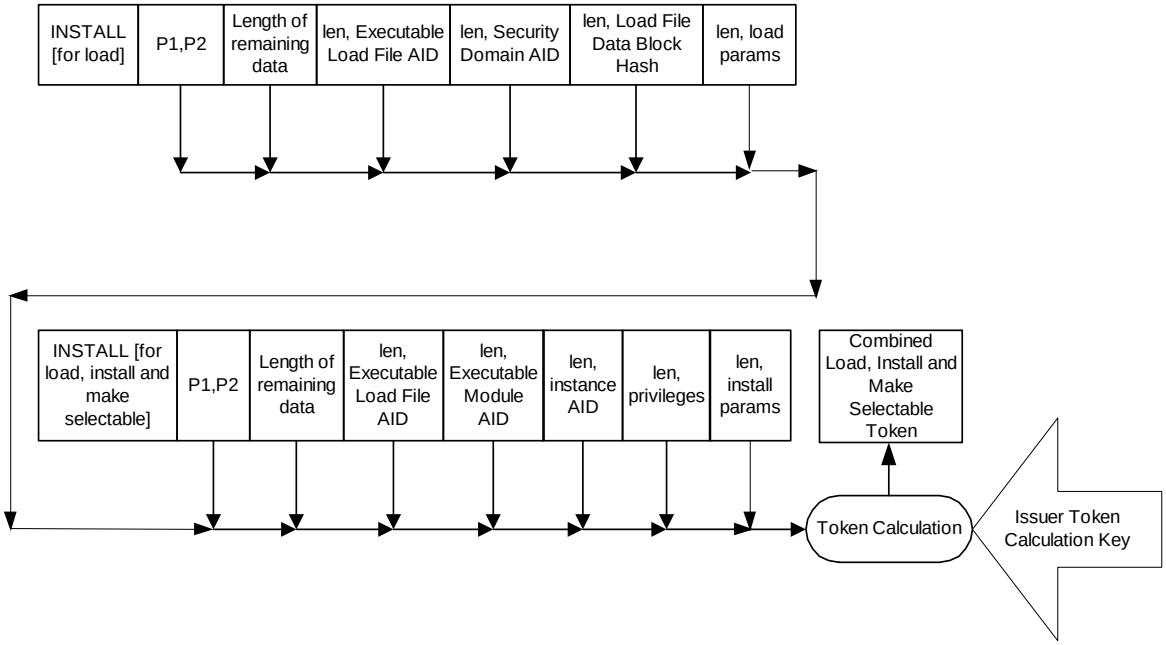

Figure C-9: Load, Install and Make Selectable Token Calculation

Generating a combined Load, Install and Make Selectable Token ensures the following:

- Only the application code included in this signature may be loaded to the card;
- The AID of the Load File Data Block may only be that included in the signature;
- The Executable Load File (derived from the Load File Data Block) and all Executable Modules within the Executable Load File may only be associated with the Security Domain included in the signature;
- Only the application (Executable Module) included in this signature and present in the Executable Load File may be installed on the card;
- That only the instance AID included in this signature may be used as the AID to select the instance;
- The Application may only be installed with the privileges included in this signature;
- Only the parameters included in this signature may be used to install the Application;
- The issuer of the combined Load, Install and Make Selectable Token may only be the Security Domain Provider of the Security Domain with Token Verification Privilege (e.g. the Card Issuer).

The combined Load, Install and Make Selectable Token is an RSA signature of the following data that will be included in an INSTALL [for load, install and make selectable] command.

Name	Length
Reference control parameter P1	1
Reference control parameter P2	1
Length of the following data fields (including the length fields)	1
Load File AID length	1
Load File AID	5-16
Security Domain AID length	1
Security Domain AID	0 or 5-16
Length of the Load File Data Block Hash	1
Load File Data Block Hash	0 or 20
Load parameters field length	1
Load parameters field	Var
Reference control parameter P1	1
Reference control parameter P2	1
Length of the following data fields (including the length fields)	1
Executable Load File AID length	1
Executable Load File AID	0 or 5-16
Executable Module AID length	1
AID within the Executable Load File	0 or 5-16
Instance AID length	1
Instance AID	5-16
Privileges length	1
Privileges (byte 1 - byte 2 - byte 3)	1 or 3
Install Parameters field length	1
Install Parameters (system and Application) field	Var

Table C-9: Data Elements Included in the Load, Install and Make Selectable Token

Note that 'Length of the following data fields' is the value of Lc of the INSTALL command, prior to a Token being added. It is coded on 1 byte with a value up to 255.

C.5 Receipts

Receipts are optional. They are generated by the Security Domain with Receipt Generation privilege, which requires knowledge of the appropriate receipt key. The Delete Receipt is comprised of data appropriate to the function that has been performed, together with Card Unique Data generated by the Security Domain with Receipt Generation privilege.

Each Receipt is generated either by:

- Symmetric cryptography using the receipt key, an ICV of binary zeroes and the signature method described in appendix B.1.2.2 - *Single DES Plus Final Triple DES MAC* across the receipt data. Prior to generating the signature, the data shall be padded according to appendix B.4 - *DES Padding*;

- asymmetric cryptography using the private key of the Security Domain with Receipt Generation privilege to decipher (sign) the SHA-1 digest of the receipt data using the signature method described in appendix B.2.1 - *Secure Hash Algorithm (SHA-1)*.

In addition to the appropriate cryptographic key, the Security Domain with Receipt Generation privilege also keeps track of a Confirmation Counter. The Confirmation Counter is a 16-bit value that is incremented by one (1) following each receipt generation. The Confirmation Counter is initialized to zero. When reaching its maximum value, the Confirmation Counter shall not be reset to zero. Cards are not required to support counter values beyond 32767.

C.5.1 Load Receipt

The Load Receipt provides confirmation that a successful load has occurred through the delegated loading process. A Load Receipt may be included in the response message for the last LOAD command of a sequence of LOAD commands.

Figure C-10 details the Load Receipt calculation.

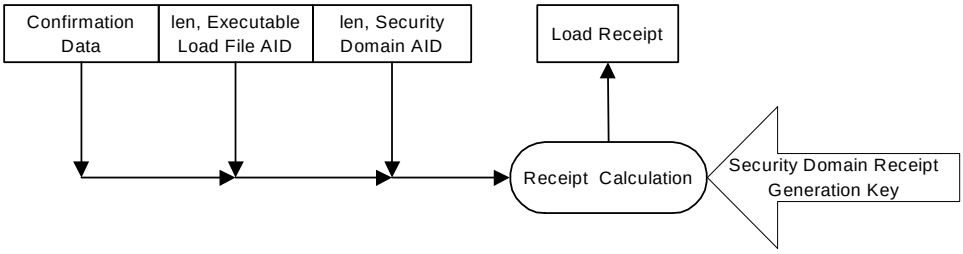

Figure C-10: Load Receipt Calculation

A Load Receipt is a signature of the following data:

Name	Length
Confirmation Data as defined in section 11.1.6	1-n
Executable Load File AID length	1
Executable Load File AID	5-16
Security Domain AID length	1
Security Domain AID	5-16

Table C-10: Data Elements Included in the Load Receipt

C.5.2 Install Receipt and Make Selectable Receipt

The Install/Make Selectable Receipt provides confirmation card that a successful installation/make selectable has occurred through the delegated installation/make selectable process. An Install/Make Selectable Receipt may be returned in the response message to the INSTALL [for install] command, the INSTALL [for make selectable] and the INSTALL [for install and make selectable] command. An Install Receipt is not returned in the response message to the INSTALL [for personalization] command.

Figure C-11 details the Install/Make Selectable Receipt calculation.

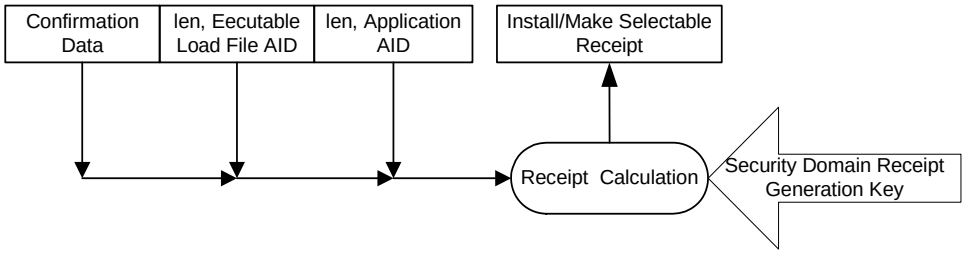

Figure C-11: Install/Make Selectable Receipt Calculation

An Install/Make Selectable Receipt is a signature of the following data:

Name	Length
Confirmation Data as defined in section 11.1.6	1-n
Executable Load File AID length	1
Executable Load File AID	5-16
Instance AID length indicator	1
Instance AID	5-16

Table C-11: Data Elements Included in the Install/Make Selectable Receipt

C.5.3 Extradition Receipt

The Extradition Receipt provides confirmation that the identified old Security Domain extradited the identified Application or Executable Load File to the identified new Security Domain, on a specific card, through the Delegated Management process. An Extradition Receipt may be returned in the response message to the INSTALL [for extradition] command.

Figure C-12 details the Extradition Receipt calculation performed by the Security Domain with Receipt Generation privilege.

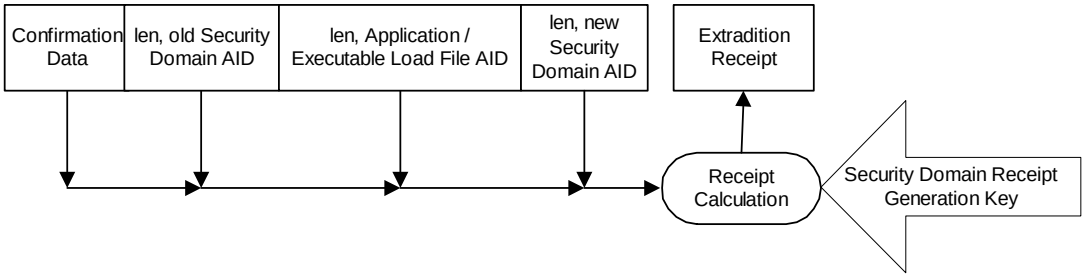

Figure C-12: Extradition Receipt Calculation

An Extradition Receipt is a signature of the following data:

Name	Length	Presence
Confirmation Data as defined in section 11.1.6	1-n	Mandatory
AID length indicator	1	Mandatory

Name	Length	Presence
Old Security Domain AID	5-16	Mandatory
AID length indicator	1	Mandatory
Application instance or Executable Load File AID	5-16	Mandatory
AID length indicator	1	Mandatory
New Security Domain AID	5-16	Mandatory

Table C-12: Data Elements Included in the Extradition Receipt

C.5.4 Registry Update Receipt

The Registry Update Receipt provides confirmation that GlobalPlatform Registry data (Privileges and/or Registry Update Parameters) for the identified Application was modified to the specified values on a specific card.

If registry update is used for extradition, the Registry Update Receipt also provides confirmation that the identified old Security Domain extradited the identified Application to the identified new Security Domain, on a specific card, through the Delegated Management process.

The following figure details the Registry Update Receipt calculation performed by the Security Domain with Receipt Generation privilege.

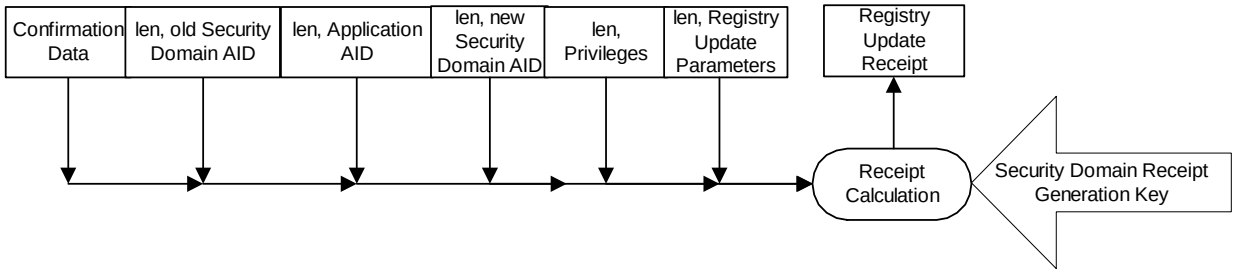

Figure C-13: Registry Update Receipt Calculation

A Registry Update Receipt is a signature of the following data:

Name	Length	Presence
Confirmation Data as defined in section 11.1.6	1-n	Mandatory
AID length indicator	1	Mandatory
Old Security Domain AID	0 or 5-16	Conditional
AID length indicator	1	Mandatory
Application AID	0 or 5-16	Conditional
AID length indicator	1	Mandatory
New Security Domain AID	0 or 5-16	Conditional
Length of Privileges	1	Mandatory
Privileges (byte 1 - byte 2 – byte 3)	0-3	Conditional
Length of Registry Update Parameters	1	Mandatory
Registry Update Parameters	0-n	Conditional

Table C-13: Data Elements Included in the Registry Update Receipt

The presence of conditional data depends on what Registry data (Privileges and/or Registry Update Parameters) is being updated and whether registry update is being used for extradition. See sections 11.5.2.3.5 and 11.5.2.3.7 for further details and the format of data. Security Domain AIDs are present for extradition.

C.5.5 Delete Receipt

The Delete Receipt provides proof that the identified Security Domain deleted an Application, an Executable Load File, or an Executable Load File and all its Application instances from a specific card through the delegated deletion process. The Delete Receipt may be returned in the response message to the DELETE [card content] command.

Figure C-14 details the Delete Receipt calculation performed by the Security Domain with Receipt Generation privilege.

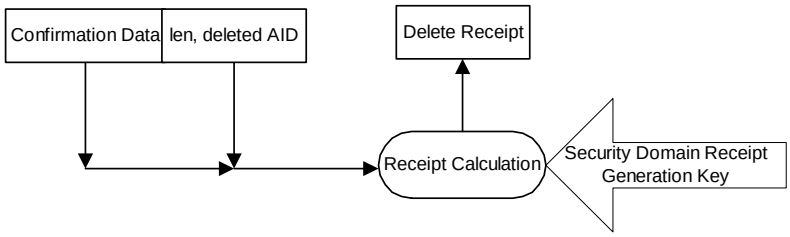

Figure C-14: Delete Receipt Calculation

A Delete Receipt is a signature of the following data:

Name	Length
Confirmation Data as defined in section 11.1.6	1-n
AID length indicator	1
Application instance or Executable Load File AID	5-16

Table C-14: Data Elements Included in the Delete Receipt

If the function was to delete an Executable Load File and all its Application instances, the receipt returns only the Executable Load File AID.

C.5.6 Combined Load, Install and Make Selectable Receipt

The combined Load, Install and Make Selectable Receipt provides confirmation card that a successful installation has occurred through the delegated installation process. A combined Load, Install and Make Selectable Receipt may be returned in the response message to the INSTALL [for load, install and make selectable] command.

Figure C-15 details the combined Load, Install and Make Selectable Receipt calculation.

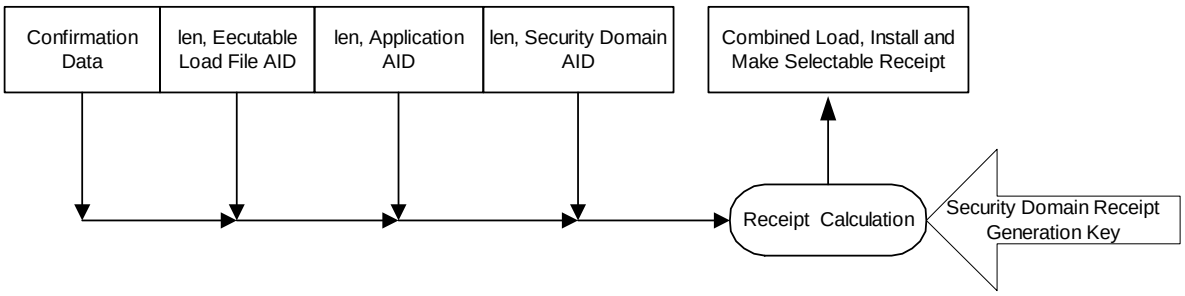

Figure C-15: Load, Install and Make Selectable Receipt Calculation

A combined Load, Install and Make Selectable Receipt is a signature of the following data:

Name	Length
Confirmation Data as defined in section 11.1.6	1-n
Executable Load File AID length	1
Executable Load File AID	5-16
Instance AID length indicator	1
Instance AID	5-16
Security Domain AID length	1
Security Domain AID	5-16

Table C-15: Data Elements Included in the Load, Install and Make Selectable Receipt

C.6 DAP Verification

Load File Data Block Signature verification is used to ensure that the Application has been validated.

In order to support DAP Verification, a Security Domain with DAP Verification privileges shall be present on the card. The Security Domain shall have knowledge of the appropriate key.

C.6.1 PKC Scheme

An entity generates a signature of the Load File Data Block Hash as defined in appendix B.2.1 - *Secure Hash Algorithm (SHA-1)*, i.e. computing a SHA-1 message digest of the Load File Data Block Hash and applying the DAP Verification private key to obtain the signature. This RSA signature is the decryption with the DAP Verification private key of the SHA-1 message digest of the Load File Data Block Hash (i.e. hash of a hash value). Padding of the data is as defined by SHA-1 and PKCS#1. The signature is provided to any entity responsible for loading the signed application code to a GlobalPlatform card. The AID of the Security Domain verifying the DAP along with this signature are placed in the Load File as a DAP Block to be transmitted to the card.

C.6.2 DES Scheme

A Security Domain's provider generates a signature of the Load File Data Block Hash. This signature is a MAC of the Load File Data Block Hash according to appendix B.1.2.2 - *Single DES Plus Final Triple DES*. Padding of the data is as defined in appendix B.4 - *DES Padding*. The signature is provided to any entity responsible for loading the signed application code to a GlobalPlatform card. The AID of the Security Domain verifying the DAP along with this signature are placed in the Load File as a DAP Block to be transmitted to the card.

C.7 GlobalPlatform on MULTOS

C.7.1 Keys

In order to perform Card Content management operations a GlobalPlatform on MULTOS smart card requires the following key:

- Issuer's key.

The following keys may optionally be supplied to the card during a card content loading operation:

- Application Provider's key;
- Application Provider's transport keys.

These keys are described in MAO-DOC-REF-009.

C.7.2 Cryptographic Structures

A GlobalPlatform on MULTOS smart card utilizes the following cryptographic structures:

- Public Key Certificate;
- Card Content Management Certificates:
  - Application Load Certificate;
  - Application Delete Certificate.
- Application Provider's Signature;
- Key Transformation Unit.

These structures are described in MAO-DOC-REF-009.

D Secure Channel Protocol '01' (Deprecated)

This Secure Channel Protocol is deprecated, and will be removed from a future version of the Specification. Use of another Secure Channel Protocol, such as Secure Channel Protocol '02' (SCP02), is recommended.

D.1 Secure Communication

D.1.1 SCP01 Secure Channel

The 3 levels of security provided by SCP01 are:

Mutual authentication - in which the card and the off-card entity each prove that they have knowledge of the same secrets;

Integrity and data origin authentication - in which the card ensures that the data being received from the off-card entity actually came from an authenticated off-card entity in the correct sequence and has not been altered;

Confidentiality - in which data being transmitted from the off-card entity to the card, is not viewable by an unauthorized entity.

A further level of security applies to sensitive data (e.g. secret keys) that shall always be transmitted as confidential data.

A maximum of 3 Supplementary Logical Channels is supported in SCP01.

Only explicit Secure Channel initiation is supported in SCP01. Both implicit and explicit Secure Channel termination are supported in SCP01.

In SCP01 the card shall support the following implementation options defined by "i" (see appendix H - *GlobalPlatform Data Values and Card Recognition Data*):

- "i" = '05': Initiation mode explicit, C-MAC on modified APDU, ICV set to zero, no ICV encryption, 3 Secure Channel Keys;
- "i" = '15': Initiation mode explicit, C-MAC on modified APDU, ICV set to zero, ICV encryption, 3 Secure Channel Keys.

Implementation option '15' is an enhancement over implementation option '05' and is therefore recommended.

D.1.2 Mutual Authentication

Mutual authentication is achieved through the process of initiating a Secure Channel and provides assurance to both the card and the off-card entity that they are communicating with an authenticated entity. If any step in the mutual authentication process fails, the process shall be restarted i.e. new challenges and Secure Channel Session generated.

The Secure Channel is explicitly initiated by the off-card entity using the INITIALIZE UPDATE and EXTERNAL AUTHENTICATE commands defined in this section. The Application may pass the APDU to the Security Domain using the appropriate API e.g. the processSecurity() method of a GlobalPlatform Java Card.

The explicit Secure Channel initiation allows the off-card entity to instruct the card (see appendix D.4.2 - *EXTERNAL AUTHENTICATE Command*) as to what level of security is required for the current Secure Channel (integrity and/or confidentiality) and apply this level of security to all the subsequent messages exchanged between the card and the off-card entity until the end of the Secure Channel Session. It also gives the off-card entity the possibility of selecting the Key Version Number and Key Identifier to be used (see the INITIALIZE UPDATE command).

Note: The explicit Secure Channel initiation also allows the card to inform the off-card entity what Secure Channel Protocol is supported, using the returned Secure Channel Protocol identifier.

The Secure Channel is always initiated (see appendix D.4.1 - *INITIALIZE UPDATE Command*) by the off-card entity by passing a "host" challenge (random data unique to this Secure Channel Session) to the card.

The card, on receipt of this challenge, generates its own "card" challenge (again random data unique to this Secure Channel Session).

The card, using the host challenge, the card challenge and its internal static keys, creates new secret Secure Channel session keys and generates a first cryptographic value (card cryptogram) using one of its newly created Secure Channel session keys (see appendix D.3.1 - *DES Session Keys*).

This card cryptogram along with the card challenge, the Secure Channel Protocol identifier, and other data is transmitted back to the off-card entity.

As the off-card entity should now have all the same information that the card used to generate the card cryptogram, it should be able to generate the same Secure Channel session keys and the same card cryptogram and by performing a comparison, it is able to authenticate the card.

The off-card entity now uses a similar process to create a second cryptographic value (host cryptogram) to be passed back to the card (see appendix D.4.2 - *EXTERNAL AUTHENTICATE Command*).

As the card has all the same information that the host used to generate the host cryptogram, it should be able to generate the same host cryptogram and, by performing a comparison, it is able to authenticate the off-card entity.

SCP01 Mutual Authentication Flow

The following flow is an example of mutual authentication between a card and an off-card entity. This flow shows mutual authentication occurring between a Security Domain and an off-card entity.

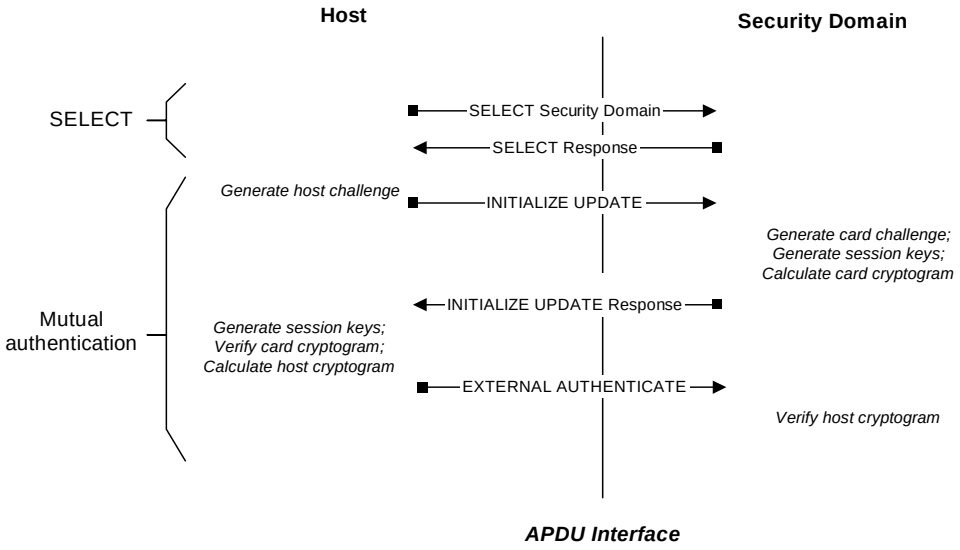

Figure D-1: Mutual Authentication Flow (Security Domain)

Expanding the authentication process shown in the flow described in section 7.3.4 - *Runtime Messaging Support*, it can be seen how an Application would use the services of an associated Security Domain to achieve mutual authentication.

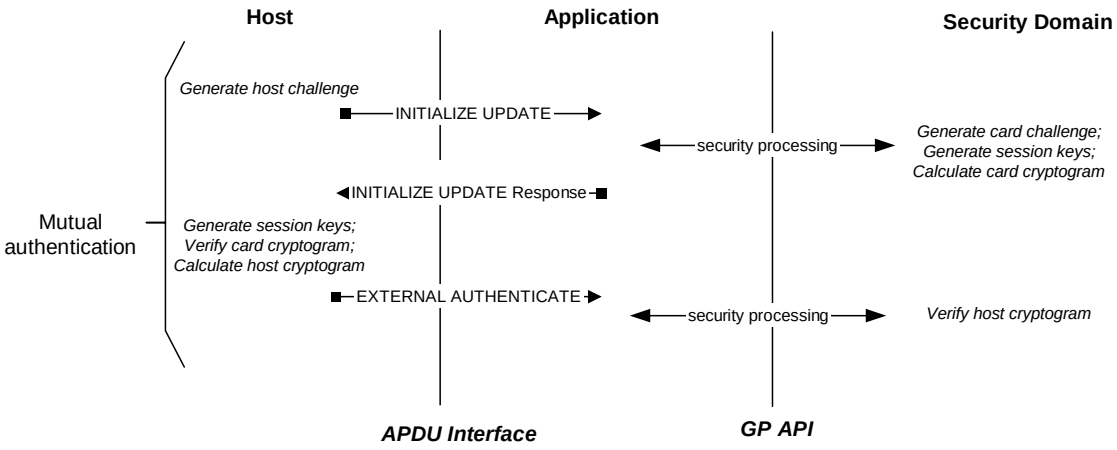

Figure D-2: Mutual Authentication Flow (using services of Security Domain)

D.1.3 Message Integrity

The C-MAC is generated by applying multiple chained DES operations (using a Secure Channel session key generated during the mutual authentication process) across the header and data field of an APDU command.

The card, on receipt of the message containing a C-MAC, using the same Secure Channel session key, performs the same operation and by comparing its internally generated C-MAC with the C-MAC received from the off-card entity is assured of the integrity of the full command. (If message data confidentiality has also been applied to the message, the C-MAC applies to the message data field before encryption.)

The integrity of the sequence of commands being transmitted to the card is achieved by using the C-MAC from the current command as the (possibly encrypted) Initial Chaining Vector (ICV) for the subsequent command. This ensures the card that all commands in a sequence have been received.

D.1.4 Message Data Confidentiality

The message data field is encrypted by applying multiple chained DES operations (using a Secure Channel session key generated during the mutual authentication process) across the entire data field of the command message to be transmitted to the card, regardless of its contents (clear text data and/or already protected sensitive data).

D.1.5 ICV Encryption

As an enhancement to the C-MAC mechanism, the ICV is encrypted before being applied to the calculation of the next C-MAC. The encryption mechanism used is triple DES with the C-MAC session key. The first ICV of a session, used to generate the C-MAC on the EXTERNAL AUTHENTICATE command, is not encrypted.

D.1.6 Security Level

The Current Security Level of a communication not included in a Secure Channel Session shall be set to NO_SECURITY_LEVEL.

For Secure Channel Protocol '01', the Current Security Level established in a Secure Channel Session is a bitmap combination of the following values: AUTHENTICATED, C_MAC, and C_DECRYPTION. The Current Security Level shall be set as follows:

- NO_SECURITY_LEVEL when a Secure Channel Session is terminated or not yet fully initiated;
- AUTHENTICATED after a successful processing of an EXTERNAL AUTHENTICATE command: AUTHENTICATED shall be cleared once the Secure Channel Session is terminated;
- AUTHENTICATED and C_MAC after a successful processing of an EXTERNAL AUTHENTICATE command with P1 indicating C-MAC (P1='01'): AUTHENTICATED and C-MAC shall be cleared once the Secure Channel Session is terminated;
- AUTHENTICATED, C_MAC, and C_DECRYPTION after a successful processing of an EXTERNAL AUTHENTICATE command with P1 indicating C-MAC and command encryption (P1= '03'): AUTHENTICATED, C_MAC, and C_DECRYPTION shall be cleared once the Secure Channel Session is terminated.

D.1.7 Protocol Rules

In accordance with the general rules described in chapter 10 - *Secure Communication*, the following protocol rules apply to Secure Channel Protocol '01':

- The successful initiation of a Secure Channel Session shall set the Current Security Level to the security level indicated in the EXTERNAL AUTHENTICATE command: it is at least set to AUTHENTICATED;
- The Current Security Level shall apply to the entire Secure Channel Session unless successfully modified at the request of the Application;
- When the Current Security Level is equal to NO_SECURITY_LEVEL:
  - If the Secure Channel Session was aborted during the same Application Session, the incoming command shall be rejected with a security error;
  - Otherwise no security verification of the incoming command shall be performed. The Application processing the command is responsible to apply its own security rules.
- If a Secure Channel Session is active (i.e. Current Security Level at least set to AUTHENTICATED), the security of the incoming command shall be checked according to the Current Security Level regardless of the command secure messaging indicator:
  - When the security of the command does not match (nor exceeds) the Current Security Level, the command shall be rejected with a security error, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;
  - If a security error is found, the command shall be rejected with a security error, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;
  - In all other cases, the Secure Channel Session shall remain active and the Current Security Level unmodified. The Application is responsible for further processing the command.
- If the Security Domain supports application data encryption and/or decryption, it shall decrypt or encrypt a block of secret data upon request. If the service is not supported or if (one of) the appropriate cryptographic key(s) is not available, the request shall be rejected but the Current Security Level, Session Security Level and Secure Channel Session in operation (if any) shall not be impacted;
- If a Secure Channel Session is aborted, it is still considered not terminated;
- The current Secure Channel Session shall be terminated (if aborted or still open) and the Current Security Level reset to NO_SECURITY_LEVEL on either:
  - Attempt to initiate a new Secure Channel Session (new INITIALIZE UPDATE command);
  - Termination of the Application Session (e.g. new Application selection);

- Termination of the associated logical channel;
- Termination of the Card Session (card reset or power off);
- Explicit termination by the Application (e.g. invoking GlobalPlatform API).

D.2 Cryptographic Keys

Key	Usage	Length	Remark
Secure Channel Encryption Key (S-ENC)	Secure Channel Authentication & Encryption (DES)	16 bytes	Mandatory
Secure Channel Message Authentication Code Key (S-MAC)	Secure Channel MAC Verification (DES)	16 bytes	Mandatory
Data Encryption Key (DEK)	Sensitive Data Decryption (DES)	16 bytes	Mandatory

Table D-1: Security Domain Secure Channel Keys

A Security Domain, including the Issuer Security Domain shall have at least one key set containing 3 keys to be used in the initiation and use of a Secure Channel. These keys are all double length DES keys and are the following:

- The Secure Channel encryption key (S-ENC) and the Secure Channel MAC key (S-MAC). These keys are only used to generate Secure Channel session keys during the initiation of a Secure Channel;
- The data encryption key (DEK) for decrypting sensitive data, e.g. secret or private keys. This key is a double length DES key and is used as a static key.

D.3 Cryptographic Usage

D.3.1 DES Session Keys

DES session keys shall be generated every time a Secure Channel is initiated and are used in the mutual authentication process. These same session keys may be used for subsequent commands if the Current Security Level indicates that secure messaging is required.

Session keys are generated to ensure that a different set of keys is used for each Secure Channel Session. While this is not required for key encryption operations, it is important for authentication operations, MAC generation and verification and command message encryption and decryption. It is therefore only necessary to create session keys from the static Secure Channel encryption key (S-ENC) and the Secure Channel MAC key (S-MAC).

DES session keys are created using the static Secure Channel encryption key (S-ENC) and the Secure Channel MAC key (S-MAC) and the random host and card challenges. Creating session keys involves 3 steps.

- Generating the session key derivation data. (The same derivation data is used to create both the Secure Channel encryption and Secure Channel MAC session keys.);
- Creating the Secure Channel encryption session key;
- Creating the Secure Channel MAC session key.

The DES operation used to generate these keys is always triple DES in ECB mode.

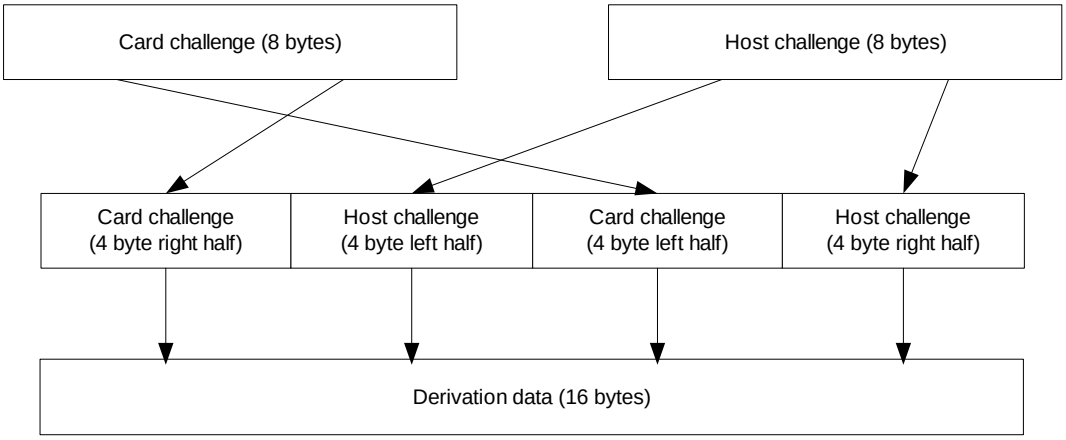

Figure D-3: Session Key - Step 1 - Generate Derivation data

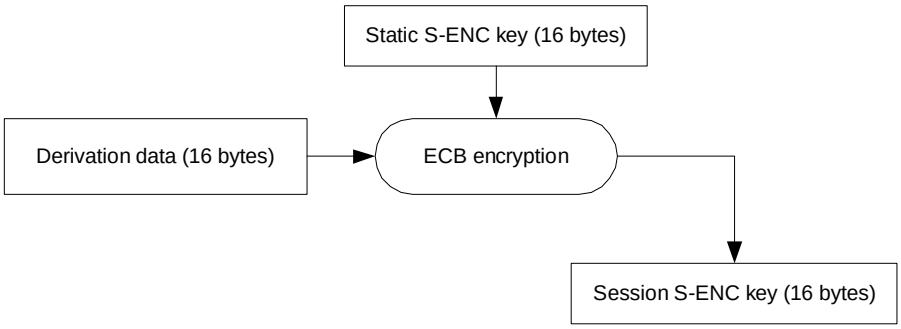

Figure D-4: Session Key - Step 2 - Create S-ENC Session Key

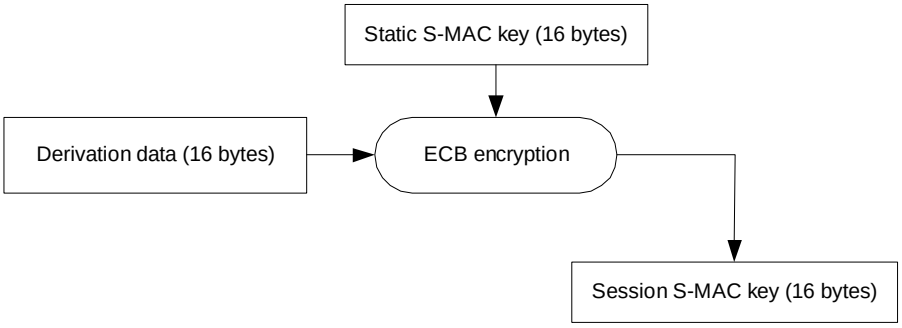

Figure D-5: Session Key – Step 3 - Create S-MAC Session Key

D.3.2 Authentication Cryptograms

Both the card and the off-card entity (Host) generate an authentication cryptogram. The off-card entity verifies the card cryptogram and the card verifies the host cryptogram. Generating or verifying an authentication cryptogram uses the S-ENC session key and the signing method described in appendix B.1.2.1 - *Full Triple DES*.

D.3.2.1 Card Authentication Cryptogram

The generation and verification of the card cryptogram is performed by concatenating the 8-byte host challenge and 8-byte card challenge resulting in a 16-byte block.

Applying the same padding rules defined in appendix B.4 - *DES Padding*, the data shall be padded with a further 8-byte block ('80 00 00 00 00 00 00 00').

The signature method, using the S-ENC session key and an ICV of binary zeroes, is applied across this 24-byte block and the resulting 8-byte signature is the card cryptogram.

D.3.2.2 Host Authentication Cryptogram

The generation and verification of the host cryptogram is performed by concatenating the 8-byte card challenge and 8-byte host challenge resulting in a 16-byte block.

Applying the same padding rules defined in appendix B.4 - *DES Padding*, the data shall be padded with a further 8-byte block ('80 00 00 00 00 00 00 00').

The signature method, using the S-ENC session key and an ICV of binary zeroes, is applied across this 24-byte block and the resulting 8-byte signature is the host cryptogram.

D.3.3 APDU Command MAC Generation and Verification

The Secure Channel mandates the use of a MAC on the EXTERNAL AUTHENTICATE command. Depending on the Session Security Level defined in the initiation of the Secure Channel, all other commands within the Secure Channel may require secure messaging and as such the use of a C-MAC.

A C-MAC is generated by an off-card entity and applied across the full APDU command being transmitted to the card including the header (5 bytes) and the data field in the command message. (It does not include Le.)

Modification of the APDU command header and padding is required prior to the MAC operation being performed.

The rules for APDU command header modification are as follows:

- The length of the command message (Lc) shall be incremented by 8 to indicate the inclusion of the C-MAC in the data field of the command message;
- The class byte shall be modified to indicate that this APDU command includes secure messaging. This is achieved by setting bit b3 of the class byte. For all the commands defined in this specification, the class byte of commands that contain secure messaging shall be '84'. The C-MAC generation and verification does not reflect the logical channel information included in the class byte of the command: the logical channel number is always assumed to be zero in the generation or verification of the C-MAC;
- The rules for MAC padding are as defined in appendix B.4 - *DES Padding*.

As the ICV is used to chain the commands for command sequence integrity, the value of the ICV depends on which APDU command within the sequence the MAC is being generated for:

- For the EXTERNAL AUTHENTICATE command, the ICV is set to binary zeroes;

- For any command following the EXTERNAL AUTHENTICATE command, the ICV is the C-MAC value successfully verified for the previous command received by the card. (Prior to using the ICV, the ICV can be encrypted as described in appendix D.1.5 - ICV Encryption.)

The signature method defined in appendix B.1.2.1 - *Full Triple DES*, using the MAC session key and the ICV, is applied across the padded data block and the resulting 8-byte signature is the C-MAC.

The C-MAC is appended at the end of the APDU command message excluding any padding but including the modifications made to the command header (class and Lc).

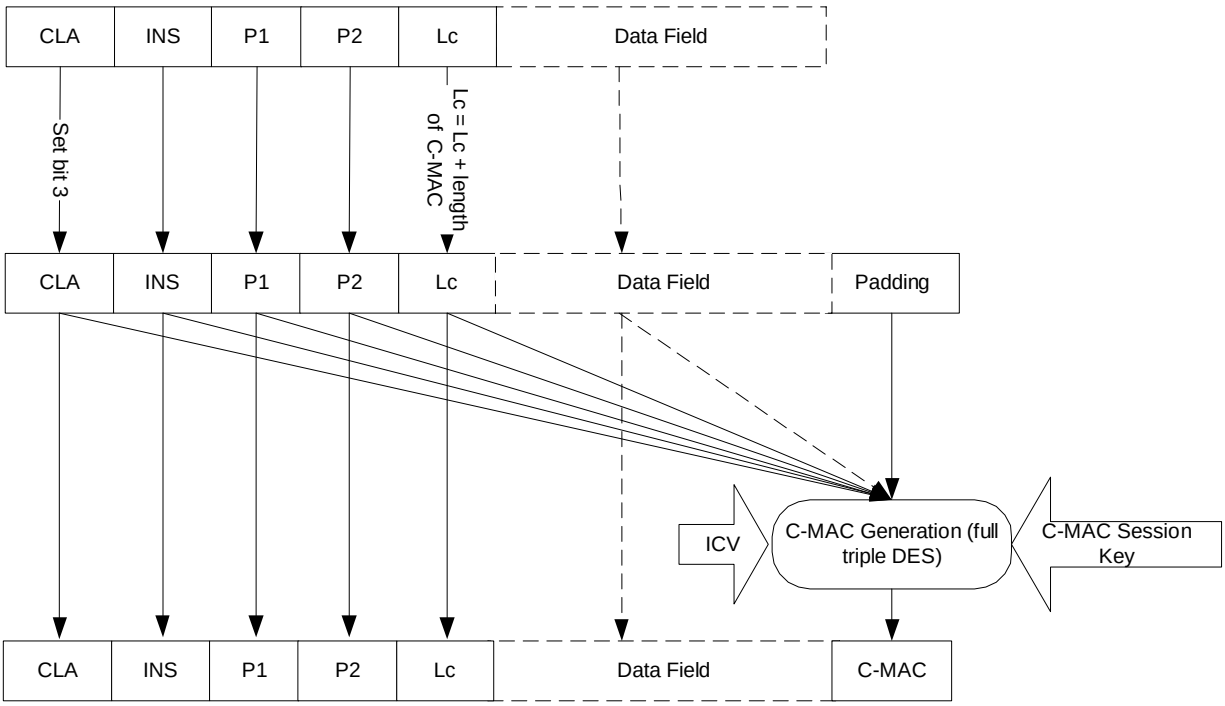

Figure D-6: APDU Command MAC Generation and Verification

If no other secure messaging is required, the message is now prepared for transmission to the card. The C-MAC shall be retained and used as the ICV for the subsequent C-MAC verification (if any).

The card, in order to verify the C-MAC, shall perform the same padding mechanism to the data and use the same ICV and MAC session key employed by the off-card entity in order to verify the C-MAC. As with the off-card entity, the C-MAC shall be retained and used as the ICV for the subsequent C-MAC verification (if any).

If the Secure Channel Session is occurring on a Supplementary Logical Channel, the logical channel number is added to the class byte of the message after the C-MAC has been generated and prior to the message being transmitted to the card. The card shall ignore any logical channel information in the class byte (i.e. assume the logical channel number is zero) prior to verifying the C-MAC.

D.3.4 APDU Data Field Encryption and Decryption

Depending on the Session Security Level defined in the initiation of the Secure Channel, all subsequent APDU commands within the Secure Channel may require secure messaging and as such the use of a C-MAC (integrity) and encryption (confidentiality).

If confidentiality is required, the off-card entity encrypts the "clear text" data field of the command message being transmitted to the card. (If the APDU command does not originally contain command data, encryption is not performed.) This excludes the header and the C-MAC but includes any data within the data field that has been protected for another purpose e.g. secret or private keys encrypted with the data encryption key (DEK).

Command message encryption and decryption uses the Secure Channel encryption (S-ENC) session key and the full triple DES encryption method described in appendix B.1.1.1 – *Encryption/Decryption CBC Mode*

Prior to encrypting the data, the data shall be padded. Unlike the C-MAC, this padding now becomes part of the data field and this necessitates further modification of the Lc value.

Padding of the data field to be encrypted is performed according to the following rules:

- The length of the original, "clear text", data field is appended to the left of, and becomes part of the command data;
- If the length of the data field is now a multiple of 8, no further padding is required else continue with padding as defined in appendix B.4 - *DES Padding*.

The number of bytes appended to the data field in order to fulfill the above padding shall be added to Lc. This includes the mandatory length byte, the optional '80' and any optional binary zeroes.

The encryption can now be performed across the padded data field using the Secure Channel encryption session key (S-ENC) and the result of each encryption becomes part of the encrypted data field in the command message.

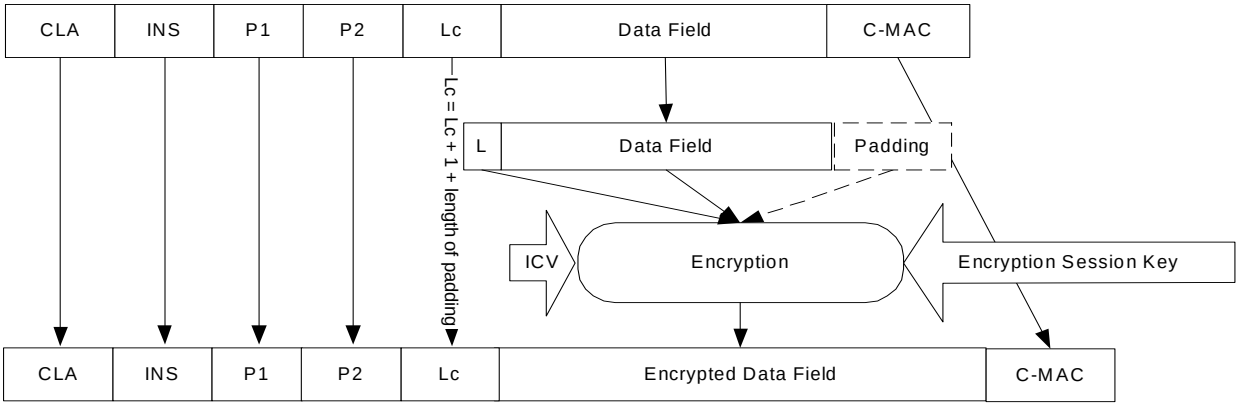

Figure D-7: APDU Data Field Encryption

Note: The ICV and the chaining are only used to link the blocks of the data currently being encrypted. For this reason the ICV for command data encryption is always binary zeroes.

The message is now prepared for transmission to the card.

The card is required to first decrypt the command message and strip off any padding prior to attempting to verify the C-MAC. This decryption uses an ICV of binary zeroes and the same encryption session key employed by the off-card entity. The padding shall be removed and Lc shall be modified to reflect the length prior to encryption i.e. original clear text data plus C-MAC length.

D.3.5 Key Sensitive Data Encryption and Decryption

Key data encryption is used when transmitting key sensitive data to the card and is over and beyond the security level required for the Secure Channel. For instance all DES keys transmitted to a card should be encrypted.

The key data encryption process uses the static data encryption key and the encryption method described in appendix B.1.1.2 - *ECB Mode*.

As all DES keys are by their very nature a multiple of 8-byte lengths no padding is required for key encryption operations. Similarly the sensitive data block length shall be constructed as a multiple of 8-byte long block before the data encryption operations: the eventual padding method is application specific.

The encryption is performed across the key sensitive data and the result of each encryption becomes part of the encrypted key data. This encrypted key data becomes part of the "clear text" data field in the command message.

The on-card decryption of key data is the exact opposite of the above operation; in particular, no padding is removed by the decryption operation.

D.4 Secure Channel APDU Commands

Table D-2 provides the list of minimum security requirements for SCP01 commands.

Command	Minimum Security
INITIALIZE UPDATE	None
EXTERNAL AUTHENTICATE	C-MAC

Table D-2: Minimum Security Requirements for SCP01 Commands

The following table provides the list of SCP01 command support per card Life Cycle State.

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED		TERMINATED	
	AM SD	DM SD	SD	AM SD	DM SD	SD	AM SD	DM SD	SD	FA SD	SD	FA SD	SD
INITIALIZE UPDATE	✓	✓		✓	✓		✓	✓		✓			
EXTERNAL AUTHENTICATE	✓	✓		✓	✓		✓	✓		✓			

Table D-3: SCP01 Command Support per Card Life Cycle State

Legend:

AM SD: Security Domain with Authorized Management privilege.

DM SD: Supplementary Security Domain with Delegated Management privilege.

FA SD: Security Domain with Final Application privilege. (Note: command support for an Application with Final Application Privilege which is not a Security Domain is subject to Issuer policy, within the constraints of what is permitted according to the card Life Cycle State.)SD: Supplementary Security Domain without Delegated Management privilege.

SD: Other Security Domain.

✓ : Support required.

Blank cell: Support optional.

Striped cell: Support prohibited.

D.4.1 INITIALIZE UPDATE Command

D.4.1.1 Definition and Scope

The INITIALIZE UPDATE command is used to transmit card and Secure Channel Session data between the card and the host. This command initiates the initiation of a Secure Channel Session.

At any time during a current Secure Channel, the INITIALIZE UPDATE command can be issued to the card in order to initiate a new Secure Channel Session.

This INITIALIZE UPDATE command is not to be confused with commands of the same name in legacy applications.

D.4.1.2 Command Message

The INITIALIZE UPDATE command message is coded according to the following table:

Code	Value	Meaning
CLA	'80' - '83	See section 11.1.4
INS	'50'	INITIALIZE UPDATE
P1	'xx'	Key Version Number
P2	'xx'	Key Identifier
Lc	'08'	Length of host challenge
Data	'xx xx…'	Host challenge
Le	'00'	

Table D-4: INITIALIZE UPDATE Command Message

D.4.1.3 Reference Control Parameter P1 - Key Version Number

The Key Version Number defines the Key Version Number within the Security Domain to be used to initiate the Secure Channel Session. If this value is zero, the first available key chosen by the Security Domain will be used.

D.4.1.4 Reference Control Parameter P2 - Key Identifier

The Key Identifier together with the Key Version Number defined in reference control parameter P1 provide a unique reference to the set of keys to be used to initiate the Secure Channel Session.

D.4.1.5 Data Field Sent in the Command Message

The data field of the command message contains 8 bytes of host challenge. This challenge, chosen by the off-card entity, should be unique to this session.

D.4.1.6 Response Message

The data field of the response message shall contain the concatenation without delimiters of the following data elements:

Name	Length
Key diversification data	10 bytes
Key information	2 bytes
Card challenge	8 bytes
Card cryptogram	8 bytes

Table D-5: INITIALIZE UPDATE Response Message

The key diversification data is data typically used by a backend system to derive the card static keys.

The key information includes the Key Version Number and the Secure Channel Protocol identifier, here '01', used in initiating the Secure Channel Session.

The card challenge is an internally generated random number.

The card cryptogram is an authentication cryptogram.

D.4.1.7 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or the following error condition:

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table D-6: INITIALIZE UPDATE Error Condition

D.4.2 EXTERNAL AUTHENTICATE Command

D.4.2.1 Definition and Scope

The EXTERNAL AUTHENTICATE command is used by the card to authenticate the host and to determine the level of security required for all subsequent commands.

A successful execution of the INITIALIZE UPDATE command shall precede this command.

D.4.2.2 Command Message

The EXTERNAL AUTHENTICATE command message is coded according to the following table:

Code	Value	Meaning
CLA	'84' - '87'	See section 11.1.4
INS	'82'	EXTERNAL AUTHENTICATE
P1	'xx'	Security level
P2	'00'	Reference control parameter P2
Lc	'10'	Length of host cryptogram and MAC
Data	'xx xx…'	Host cryptogram and MAC
Le		Not present

Table D-7: EXTERNAL AUTHENTICATE Command Message

D.4.2.3 Reference Control Parameter P1 - Security Level

The reference control parameter P1 defines the level of security for all secure messaging commands following this EXTERNAL AUTHENTICATE command (it does not apply to this command) and within this Secure Channel Session.

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	0	0	0	0	0	1	1	C-DECRYPTION and C-MAC.
0	0	0	0	0	0	0	1	C-MAC
0	0	0	0	0	0	0	0	No secure messaging expected.

Table D-8: EXTERNAL AUTHENTICATE Reference Control Parameter P1

D.4.2.4 Reference Control Parameter P2

The reference control parameter P2 shall always be set to '00'.

D.4.2.5 Data Field Sent in the Command Message

The data field of the command message contains the host cryptogram and the APDU command MAC.

D.4.2.6 Data Field Returned in the Response Message

The data field of the response message is not present.

D.4.2.7 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or the following error condition:

SW1	SW2	Meaning
'63'	'00'	Authentication of host cryptogram failed

Table D-9: EXTERNAL AUTHENTICATE Error Condition

E Secure Channel Protocol '02'

E.1 Secure Communication

Secure Channel Protocol '02' (SCP02) supports up to 19 Supplementary Logical Channels.

E.1.1 SCP02 Secure Channel

Either the card or the off-card entity may play the role of being the secure sending and receiving entities. SCP02 provides the three followings levels of security:

Entity authentication - in which the card authenticates the off-card entity and the off-card entity may authenticate the card, proving that the off-card entity has knowledge of the same secret(s) as the card;

Integrity and Data origin authentication - in which the receiving entity (the card or off-card entity) ensures that the data being received actually came from an authenticated sending entity (respectively the off-card entity or card) in the correct sequence and has not been altered;

Confidentiality - in which data being transmitted from the sending entity (the off-card entity or card) to the receiving entity (respectively the card or off-card entity) is not viewable by an unauthorized entity.

A further level of security applies to sensitive data (e.g. secret keys) that shall always be transmitted as confidential data.

In SCP02 the "i" parameter as defined in appendix H - *GlobalPlatform Data Values and Card Recognition Data*) shall be formed as a bit map on one byte as follows:

b8	b7	b6	b5	b4	b3	b2	b1	Description
Reserved							1	3 Secure Channel Keys
Reserved							0	1 Secure Channel base key
Reserved						1		C-MAC on unmodified APDU
Reserved						0		C-MAC on modified APDU
Reserved					1			Initiation mode explicit
Reserved					0			Initiation mode implicit
Reserved				1				ICV set to MAC over AID
Reserved				0				ICV set to zero
Reserved			1					ICV encryption for C-MAC session
Reserved			0					No ICV encryption
Reserved		1						R-MAC support
Reserved		0						No R-MAC support
Reserved	1							Well-known pseudo-random algorithm (card challenge)
Reserved	0							Unspecified card challenge generation method

Table E-1: Values of Parameter "i"

Note: "i" is a subidentifier within an object identifier, and bit b8 is reserved for use in the structure of the object identifier according to ISO/IEC 8825-1.

The Security Domain shall support at least one of the following implementation options as defined by "i":

- "i" = '15': Initiation mode explicit, C-MAC on modified APDU, ICV set to zero, ICV encryption for C-MAC session, 3 Secure Channel Keys, unspecified card challenge generation method, no R-MAC;
- "i" = '1A': Initiation mode implicit, C-MAC on unmodified APDU, ICV set to MAC over AID, ICV encryption for C-MAC session, 1 Secure Channel base key. no R-MAC;
- "i" = '55': Initiation mode explicit, C-MAC on modified APDU, ICV set to zero, ICV encryption for C-MAC session, 3 Secure Channel Keys, well-known pseudo-random algorithm (card challenge), no R-MAC.

E.1.2 Entity Authentication

Off-card entity authentication is achieved through the process of initiating a Secure Channel and provides assurance to the card that it is communicating with an authenticated off-card entity. If any step in the off-card authentication process fails, the process shall be restarted (i.e. new session keys generated).

The Secure Channel initiation and off-card entity authentication implies the creation of session keys derived from card static key(s) using a Secure Channel Sequence Counter maintained by the Security Domain. The Security Domain shall manage one Sequence Counter per Secure Channel base key or Secure Channel keys sharing the same Key Version Number (see appendix E.2 - *Cryptographic Keys*).

The Sequence Counter is incremented by 1 when and only when the first C-MAC of a secure channel (started implicitly or explicitly) is verified as valid. It is incremented before the processing specific for the command. The Sequence Counter is reset to zero on creation or update of the Secure Channel key(s). When the Secure Channel key set contains more than one key (see appendix E.2 - *Cryptographic Keys*), the Sequence Counter is reset to zero on the creation or update of any one of the keys of this key set.

Note: the Sequence Counter is not updated for a C-MAC that is not the first of a Secure Channel Session or when a Secure Channel Session is started and the C-MAC is invalid. In this respect, the Sequence Counter keeps track of the number of valid Secure Channel Sessions the corresponding secure messaging key set has experienced so far. When reaching its maximum value, the Sequence Counter shall not be reset to zero. Cards are not required to support counter values beyond 32767.

E.1.2.1 Explicit Secure Channel Initiation

The Secure Channel may be explicitly initiated by the off-card entity using the INITIALIZE UPDATE and EXTERNAL AUTHENTICATE commands. The Application may pass the APDU to the Security Domain using the appropriate API e.g. the processSecurity() method of a GlobalPlatform Java Card.

The explicit Secure Channel initiation allows the off-card entity to instruct the card (see appendix E.5.2 - *EXTERNAL AUTHENTICATE Command*) as to what level of security is required for the current Secure Channel (integrity and/or confidentiality) and apply this level of security to all the subsequent messages exchanged between the card and the off-card entity until the end of the session. It also gives the off-card entity the possibility of selecting the Key Version Number to be used (see appendix E.5.1 - *INITIALIZE UPDATE Command*).

Note: The explicit Secure Channel Session initiation also allows the card to inform the off-card entity what Secure Channel Protocol is supported, using the returned Secure Channel Protocol identifier.

The Secure Channel is always initiated (see appendix E.5.1 - *INITIALIZE UPDATE Command*) by the off-card entity by passing a “host” challenge (random data unique to this session) to the card.

The card, on receipt of this challenge, generates its own “card” challenge (again random data unique to this session).

The card, using its internal Sequence Counter and static keys, creates new secret session keys and generates a first cryptographic value (card cryptogram) using one of its newly created session keys (see appendix E.4.1 - *DES Session Keys*).

This card cryptogram along with the Sequence Counter, the card challenge, the Secure Channel Protocol identifier, and other data is transmitted back to the off-card entity.

As the off-card entity should now have all the same information that the card used to generate the card cryptogram, it should be able to generate the same session keys and the same card cryptogram and by performing a comparison, it is able to authenticate the card.

The off-card entity now uses a similar process to create a second cryptographic value (host cryptogram) to be passed back to the card (see appendix E.5.2 - *EXTERNAL AUTHENTICATE Command).*

As the card has all the same information that the host used to generate the host cryptogram, it should be able to generate the same cryptogram and, by performing a comparison, it is able to authenticate the off-card entity.

The off-card entity also creates a MAC to be passed back to the card and verified by the card. The verified MAC is used by the card to create the Initial Chaining Vector for the verification of the subsequent C-MAC and/or R-MAC.

When the off-card entity is successfully authenticated, the card increments its internal Secure Channel Sequence Counter.

Explicit Secure Channel Initiation Flow

The following flow is an example of explicit Secure Channel initiation between a card and an off-card entity. Expanding the authentication process shown in the flow described in section 7.3.4 - *Runtime Messaging Support*, it can be seen how an Application would use the services of a Security Domain to achieve the explicit Secure Channel initiation.

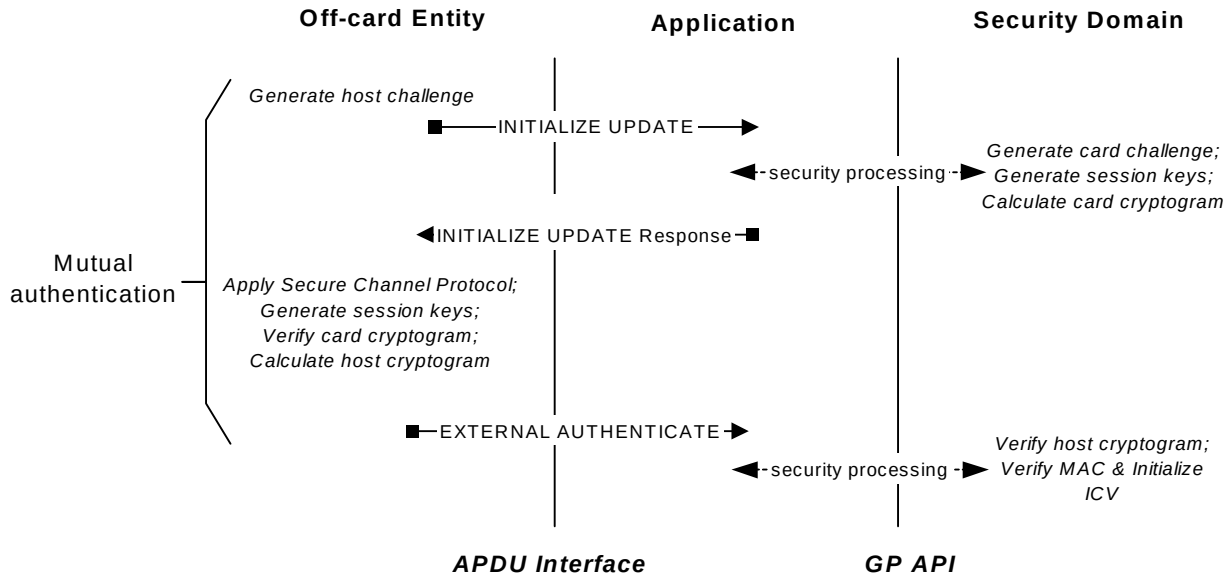

Figure E-1: Explicit Secure Channel Initiation Flow

E.1.2.2 Implicit Secure Channel Initiation

The Secure Channel is implicitly initiated when receiving the first APDU command that contains a cryptographic protection (C-MAC). The required level of security is implicitly known by both the card and off-card entity as command message integrity only. The off-card entity implicitly knows which keys are to be used and the current

value of the Secure Channel Sequence Counter, or may have previously retrieved the corresponding information using a GET DATA command.

The off-card entity, using the information it knows about the card static keys and its Secure Channel Sequence Counter, creates new secret session keys and generates a C-MAC on an APDU command message.

The Secure Channel is initiated by the off-card entity by appending a C-MAC to an APDU command.

The card, on receipt of this first C-MAC, creates the C-MAC session key using its internal card static key(s) and Secure Channel Sequence Counter.

The card has the same information that the host used to generate the C-MAC. It generates the same MAC and, by performing a comparison, it authenticates the off-card entity.

When the off-card entity is successfully authenticated, the card increments its internal Secure Channel Sequence Counter.

For sensitive data decryption, the card creates the data encryption session key using its internal card static key(s) and the newly incremented value of the Secure Channel Sequence Counter. For R-MAC generation, the card creates the R-MAC session key using its internal card static key(s) and the newly incremented value of the Secure Channel Sequence Counter.

Note: In addition to command message integrity, the off-card entity may, at any moment during the Secure Channel Session, initiate response message integrity using the BEGIN R-MAC SESSION command.

E.1.3 Message Integrity

The MAC is generated by applying multiple chained DES operations (using a session key generated prior to or when opening the Secure Channel) across an APDU message.

A MAC may be generated on the following:

- C-MAC for APDU command messages (generated by the off-card entity);
- R-MAC for APDU response messages (generated by the card).

The receiving entity, on receipt of the message containing a MAC, using the same session key, performs the same operation and by comparing its generated MAC with the MAC received from the sending entity is assured of the integrity of the full command or response. (If message data confidentiality has also been applied to the message, the MAC applies to the message data field before encryption.)

The integrity of the sequence of APDU command or response messages being transmitted to the receiving entity is achieved by using the MAC from the current command or response as the (possibly encrypted) Initial Chaining Vector (ICV) for the subsequent command or response. This ensures the receiving entity that all messages in a sequence have been received. Computing the ICV is detailed in appendix E.3 - *Cryptographic Algorithms.*

E.1.4 Message Data Confidentiality

The message data field is encrypted by applying multiple chained DES operations (using a session key generated during the Secure Channel initiation process) across the entire data field of either a command message or a response, regardless of its contents (clear text data and/or already protected sensitive data).

E.1.5 Security Level

The Current Security Level of a communication not included in a Secure Channel Session shall be set to NO_SECURITY_LEVEL.

For Secure Channel Protocol '02' with explicit initiation mode, the Current Security Level established in a Secure Channel Session is a bitmap combination of the following values: AUTHENTICATED, C_MAC, R_MAC, and C_DECRYPTION. The Current Security Level shall be set as follows:

- NO_SECURITY_LEVEL when a Secure Channel Session is terminated or not yet fully initiated;
- AUTHENTICATED after a successful processing of an EXTERNAL AUTHENTICATE command: AUTHENTICATED shall be cleared once the Secure Channel Session is terminated;
- C_MAC after a successful processing of an EXTERNAL AUTHENTICATE command with P1 indicating C-MAC (P1='x1' or 'x3'): C-MAC shall be cleared once the Secure Channel Session is terminated. Note that C_MAC is always combined with AUTHENTICATED and simultaneously set and cleared;
- C_DECRYPTION after a successful processing of an EXTERNAL AUTHENTICATE command with P1 indicating Command Encryption (P1= 'x3'): C_DECRYPTION shall be cleared once the Secure Channel Session is terminated. Note that C_DECRYPTION is always combined with AUTHENTICATED and C_MAC and simultaneously set and cleared;
- R_MAC after a successful processing of an EXTERNAL AUTHENTICATE command with P1 indicating R-MAC (P1='1x'): R-MAC shall be cleared once the Secure Channel Session is terminated. Note that in this case R_MAC is always combined with AUTHENTICATED and simultaneously set and cleared. R_MAC may also be combined with C_MAC or C_DECRYPTION (according to the P1 value of the EXTERNAL AUTHENTICATE command) and simultaneously set and cleared;
- R_MAC after a successful processing of a BEGIN R-MAC SESSION command: R-MAC shall be cleared after a successful processing of an END R-MAC SESSION command. Note that in this case R_MAC may be combined with AUTHENTICATED, C_MAC or C_DECRYPTION depending on the pre-existing Current Security Level of the Secure Channel Session. R_MAC is set and cleared independently of AUTHENTICATED, C_MAC or C_DECRYPTION.

For Secure Channel Protocol '02' with implicit initiation mode, the Current Security Level established in a Secure Channel Session is a bitmap combination of the following values: AUTHENTICATED, C_MAC, and R_MAC. The Current Security Level shall be set as follows:

- NO_SECURITY_LEVEL when a Secure Channel Session is terminated or not yet initiated;
- AUTHENTICATED and C_MAC after a successful verification of the C- MAC of the first command message with a C-MAC. The Session Security Level shall be set to AUTHENTICATED only. C_MAC shall be cleared after the reception of the first command message without a C-MAC. AUTHENTICATED and C_MAC shall be cleared once the Secure Channel Session is terminated;
- R_MAC after a successful processing of a BEGIN R-MAC SESSION command: R-MAC shall be cleared after a successful processing of an END R-MAC SESSION command.

E.1.6 Protocol Rules

In accordance with the general rules described in chapter 10 - *Secure Communication*, the following protocol rules apply to Secure Channel Protocol ‘02’ with explicit initiation mode:

- The successful explicit initiation of a Secure Channel Session shall set the Current Security Level to the security level value indicated in the EXTERNAL AUTHENTICATE command: it is at least set to AUTHENTICATED;
- The Current Security Level shall apply to the entire Secure Channel Session unless successfully modified at the request of the Application;
- When the Current Security Level is not set to AUTHENTICATED (i.e. equal to NO_SECURITY_LEVEL or R-MAC only – case of a R-MAC session in progress originally initiated with a BEGIN R-MAC SESSION command):

  - If the Secure Channel Session was aborted during the same Application Session, the incoming command shall be rejected with a security error;
  - Otherwise no security verification of the incoming command shall be performed. The Application processing the command is responsible for applying its own security rules.

- If a Secure Channel Session is active for incoming commands (i.e. Current Security Level at least set to AUTHENTICATED), the security of the incoming command shall be checked according to the Current Security Level regardless of the command secure messaging indicator:
  - When the security of the command does not match (nor exceeds) the Current Security Level, the command shall be rejected with a security error, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;
  - If a security error is found, the command shall be rejected with a security error, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;
  - In all other cases, the Secure Channel Session shall remain active and the Current Security Level unmodified. The Application is responsible for further processing the command.
- If the Security Domain supports application data encryption and/or decryption, it shall decrypt or encrypt a block of secret data upon request. If the service is not supported or if (one of) the appropriate cryptographic key(s) is not available, the request shall be rejected but the Current Security Level, Session Security Level and Secure Channel Session in operation (if any) shall not be impacted;
- If a Secure Channel Session is aborted, it is still considered not terminated;
- The current Secure Channel Session shall be terminated (if aborted or still open) and the Current Security Level reset to NO_SECURITY_LEVEL on either:
  - Attempt to initiate a new Secure Channel Session (new INITIALIZE UPDATE command);
  - Termination of the Application Session (e.g. new Application selection);
  - Termination of the associated logical channel;
  - Termination of the Card Session (card reset or power off);
  - Explicit termination by the Application (e.g. invoking GlobalPlatform API).

In accordance with the general rules described in chapter 10 - *Secure Communication*, the following protocol rules apply to Secure Channel Protocol ‘02’ with implicit initiation mode:

- The successful implicit initiation of a Secure Channel Session for incoming commands shall set the Current Security Level to AUTHENTICATED and C-MAC (the R-MAC indicator remaining as is);
- The Session Security Level for incoming commands is implicit and set to AUTHENTICATED for the entire Secure Channel Session. It is reset on the normal termination or abort of a Secure Channel Session for incoming commands;
- When the Current Security Level is not set to AUTHENTICATED (i.e. equal to NO_SECURITY_LEVEL or R-MAC only), no Secure Channel Session is active for incoming commands:
  - If no secure messaging is indicated in the incoming command, no security verification of the command shall be performed. The Application processing the command is responsible to apply its own security rules;
  - If secure messaging is indicated, a new Secure Channel Session initiation for incoming commands shall be attempted and the security of the incoming command verified.

- If a Secure Channel Session is active for incoming commands (i.e. Current Security Level at least set to AUTHENTICATED), the security of the incoming command shall be checked according to the Current Security Level:
  - If no secure messaging is indicated in the command, the Secure Channel Session shall remain active, the Current Security Level set to AUTHENTICATED (the R-MAC indicator remaining as is) and no further security verification of the command performed. The Application processing the command is responsible to apply its own security rules;
  - If secure messaging is indicated and the Current Security Level is set to AUTHENTICATED and C-MAC (ignoring the R-MAC indicator), the security of the incoming command shall be verified:

    If a security error is found, the command shall be rejected with a security error, the Secure Channel Session aborted and Current Security Level reset to NO_SECURITY_LEVEL or R-MAC only (in case of a R-MAC session in progress);

    Otherwise, the Secure Channel Session shall remain active and the Current Security Level unmodified. The Application is responsible for further processing the command.
  - If secure messaging is indicated and the Current Security Level is set to AUTHENTICATED (ignoring the R-MAC indicator), the current Secure Channel Session for incoming commands shall be terminated and the Current Security Level reset to NO_SECURITY_LEVEL or R-MAC only (in case of a R-MAC session in progress). A new Secure Channel Session initiation for incoming commands shall be attempted and the security of the incoming command verified.
- The current Secure Channel Session shall be terminated (if aborted or still open) and the Current Security Level reset to NO_SECURITY_LEVEL on either:
  - Termination of the Application Session (e.g. new Application selection);
  - Termination of the associated logical channel;
  - Termination of the Card Session (card reset or power off);
  - Explicit termination by the Application (e.g. invoking GlobalPlatform API).

E.2 Cryptographic Keys

A Security Domain, including the Issuer Security Domain, shall have at least one key set to be used in the initiation and use of a Secure Channel. This key set shall contain at least one double length DES key, the Secure Channel base key. This Secure Channel base key is only used to generate session keys during the initiation of a Secure Channel.

Key	Usage	Length	Remark
Secure Channel base key	Secure Channel authentication, MAC verification, & sensitive data decryption (DES)	16 bytes	Mandatory

Table E-2: SCP02 - Security Domain Secure Channel Base Key

A Security Domain's, including the Issuer Security Domain's, Secure Channel key set may contain 3 double length DES keys: the Secure Channel encryption key (S-ENC), the Secure Channel MAC key (S-MAC), and the data encryption key (DEK) for decrypting sensitive data, e.g. secret or private keys. These keys are only used to generate session keys during the initiation of a Secure Channel.

Key	Usage	Length	Remark
Secure Channel encryption key (S-ENC)	Secure Channel authentication & encryption (DES)	16 bytes	Mandatory

Key	Usage	Length	Remark
Secure Channel message authentication code key (S-MAC)	Secure Channel MAC verification and generation (DES)	16 bytes	Mandatory
Data encryption key (DEK)	Sensitive data decryption (DES)	16 bytes	Mandatory

Table E-3: SCP02 - Security Domain Secure Channel Keys

E.3 Cryptographic Algorithms

The cryptographic and hashing algorithms described in appendix B - *Algorithms (Cryptographic and Hashing)* apply to SCP02. This section defines the additional requirements for SCP02.

E.3.1 Cipher Block Chaining (CBC)

The Initial Chaining Vector (ICV) used for chained data encryption in CBC mode is always 8 bytes of binary zero ('00').

The Initial Chaining Vector (ICV) used for chained message integrity in CBC mode is always 8 bytes and initialized during the Secure Channel initiation.

E.3.2 Message Integrity ICV using Explicit Secure Channel Initiation

When using explicit Secure Channel initiation, SCP02 mandates the use of a MAC on the EXTERNAL AUTHENTICATE command.

For the EXTERNAL AUTHENTICATE command MAC verification, the ICV is set to zero.

Once successfully verified, the MAC of the EXTERNAL AUTHENTICATE command becomes the ICV for the subsequent C-MAC verification and/or R-MAC generation.

E.3.3 Message Integrity ICV using Implicit Secure Channel Initiation

When using implicit Secure Channel Session initiation, the ICV shall be a MAC computed on the AID of the selected Application. The ICV for the first C-MAC calculation of a new Secure Channel Session is calculated as follows:

- Apply reversible padding to the AID of the selected Application as defined in appendix B.4 - *DES Padding*;
- Calculate a MAC as defined in appendix B.1.2.2 - *Single DES Plus Final Triple DES* with the C-MAC key over the padded Application AID with an ICV value of binary zeroes.

The resulting MAC is the ICV for the first C-MAC of the Secure Channel Session. This ICV makes the Secure Channel Session application specific.

The ICV for the first R-MAC of a Secure Channel Session is calculated the same way, except that the R-MAC key is used in step 2 for the MAC calculation.

E.3.4 ICV Encryption

As an enhancement to the C-MAC mechanism, the ICV is encrypted before being applied to the calculation of the next C-MAC. The encryption mechanism used is single DES with the first half of the Secure Channel C-MAC session key. The first ICV of a session is not encrypted.

E.4 Cryptographic Usage

E.4.1 DES Session Keys

DES session keys are generated every time a Secure Channel is initiated. These session keys may be used for subsequent commands if secure messaging is required.

Session keys are generated to ensure that a different set of keys is used for each secure communication session.

DES session keys are created using the static Secure Channel key(s), the current value of the Secure Channel Sequence Counter, a constant, and a padding of binary zeroes. Creating session keys is performed as follows:

- Generating the Secure Channel C-MAC session keys using the Secure Channel base key or MAC key (S-MAC) and the session keys derivation data with a constant of '0101';
- Generating the Secure Channel R-MAC session keys using the Secure Channel base key or MAC key (S-MAC) and the session keys derivation data with a constant of '0102';
- Generating the Secure Channel encryption session keys using the Secure Channel base key or encryption key (S-ENC) and the session keys derivation data with a constant of '0182';
- Generating the Secure Channel data encryption session keys using the Secure Channel base key or data encryption key (DEK) and the session keys derivation data with a constant of '0181'.

The DES operation used to generate these keys is always triple DES in CBC mode.

R-MAC session keys are generated using the current value of the sequence counter at the time the R-MAC session is initiated. If the R-MAC session starts with the EXTERNAL AUTHENTICATE command, the same sequence counter value is used to generate the R-MAC session key as is used to generate the other session keys for the same Secure Channel Session.

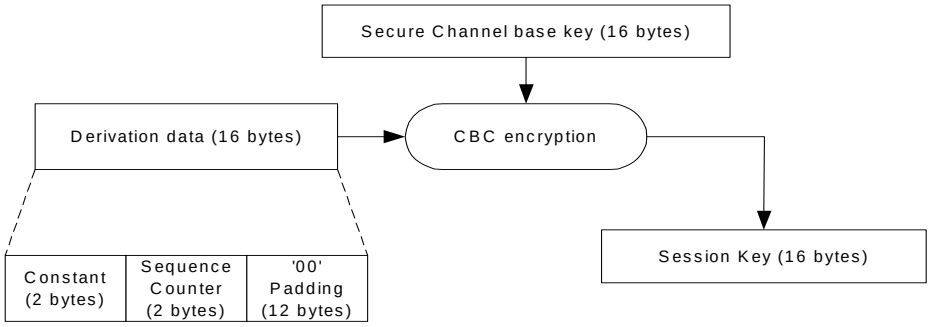

Figure E-2: Create Secure Channel Session Key from the Base Key

E.4.2 Authentication Cryptograms in Explicit Secure Channel Initiation

Both the card and the off-card entity (host) generate an authentication cryptogram. The off-card entity verifies the card cryptogram and the card verifies the host cryptogram. Generating or verifying an authentication cryptogram uses the S-ENC session key and the signing method described in appendix B.1.2.1 - *Full Triple DES*.

E.4.2.1 Card Authentication Cryptogram

The generation and verification of the card cryptogram is performed by concatenating the 8-byte host challenge, 2-byte Sequence Counter, and 6-byte card challenge resulting in a 16-byte block.

Applying the same padding rules defined in appendix B.4 - *DES Padding*, the data shall be padded with a further 8-byte block ('80 00 00 00 00 00 00 00').

The signature method, using the S-ENC session key and an ICV of binary zeroes, is applied across this 24-byte block and the resulting 8-byte signature is the card cryptogram.

E.4.2.2 Host Authentication Cryptogram

The generation and verification of the host cryptogram is performed by concatenating the 2-byte Sequence Counter, 6-byte card challenge, and 8-byte host challenge resulting in a 16-byte block.

Applying the same padding rules defined in appendix B.4 - *DES Padding*, the data shall be padded with a further 8-byte block ('80 00 00 00 00 00 00 00').

The signature method, using the S-ENC session key and an ICV of binary zeroes, is applied across this 24-byte block and the resulting 8-byte signature is the host cryptogram.

E.4.2.3 Card Challenge

The card challenge is either a random or pseudo-random number that shall be unique to a Secure Channel Session. A pseudo-random card challenge may be generated as follows:

- The AID of the currently selected Application is padded according to the padding rules defined in appendix B.4 - *DES Padding*;
- A MAC is calculated across the padded data as defined in appendix B.1.2.2 - Single DES Plus Final Triple DES MAC, using the C-MAC session key and an ICV of binary zeroes;
- The six leftmost bytes of the resultant MAC constitute the card challenge.

E.4.3 Authentication Cryptogram in Implicit Secure Channel Initiation

When using the implicit Secure Channel Session initiation, the authentication cryptogram is the first C-MAC received by the card from the off-card entity.

The rules for C-MAC generation and verification are detailed in the following subsection.

E.4.4 APDU Command C-MAC Generation and Verification

A C-MAC is generated by an off-card entity and applied across the full APDU command being transmitted to the card including the header and the data field in the command message. It does not include Le.

C-MAC generation and verification uses the Secure Channel C-MAC session key, an ICV and the signature method described in appendix B.1.2.2 - *Single DES Plus Final Triple DES*. (Prior to using the ICV, the ICV can be encrypted as described in appendix E.3.4 - *ICV Encryption*)

The ICV is used to chain the commands for command sequence integrity; the initial value of the ICV is described in appendix E.3 - *Cryptographic Algorithms*. For any subsequent command following the first successful C-MAC verification, the ICV is the C-MAC value successfully verified for the previous command received by the card.

The signature method, using the Secure Channel C-MAC session key and the ICV, is applied across the padded command message and the resulting 8-byte signature is the C-MAC. The rules for C-MAC padding are as defined in appendix B.4 - *DES Padding.*

To reflect the presence of a C-MAC in the command message, the command header shall be modified as follows:

- The length of the command message (Lc) shall be incremented by 8 to indicate the inclusion of the C-MAC in the data field of the command message;
- The class byte shall be modified to indicate that this APDU command includes secure messaging. This is achieved by either setting to '1' bit 3 of a class byte indicating a logical channel number 0 to 4 (unprotected CLA set to '00' - '03' or '80' - '83') or by setting to '1' bit 6 of a class byte indicating a logical channel number 4 to 19 (unprotected CLA set to '40' - '4F' or 'C0' - 'CF'): see section 11.1.4. The C-MAC generation and verification does not reflect the logical channel information included in the class byte of the command: the logical channel number is always assumed to be zero in the generation or verification of the C-MAC.

The C-MAC is appended at the end of the APDU command message excluding any padding but including the modifications made to the command header (class and Lc).

The two following methods are defined for the C-MAC generation:

- C-MAC generation on unmodified APDU: padding of the APDU command is required prior to the C-MAC generation being performed, and modification of the APDU command header is required after the C-MAC generation has been performed;

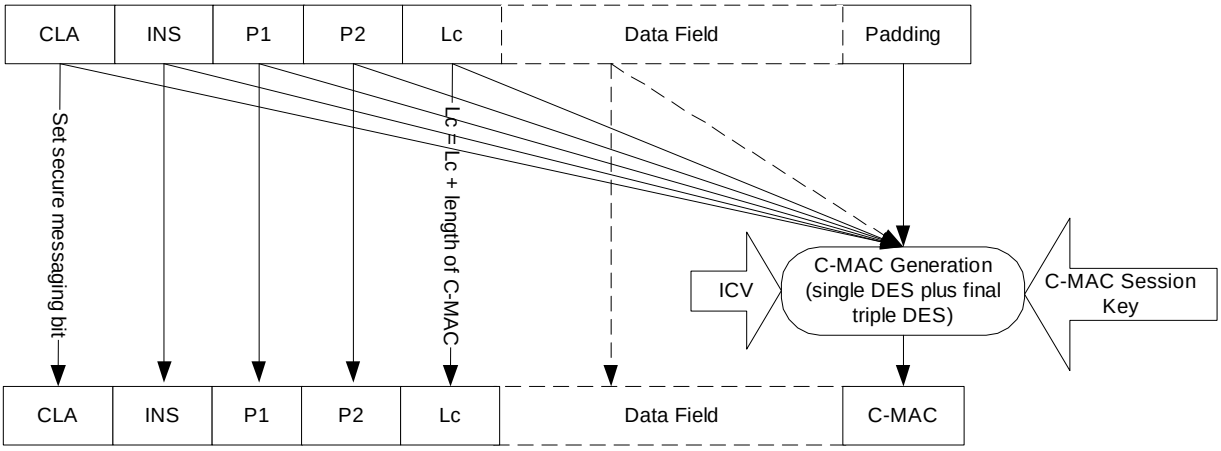

Figure E-3: C-MAC Generation on Unmodified APDU

- C-MAC generation on modified APDU: padding of the APDU command and modification of the command header is required prior to the C-MAC generation being performed.

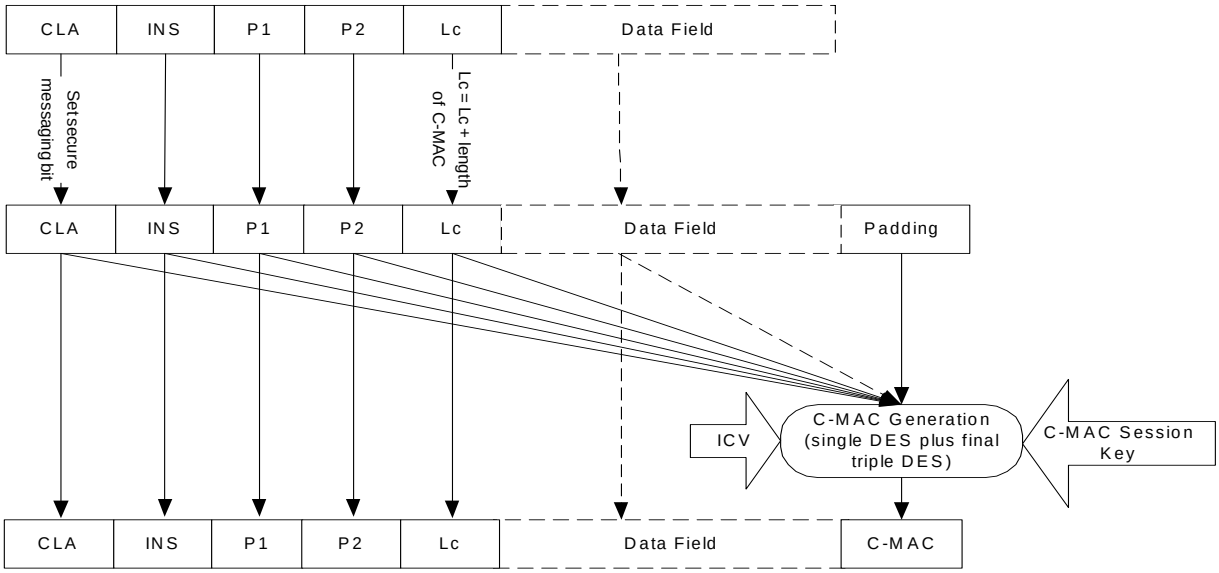

Figure E-4: C-MAC Generation on Modified APDU

If no other secure messaging is required, the message is now prepared for transmission to the card.

The card, in order to verify the C-MAC, shall perform the same padding mechanism to the data and use the same ICV and C-MAC session key employed by the off-card entity in order to verify the C-MAC. The verified C-MAC shall be retained and used as the ICV for any subsequent C-MAC verification. This is true regardless of whether the APDU completed successfully or not i.e. a verified C-MAC shall never be discarded in favor of a previous verified C-MAC value.

If the Secure Channel Session is occurring on a Supplementary Logical Channel, the logical channel number is added to the class byte of the message after the C-MAC has been generated and prior to the message being transmitted to the card. The card shall remove any logical channel information from the class byte (i.e. assume the logical channel number is zero) prior to verifying the C-MAC.

E.4.5 APDU Response R-MAC Generation and Verification

When using explicit Secure Channel Session initiation, the off-card entity instructs the card whether R-MAC generation applies to all the subsequent command/response messages. The APDU response message integrity does not include the EXTERNAL AUTHENTICATE command/response pair and lasts until the end of the Secure Channel Session.

When using implicit Secure Channel Session initiation, the off-card entity instructs the card to start APDU response message integrity using the BEGIN R-MAC SESSION command (see appendix E.5.3 - *BEGIN R-MAC SESSION Command*). The APDU response message integrity includes the BEGIN R-MAC SESSION command/response pair and lasts until reception of the END R-MAC SESSION command (see appendix E.5.4 - *END R-MAC SESSION Command*).

The R-MAC is computed on the following data block:

- The stripped APDU command message, i.e. without any C-MAC and modified command header (the logical channel number is always assumed to be zero). In the case of a case 1 or case 2 command, Lc is always present and set to zero;

- In case of a successful execution or a warning, the response data preceded with a byte Li that codes its length modulo 256. Li is generated by the Security Domain. If there is no response data this byte is present and is set to zero;
- In case of an error, a byte set to '00' indicating no response data;
- The status bytes.

When an R-MAC is returned in response to every command, in the event of case 1 and case 3 commands received by the card, these commands shall be processed respectively as case 2 and case 4 commands with Le set to zero.

The signature method, using the Secure Channel R-MAC session key and the ICV, is applied across the padded data block and the resulting 8-byte signature is the R-MAC. The rules for R-MAC padding are as defined in appendix B.4 - *DES Padding.*

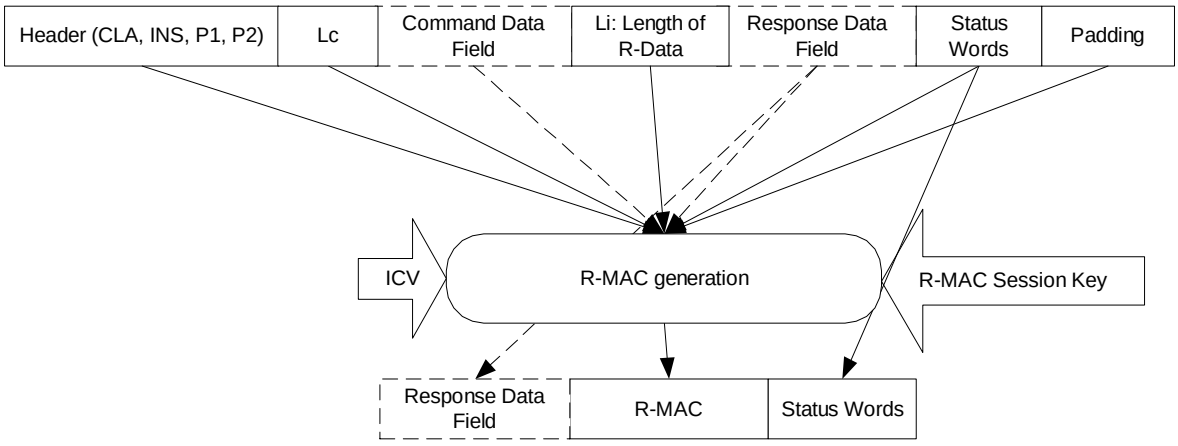

Figure E-5: R-MAC Generation

The R-MAC can be retrieved from the card by sending an END R-MAC SESSION command. The END R-MAC SESSION command allows the off-card entity to instruct the card to either terminate or continue the R-MAC session. The END R-MAC SESSION command is not included in the R-MAC computation.

The off-card entity, in order to verify the R-MAC, shall perform the same padding mechanism to the command/response pair and use the same ICV and R-MAC session key employed by the card in order to verify the R-MAC. The generated R-MAC shall be retained and used as the ICV for any subsequent R-MAC generation. (This is true regardless of whether the APDU command completed successfully or not.)

If the Secure Channel Session is occurring on a Supplementary Logical Channel, the logical channel information is not included in any of the R-MAC computations. The R-MAC is generated with a logical channel number assumed to be zero and the off-card entity is either not aware of, or removes, any logical channel information from the class byte of the corresponding command when verifying the R-MAC.

E.4.6 APDU Command Data Field Encryption and Decryption

Depending on the Session Security Level defined in the explicit initiation of the Secure Channel, all subsequent APDU commands within the Secure Channel may require secure messaging and as such the use of a C-MAC (integrity) and encryption (confidentiality).

Note that, with an implicit initiation of the Secure Channel, command message data field encryption does not apply.

If confidentiality is required, the off-card entity encrypts the data field of the command message being transmitted to the card. This excludes the header and the C-MAC but includes any data within the data field that has been protected for another purpose e.g. secret or private keys encrypted with the data encryption session key.

Command message encryption and decryption uses the Secure Channel encryption session key and the encryption method described in appendix B.1.1.1 – *Encryption/Decryption CBC Mode*.

Prior to encrypting the data, the data shall be padded. Unlike the C-MAC, this padding now becomes part of the data field and this necessitates further modification of the Lc value.

Padding of the data field to be encrypted is performed according to appendix B.4 - *DES Padding*.

The number of bytes appended to the data field in order to fulfill the above padding shall be added to Lc. This includes the mandatory '80' and any optional binary zeroes.

The encryption can now be performed across the padded data field using the Secure Channel encryption session key and the result of each encryption becomes part of the encrypted data field in the command message.

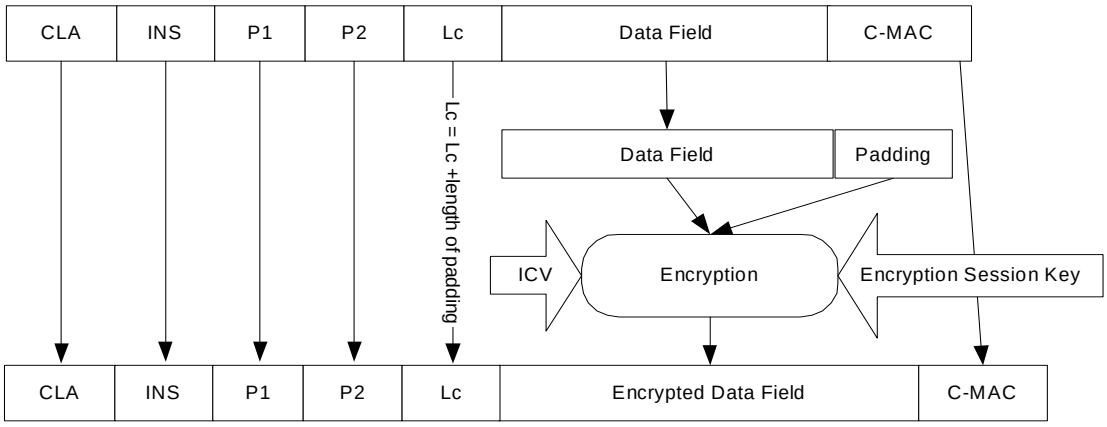

Figure E-6: APDU Command Data Field Encryption

Note: The ICV and the chaining are only used to link the blocks of the data currently being encrypted. For this reason the ICV for command data encryption is always binary zeroes.

The message is now prepared for transmission to the card. The card is required to first decrypt the command message and strip off any padding prior to attempting to verify the C-MAC. This decryption uses an ICV of binary zeroes and the same encryption session key employed by the off-card entity. The padding shall be removed and Lc shall be modified to reflect the length prior to encryption i.e. original clear text data plus C-MAC length.

E.4.7 Sensitive Data Encryption and Decryption

Data encryption is used when transmitting sensitive data to the card and is over and beyond the Current Security Level required for the Secure Channel; For instance all DES keys transmitted to a card (e.g. in a PUT KEY command) should be encrypted.

The data encryption process uses the data encryption session key and the encryption method described in appendix B.1.1.2 – *Encryption/Decryption ECB Mode* when using explicit initiation of the Secure Channel and appendix B.1.1.1 – *Encryption/Decryption CBC Mode* when using implicit initiation of the Secure Channel.

As all DES keys are by their very nature a multiple of 8-byte lengths no padding is required for key encryption operations. Similarly the sensitive data block length shall be constructed as a multiple of 8-byte long block before the data encryption operations: the eventual padding method is application specific.

The encryption is performed across the sensitive data and the result of each encryption becomes part of the encrypted data. This encrypted data becomes part of the “clear text” data field in the command message.

The on-card decryption of key data, is the exact opposite of the above operation: in particular, no padding is removed by the decryption operation.

E.5 Secure Channel APDU Commands

Command	Secure Channel Initiation	
	Explicit	Implicit
INITIALIZE UPDATE	✓	
EXTERNAL AUTHENTICATE	✓	
BEGIN R-MAC SESSION		
END R-MAC SESSION		

Table E-4: SCP02 Command Support

Table E-5 summarizes the minimum security requirements for the APDU commands.

Command	Minimum Security	
	Explicit	Implicit
INITIALIZE UPDATE	None	Dependent on the Issuer security policy
EXTERNAL AUTHENTICATE	C-MAC	
BEGIN R-MAC SESSION	Secure Channel initiation	
END R-MAC SESSION	Secure Channel initiation	

Table E-5: Minimum Security Requirements for SCP02 Commands

Note that Table E-4 shall be used in conjunction with Table E-6 to determine SCP02 command support requirements. The following table provides the list of SCP02 command support per card Life Cycle State.

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED		TERMINATED	
	AM SD	DM SD	SD	AM SD	DM SD	SD	AM SD	DM SD	SD	FA SD	SD	FA SD	SD
INITIALIZE UPDATE	✓	✓		✓	✓		✓	✓		✓			
EXTERNAL AUTHENTICATE	✓	✓		✓	✓		✓	✓		✓			
BEGIN R-MAC SESSION													
END R-MAC SESSION													

Table E-6: SCP02 Command Support per card Life Cycle State

Legend of Table E-4 and Table E-6*:*

AM SD: Security Domain with Authorized Management privilege.

DM SD: Supplementary Security Domain with Delegated Management privilege.

FA SD: Security Domain with Final Application privilege. (Note: command support for an Application with Final Application Privilege which is not a Security Domain is subject to Issuer policy, within the constraints of what is permitted according to the card Life Cycle State.)

SD: Other Security Domain.

✓ : Support required.

Blank cell: Support optional.

Striped cell: Support prohibited.

E.5.1 INITIALIZE UPDATE Command

E.5.1.1 Definition and Scope

The INITIALIZE UPDATE command is used, during explicit initiation of a Secure Channel, to transmit card and session data between the card and the host. This command initiates the initiation of a Secure Channel Session.

At any time during a current Secure Channel, the INITIALIZE UPDATE command can be issued to the card in order to initiate a new Secure Channel Session.

This INITIALIZE UPDATE command is not to be confused with commands of the same name in legacy applications.

E.5.1.2 Command Message

The INITIALIZE UPDATE command message is coded according to the following table:

Code	Value	Meaning
CLA	'80' - '83' or 'C0' - 'CF'	See section 11.1.4
INS	'50'	INITIALIZE UPDATE
P1	'xx'	Key Version Number
P2	'00'	Reference control parameter P2
Lc	'08'	Length of host challenge
Data	'xx xx…'	Host challenge
Le	'00'	

Table E-7: INITIALIZE UPDATE Command Message

E.5.1.3 Reference Control Parameter P1 - Key Version Number

The Key Version Number defines the Key Version Number within the Security Domain to be used to initiate the Secure Channel Session. If this value is zero, the first available key chosen by the Security Domain will be used.

E.5.1.4 Reference Control Parameter P2

The reference control parameter P2 shall always be set to '00'.

E.5.1.5 Data Field Sent in the Command Message

The data field of the command message contains 8 bytes of host challenge. This challenge, chosen by the off-card entity, should be unique to this session.

E.5.1.6 Response Message

The data field of the response message contains the concatenation without delimiters of the following data elements:

Name	Length
Key diversification data	10 bytes
Key information	2 bytes
Sequence Counter	2 bytes
Card challenge	6 bytes
Card cryptogram	8 bytes

Table E-8: INITIALIZE UPDATE Response Message

The key diversification data is data typically used by a backend system to derive the card static keys.

The key information includes the Key Version Number and the Secure Channel Protocol identifier, here '02', used in initiating the Secure Channel Session.

The Sequence Counter is an internal incremental counter used for creating session keys.

The card challenge is an internally generated random number.

The card cryptogram is an authentication cryptogram.

E.5.1.7 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or the following error condition:

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table E-9: INITIALIZE UPDATE Error Condition

E.5.2 EXTERNAL AUTHENTICATE Command

E.5.2.1 Definition and Scope

The EXTERNAL AUTHENTICATE command is used by the card, during explicit initiation of a Secure Channel, to authenticate the host and to determine the level of security required for all subsequent commands.

A successful execution of the INITIALIZE UPDATE command shall precede this command.

E.5.2.2 Command Message

The EXTERNAL AUTHENTICATE command message is coded according to the following table.

Code	Value	Meaning
CLA	'84' - '87' or 'E0' - 'EF'	See section 11.1.4
INS	'82'	EXTERNAL AUTHENTICATE
P1	'xx'	Security level
P2	'00'	Reference control parameter P2
Lc	'10'	Length of host cryptogram and MAC
Data	'xx xx…'	Host cryptogram and MAC
Le		Not present

Table E-10: EXTERNAL AUTHENTICATE Command Message

E.5.2.3 Reference Control Parameter P1 - Security Level

The reference control parameter P1 defines the level of security for all secure messaging commands following this EXTERNAL AUTHENTICATE command (it does not apply to this command) and within this Secure Channel Session.

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	0	1	1	0	0	1	1	RFU
0	0	1	1	0	0	0	1	RFU
0	0	1	1	0	0	0	0	RFU
0	0	0	1	0	0	1	1	C-DECRYPTION, C-MAC and R-MAC
0	0	0	1	0	0	0	1	C-MAC and R-MAC
0	0	0	1	0	0	0	0	R-MAC
0	0	0	0	0	0	1	1	C-DECRYPTION and C-MAC
0	0	0	0	0	0	0	1	C-MAC
0	0	0	0	0	0	0	0	No secure messaging expected.

Table E-11: EXTERNAL AUTHENTICATE Reference Control Parameter P1

E.5.2.4 Reference Control Parameter P2

The reference control parameter P2 shall always be set to '00'.

E.5.2.5 Data Field Sent in the Command Message

The data field of the command message contains the host cryptogram and the C-MAC.

E.5.2.6 Data Field Returned in the Response Message

The data field of the response message is not present.

E.5.2.7 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or the following warning condition:

SW1	SW2	Meaning
'63'	'00'	Authentication of host cryptogram failed

Table E-12: EXTERNAL AUTHENTICATE warning Code

E.5.3 BEGIN R-MAC SESSION Command

E.5.3.1 Definition and Scope

The BEGIN R-MAC SESSION command is used to initiate a Secure Channel Session for APDU response message integrity. At any time, the BEGIN R-MAC SESSION command may be issued to the card in order to initiate a R-MAC session.

E.5.3.2 Command Message

The BEGIN R-MAC SESSION command message is coded according to the following table:

Code	Value	Meaning
CLA	'80' - '87', 'C0' -'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'7A'	BEGIN R-MAC SESSION
P1	'xx'	Reference control parameter P1
P2	'01'	Reference control parameter P2
Lc	'xx'	Length of data field, if any
Data	'xx xx…'	BEGIN R-MAC SESSION data and C-MAC, if needed
Le		Not present

Table E-13: BEGIN R-MAC SESSION Command Message

E.5.3.3 Reference Control Parameter P1

The reference control parameter P1 defines the level of security for all subsequent APDU response messages following this BEGIN R-MAC SESSION command (it does not apply to this command).

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	0	1	1	0	0	0	0	Response Encryption and R-MAC (RFU)
0	0	0	1	0	0	0	0	R-MAC
0	0	0	0	0	0	0	0	No secure messaging expected

Table E-14: BEGIN R-MAC SESSION Reference Control Parameter P1

When P1 is set to '10' each APDU response message during the R-MAC session includes a R-MAC.

When P1 is set to '00' only the END R-MAC SESSION response message will contain a R-MAC.

E.5.3.4 Reference Control Parameter P2

The reference control parameter P2 defines the beginning of the session for APDU response message integrity.

b8	b7	b6	b5	b4	b3	b2	b1	Description
0	0	0	0	0	0	0	1	Begin R-MAC session

Table E-15: BEGIN R-MAC SESSION Reference Control Parameter P2

E.5.3.5 Data Field Sent in the Command Message

The data field of the BEGIN R-MAC SESSION contains an LV coded ‘data’ element and optionally a C–MAC. The card does not interpret the ‘data’. However since it is included in R-MAC calculation, this gives the off-card entity the possibility to include a challenge in the R-MAC.

The following table details the BEGIN R-MAC SESSION data field:

Length	Name	Presence
1	Length of data	Mandatory
0-24	data	Conditional

Table E-16: BEGIN R-MAC SESSION Command Data Field

E.5.3.6 Data Field Returned in the Response Message

The data field of the response message is not present.

E.5.3.7 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or the following error condition:

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table E-17: BEGIN R-MAC SESSION Error Conditions

E.5.4 END R-MAC SESSION Command

E.5.4.1 Definition and Scope

The END R-MAC SESSION command is used to terminate a Secure Channel Session for APDU response message integrity or to retrieve the current R-MAC without ending the R-MAC Session. The END R-MAC SESSION command may be issued to the card at any time during an R-MAC session, .

E.5.4.2 Command Message

The END R-MAC SESSION command message is coded according to the following table:

Code	Value	Meaning
CLA	'80' - '87', 'C0' -'CF' or 'E0' - 'EF'	See section 11.1.4
INS	'78'	END R-MAC SESSION
P1	'00'	Reference control parameter P1
P2	'01' or '03'	Reference control parameter P2
Lc	'xx'	Length of data field, if any
Data	'xx xx…'	C-MAC, if needed
Le	'00'	

Table E-18: END R-MAC SESSION Command Message

E.5.4.3 Reference Control Parameter P1

Reference control parameter P1 shall always be set to '00'.

E.5.4.4 Reference Control Parameter P2

The reference control parameter P2 is coded according to the following table:

b8	b7	b6	B5	b4	b3	b2	b1	Description
0	0	0	0	0	0	1	1	End R-MAC session & return R-MAC
0	0	0	0	0	0	0	1	Return R-MAC

Table E-19: END R-MAC SESSION Reference Control Parameter P2

E.5.4.5 Data Field Sent in the Command Message

The data field of the command message may optionally contain a C-MAC.

E.5.4.6 Data Field Returned in the Response Message

The data field of response message contains the R-MAC of the current R-MAC session.

E.5.4.7 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

This command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or the following error condition:

SW1	SW2	Meaning
'6A'	'88'	Referenced data not found

Table E-20: END R-MAC SESSION Error Conditions

F Secure Channel Protocol '10'

F.1 Secure Communication

Secure Channel Protocol ‘10’ (SCP10) supports up to 19 Supplementary Logical Channels.

F.1.1 SCP10 Secure Channel

SCP10 is a Secure Channel Protocol based on asymmetric cryptography and a public key infrastructure (PKI) for initiating a Secure Channel, and the use of symmetric session keys for secure messaging during subsequent operation of the Secure Channel Session. The Secure Channel Protocol operates between a Security Domain on the card (or the Issuer Security Domain) and an Off-Card Entity (OCE) which may be the Application Provider (or Card Issuer) or another party.

SCP10 provides the following three levels of security:

Authentication - in which the Security Domain authenticates the Off-Card Entity and the Off-Card Entity may authenticate the Security Domain. There are two aspects to this: Certificate Verification which involves establishing the validity of each other's public key (PK); and Entity Authentication which involves establishing that the other party in the Secure Channel Session is authentic owner of that public key.

Integrity and data origin authentication - in which the Security Domain and the Off-Card Entity ensure that the data being received from the other entity actually came from its claimed source in the correct sequence and has not been altered.

Confidentiality - in which confidential data is not viewable by an unauthorized entity.

A further level of security applies to specific sensitive data (e.g. cryptographic keys) which may be encrypted using a mechanism that is not part of the Secure Channel Session security.

The steps involved in initiating a Secure Channel Protocol are as follows:

1. **Certificate Verification (optional)** - the Security Domain and the Off-Card Entity (OCE) may need to traverse and verify a chain of certificates from established trust points down to each other's public key. The extent to which this is needed depends on what keys are currently validated by each party: the null condition is where they have both already validated each other's public key, in which case no explicit Certificate Verification is necessary;
2. **Entity Authentication** - the Security Domain and optionally the Off-Card Entity further check the authenticity of the other party by verifying the signature of a challenge sent to the other party;
3. **Session key establishment** - the two parties establish symmetric session keys for subsequent secure messaging.

Only explicit Secure Channel initiation is supported in SCP10. There is no specific command to terminate a session.

Some of the variations on the Secure Channel Protocol supported by the card or the Security Domain are announced in the parameter "i" in Card Recognition Data or Security Domain Management Data. See appendix H - *GlobalPlatform Data Values and Card Recognition Data* for details of Card Recognition Data and Security Domain Management Data. The value "i" is coded on one byte as a bit map as follows:

b8	b7	b6	b5	b4	b3	b2	b1	Description
Not available	0	0	0	0	0	-	0	Key Transport

b8	b7	b6	b5	b4	b3	b2	b1	Description
Not available	0	0	0	0	0	-	1	Key Agreement
Not available	0	0	0	0	0	0	-	Signature with message recovery
Not available	0	0	0	0	0	1	-	Signature without message recovery

Table F-1: Values of Parameter "i"

Note: "i" is a subidentifier within an object identifier, and bit b8 is reserved for use in the structure of the object identifier according to ISO/IEC 8825-1.

Key transport and key agreement relate to the process of establishing session keys for the Secure Channel Session.

- With **key agreement** the Security Domain and the Off-Card Entity exchange secret values when the Secure Channel is being initiated, and session keys are then derived from those secrets using an algorithm known to both the Off-Card Entity and the Security Domain;
- With **key transport** the Security Domain receives session keys to be used for the Secure Channel Session from the Off-Card Entity during Secure Channel initiation.

Signature with message recovery and signature without message recovery refer to the signature scheme used for digital signatures in data messages during Entity Authentication. (Note that these mechanisms apply also with the signatures on digital certificates, but the value "i" does not refer to this.)

- With **signature with message recovery**, part or all of the message data that is signed is contained in the signature block and is recovered during the process of verifying the signature. Signature with message recovery is standardized in ISO 9796-2;
- With **signature without message recovery**, the signature does not contain any part of the message data that is signed, but comprises an appendix to the complete message. Such a scheme is also known as a signature scheme 'with appendix'. Signature without message recovery is standardized in PKCS#1, and is also used in X.509.

The Security Domain may support any combination of values of parameter "i", but shall support at least one of the following options as defined by "i":

- "i" = '01': Signature with recovery, key agreement;
- "i" = '02': Signature without recovery, key transport.

There is no requirement for a Security Domain to support more than one certificate format.

F.1.2 Certificate Verification

F.1.2.1 Overview

The Security Domain verifies a certificate of the Off-Card Entity to establish the validity of its public key. A certificate must be issued by the Trust Point for External Authentication (TP_EX) or a subordinate key authority of the TP_EX. On the other hand, the Off-Card Entity establishes the validity of the Security Domain's public key by verifying a certificate issued by the Key Authority of the Security Domain. The Security Domain shall hold a single (default) public key of the Trust Point for External Authentication (PK.TP_EX.AUT) and may also hold the validated public keys of other authorities in a certificate chain.

The Off-Card Entity may know implicitly what other public keys have already been validated by the Security Domain, and can use this knowledge to reduce the number of certificates the Security Domain is asked to verify, potentially down to zero.

By default the Security Domain expects the first certificate presented for verification to be signed by the PK.TP_EX.AUT, and each subsequent certificate to be signed by the public key validated with the previous

certificate. The Off-Card Entity may request the Security Domain to use a different default initial public key by indicating it with a MANAGE SECURITY ENVIRONMENT command.

The Off-Card Entity establishes the validity of the Security Domain's public key by checking a (chain of) certificate(s).

The minimum set of keys and certificates to be stored by a Security Domain that supports asymmetric Secure Channel Protocol '10' shall be as follows:

- One Public Key for Trust Point for External Authentication (PK.TP_EX.AUT);
- One Security Domain Private Key (SK.SD.AUT);
- One Security Domain Public Key (PK.SD.AUT); and
- One Security Domain Certificate (CERT.SD.AUT) corresponding to the Security Domain Public Key and signed by the Security Domain Key Authority.

The Security Domain may also hold:

- The Off-Card Entity's Public Key(s) (PK.OCE.AUT) for External Authentication which has been validated;
- The Key Authority's Public Key(s) (PK.KA_EX.AUT) for External Authentication in a valid certificate chain from the Trust Point for External Authentication to the Off-Card Entity; and
- The Key Authority's Certificate(s) for Internal Authentication (CERT.KA_IN.AUT) in a valid certificate chain from the Trust Point for Internal Authentication to the Security Domain.

The minimum set of keys and certificates to be available to an Off-Card Entity (OCE) that wishes to communicate with a specific Security Domain is assumed to be as follows:

- One Public Key for Trust Point for Internal Authentication (PK.TP_IN.AUT));
- One Off-Card Entity Public Key (PK.OCE.AUT);
- One Off-Card Entity Certificate (CERT.OCE.AUT) corresponding to the Off-Card Entity Public Key and signed by the Off-Card Entity's Key Authority;
- The Off-Card Entity Trust Point Certification Public Key (PK.TP_OCE.AUT) that participates in a chain of certificates down to the Security Domain's Public Key.

The Off-Card Entity may also hold:

- The Key Authority's Certificate(s) for External Authentication (CERT.KA_EX.AUT) in a valid certificate chain from the Trust Point for External Authentication to the Off-Card Entity.

Note that the Security Domain's owner may be considered to be the Application Provider (for a Security Domain) or the Card Issuer(for the Issuer Security Domain).

The first example in Figure F-1 is a simple case where the Trust Point for External Authentication (TP_EX) and the Trust Point for Internal Authentication (TP_IN) certify the public key of the Key Authority of the Off-Card Entity and the Security Domain respectively, and the Key Authority of the Off-Card Entity (KA_OCE) certifies the public key of the Off-Card Entity.

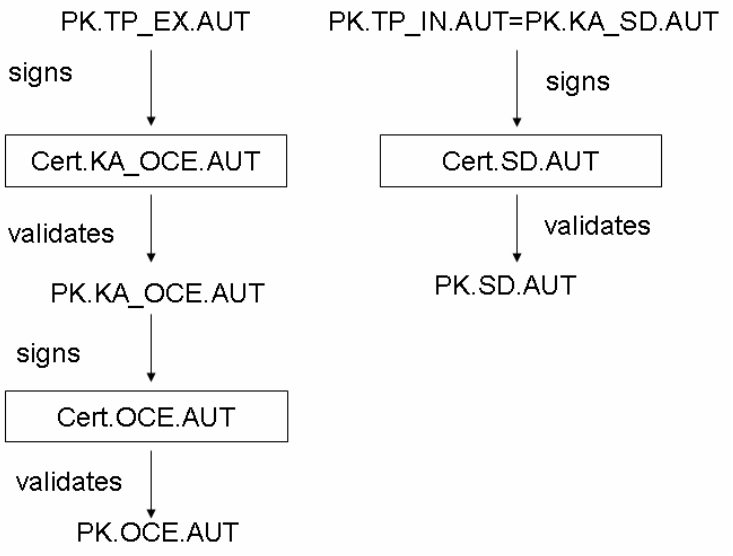

Figure F-1 - Certificate Chains - Example a

The second example in Figure F-2 illustrates that the Trust Point for External Authentication (TP_EX) and Internal Authentication (TP_IN) directly certifies the public key of both the Off-Card Entity and the Security Domain's Key Authority; and the Security Domain's Key Authority in turn certifies the Security Domain's public key.

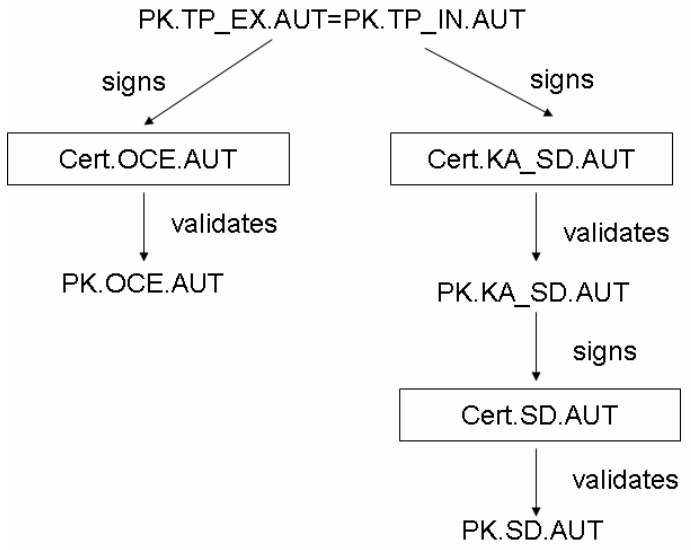

Figure F-2 - Certificate Chains - Example b

The third example in Figure F-3 illustrates that the Trust Point for External Authentication (TP_EX) and Internal Authentication (TP_IN) cross-certify each other's public keys, and there are two certificate chains down to the Security Domain and OCE public keys.

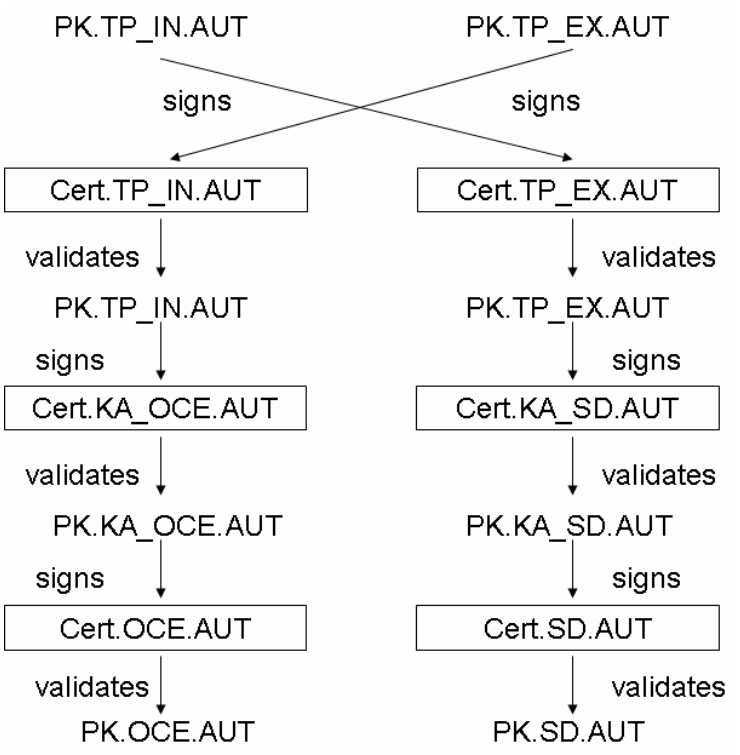

Figure F-3 Certificate Chains - Example c

F.1.2.2 Certificate Verification Process Flow

The following flow is an example of the Certificate Verification stage of Secure Channel initiation. Expanding on the authentication processing shown in the flow described in section 7.3.4 - *Runtime Messaging Support* it can be seen how an Application would use the services of a Security Domain to perform Certificate Verification.

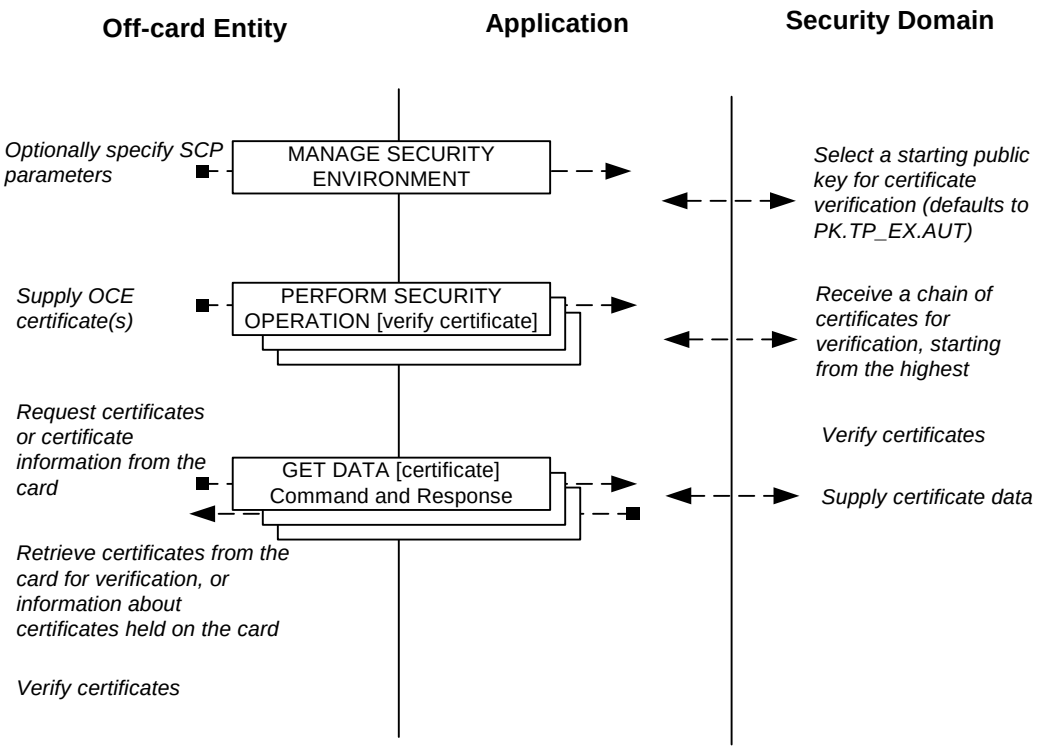

Figure F-4 - Certificate Verification Flow

The following commands shall be supported:

- MANAGE SECURITY ENVIRONMENT containing a 'cryptographic mechanism reference' designating the GlobalPlatform Secure Channel Protocol '10', see section F.4.5 for further details;
- PERFORM SECURITY OPERATION [verify certificate] command, see section F.4.7 for further details;
- GET DATA [certificate] command, see section F.4.3 for further details.

On receipt of a MANAGE SECURITY ENVIRONMENT command, any existing Secure Channel Session (on the same logical channel of the same card I/O interface) shall be terminated, regardless of the validity of the command: both the Current Security Level and Session Security Level are reset to NO_SECURITY_LEVEL. The MANAGE SECURITY ENVIRONMENT command may refer to a previously validated public key. If the MANAGE SECURITY ENVIRONMENT command is omitted, Security Domain default values and options shall apply, in particular for the initial default public key to use in subsequent command processing.

On receipt of a PERFORM SECURITY OPERATION [verify certificate] command not preceded by a MANAGE SECURITY ENVIRONMENT or another PERFORM SECURITY OPERATION [verify certificate] command, any existing Secure Channel Session (on the same logical channel of the same card I/O interface) shall be terminated, regardless of the result of the certificate verification: both the Current Security Level and Session Security Level are reset to NO_SECURITY_LEVEL.

Multiple PERFORM SECURITY OPERATION [verify certificate] commands may be received. In this case, a chain of certificates is presented to the Security Domain and the certificate contained in each command is verified using the Current Public Key. The Current Public Key is the public key that the Security Domain validated when verifying the last certificate presented by the Off-Card Entity during the same Secure Channel Session initiation. The Security

Domain shall have a default Current Public Key at the start of a Secure Channel Session initiation phase: Trust Point for External Authentication (PK.TP_EX.AUT).

Any failure in verifying an Off-Card Entity's certificate aborts the current Secure Channel Session initiation phase, and any public keys validated during that initiation phase shall be discarded.

Multiple GET DATA [certificate] commands may be received. They may be interleaved with PERFORM SECURITY OPERATION [verify certificate] commands. The Security Domain makes no assumptions about whether the Off-Card Entity has obtained enough certificates in order to validate the Security Domain's public key. The Off-Card Entity may use the GET DATA [certificate] command for EF.OD to retrieve information about the different certificates the Security Domain holds, see F.1.2.3 for further details on EF.OD.

F.1.2.3 Certificate Information

The Security Domain shall support access to EF.OD with a 'data object id' set to '5031'. EF.OD identifies each certificate that can be retrieved by the Off-Card Entity for verification. EF.OD contains a set of Cryptographic Information Objects (CIOs) as defined in ISO/IEC 7816-15, each of which describes a certificate and gives a pointer (in the form of a 'data object id') to the certificate data present within the Security Domain. The contents of EF.OD and its consistency with the certificates actually present within the Security Domain are the responsibility of the Security Domain's Provider and beyond the control of the card.

According to ISO/IEC 7816-15, an individual CIO for a certificate may be coded as follows (xxCertificate being of type CertificateChoice):

```
xxCertificate:
{
 commonObjectAttributes
      {
      label                Label              OPTIONAL
      flags                CommonObjectFlags OPTIONAL,
      authId               Identifier  OPTIONAL,
      userConsent          INTEGER (1..cia-ub-userConsent)       OPTIONAL,
      accessControlRules   SEQUENCE SIZE (1..MAX) OF AccessControlRule OPTIONAL,
      ….},
classAttributes
      {
      id             Identifier,
      authority      BOOLEAN     DEFAULT FALSE,
      identifier     CredentialIdentifier {{KeyIdentifiers}}       OPTIONAL,
      certHash       [0] CertHash    OPTIONAL,
      trustedUsage   [1] Usage OPTIONAL
      identifiers    [2] SEQUENCE OF CredentialIdentifier {{KeyIdentifiers}} OPTIONAL,
      validity       [4] Validity     OPTIONAL,
      ….},
 typeAttributes
      {
      value          ObjectValue {Certificate},
      ….}
}
```

where typeAttributes vary per type of CertificateChoice, for instance x509CertificateAttributes are:

```
typeAttributes
      {
      value          ObjectValue {Certificate},
      subject        Name             OPTIONAL,
```

```
issuer              [0] Name  OPTIONAL,
serialNumber    CertificateSerialNumber    OPTIONAL,
….}
```

F.1.2.4 Certificate Formats

Certificate contents and encoding are outside the scope of this specification. Examples are given for illustration only. Certificates may contain various data objects and elements as defined in ISO/IEC 7816-4, 7816-6 and 7816-8 (using application tagging), ITU Recommendation X.509 (universal and context-specific tagging) and elsewhere. The data objects included are defined by the certificate issuer (CA).

The following table identifies some data items that appear in certificates - 'presence' indicates when the item can be expected to be present:

Name as used in X.509	Name as used in 7816-8	Application Tag assigned by 7816	Presence
Issuer	Issuer Identification Number	'42' (7816-8)	Always
Subject	Certificate holder reference OR Cardholder name	'5F20' (7816-8)	Always
SubjectPublicKey	Cardholder public key	'5F49' or '7F49' (7816-8)	Always
KeyUsage	Certificate holder authorization	'5F4C' (7816-9)	As required
CertificateSerialNumber	Certificate serial number	-	Always
n/a	Certificate contents	'5F4E' (7816-8)	Non self descriptive certificates
Signature	Static internal authorization OR Digital signature	'9E' (7816-8)	Always
n/a	Public key remainder	'5F38' (7816-6)	Signature scheme with recovery

Table F-2: Example of Data Included in Certificates

Certificates may either be self descriptive, where the signature is across individual data objects; or non self descriptive, where the signature is across concatenated value fields, without tags and lengths. Whether the certificate to be verified by the Security Domain is self descriptive or non self descriptive must be indicated in the PERFORM SECURITY OPERATION [verify certificate] command.

A Security Domain is only required to handle a single certificate format throughout a chain of certificates to be verified by the Security Domain.

Tag '67' in Card Recognition Data or Security Domain Management Data may contain one or more OIDs identifying the Security Domain's Trust Point's certification policy and/or the format of certificates that can be verified by the

Security Domain and/or the format of certificates that can be retrieved from the Security Domain. They may also identify the cryptographic algorithms used for certificates, unless they are indicated in the certificates themselves.

F.1.2.4.1 Certificate without Message Recovery

The data and public key to be certified are concatenated and a digest is created. A signature block is then prepared containing the digest and any necessary padding, constant and random data. The format of the signature block is defined by the certificate issuer (CA). The signature block size depends on the asymmetric algorithm and the certifying key size. The signature block is then signed using the certificate issuer's private certifying key, to form the value field of the Certificate Signature data object. The result is a certificate typically containing the data and public key to be certified and the Certificate Signature.

Certificate without Message Recovery, Self Descriptive Certificate

The data objects to be certified, including the public key data object, are TLV encoded and are concatenated TLV-encoded before the digest is created and the signature block prepared. The result of the signing operation is the Certificate Signature. The certificate is TLV encoded as a series of data objects, containing all the data objects to be certified, including the public key data object and the Certificate Signature data object. X.509 certificates fall into this category.

The formation of the certificate is shown in the following example:

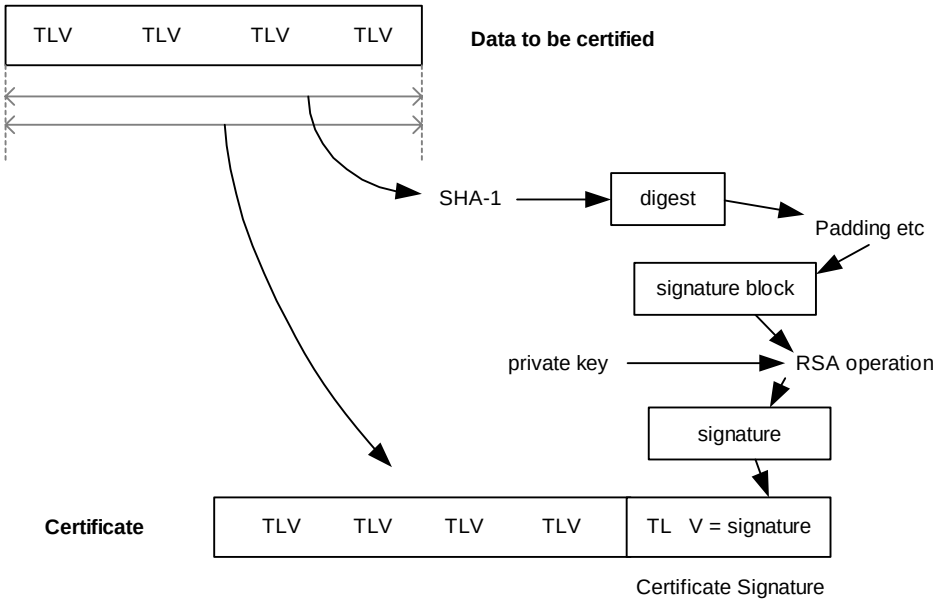

Figure F-5: Certificate Formation - Self Descriptive Certificate Without Message Recovery

Certificate without Message Recovery, Non-Self Descriptive Certificate

The data and public key to be certified are a concatenated set of value fields. The format of the value fields included is defined by the certificate issuer (CA) and implicitly known to the recipients. A digest of the concatenated value fields is created and the signature block prepared. The result of the signing operation is the Certificate Signature.

The certificate contains two data objects: the Certificate Contents, whose value is the concatenated set of value fields being certified, and the Certificate Signature.

The formation of the certificate is shown in the following example:

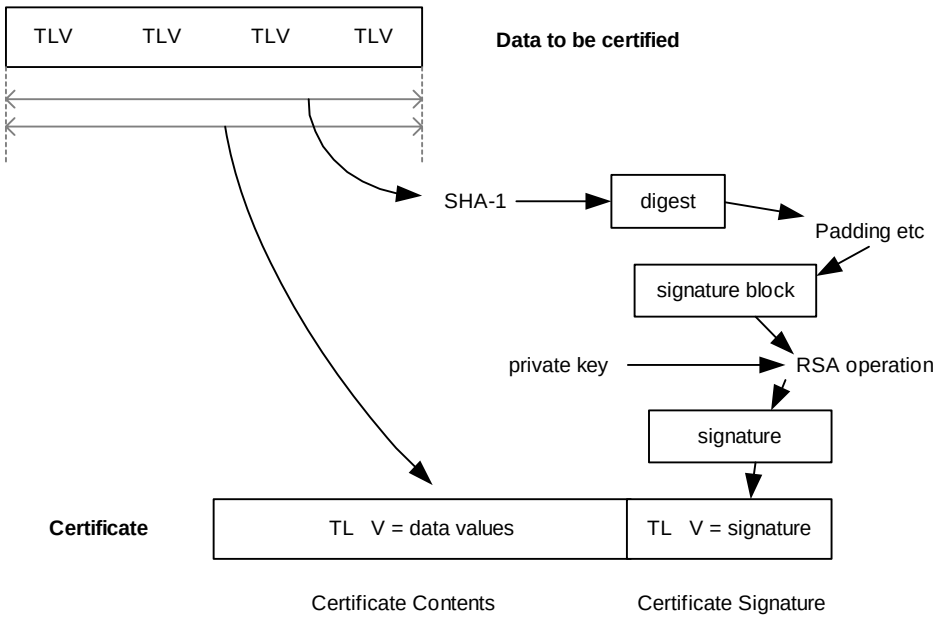

Figure F-6: Certificate Formation - Non Self Descriptive Certificate Without Message Recovery

F.1.2.4.2 Certificate with Message Recovery

The data and public key to be certified is concatenated and a digest is created. A signature block is then prepared containing the digest, as much of the concatenated set of data and public key to be certified as can be included in the block and any necessary padding, constant and random data. The format of the signature block is defined by the certificate issuer (CA). The signature block size depends on the asymmetric algorithm and the certifying key size. The signature block is then signed using the certificate issuer's private certifying key, to form the value field of the Certificate Signature data object. Any part of the public key that could not be included in the Certificate Signature is then included in the value field of a separate Public Key Remainder data object. The result is a certificate typically containing two data objects: the Public Key Remainder and the Certificate Signature.

Certificate with Message Recovery, Self Descriptive Certificate

The data objects to be certified, including the public key data object, are TLV encoded and are concatenated TLV-encoded before the digest is created and the signature block prepared. The result of the signing operation is the Certificate Signature.

The formation of the certificate is shown in the following diagram:

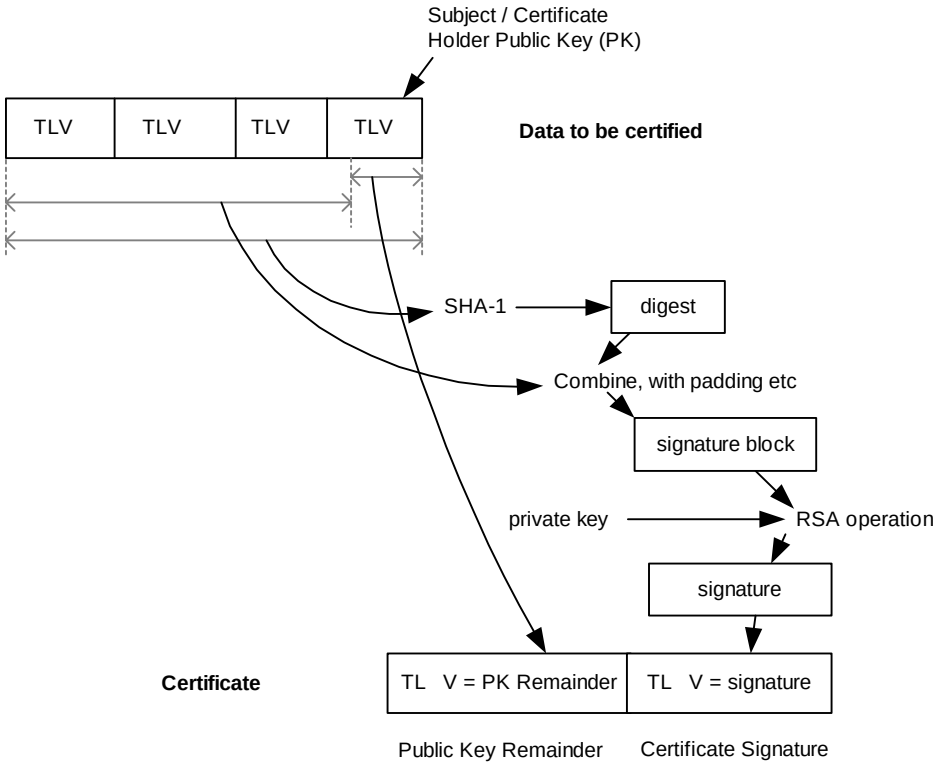

Figure F-7: Certificate Formation - Self Descriptive Certificate with Message Recovery

Certificate with Message Recovery, Non-Self Descriptive Certificate

The data and public key to be certified is a concatenated set of value fields. The format of the value fields included is defined by the certificate issuer (CA) and implicitly known to the recipients. A digest of the concatenated value fields is created and the signature block prepared. The result of the signing operation is the Certificate Signature.

The formation of the certificate is shown in the following diagram:

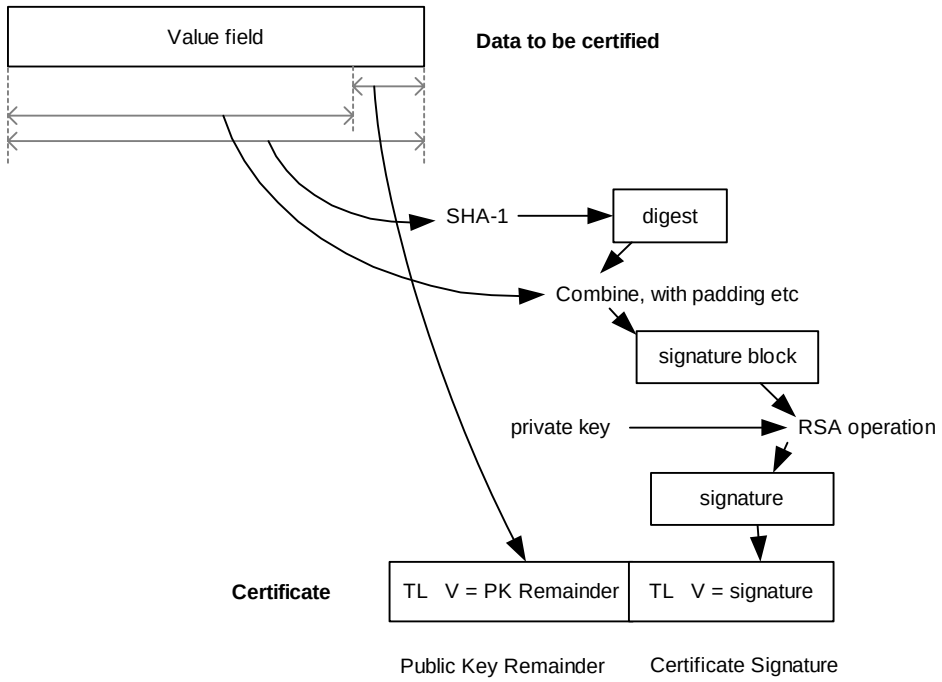

Figure F-8: Certificate Formation - Non Self Descriptive Certificate with Message Recovery

F.1.3 Entity Authentication

F.1.3.1 Overview of Entity Authentication

Once the parties have validated each other's public key, the Security Domain shall authenticate the Off-Card Entity using a challenge / response mechanism, and the Off-Card Entity may also authenticate the Security Domain in the same way.

Entity authentication of the Security Domain is optional, at the discretion of the Off-Card Entity. However, if the INTERNAL AUTHENTICATE command is required for use in session key agreement, then authentication of the Security Domain is performed.

F.1.3.2 Entity Authentication Process Flow

The following diagram gives an overview of the flow for Entity Authentication.

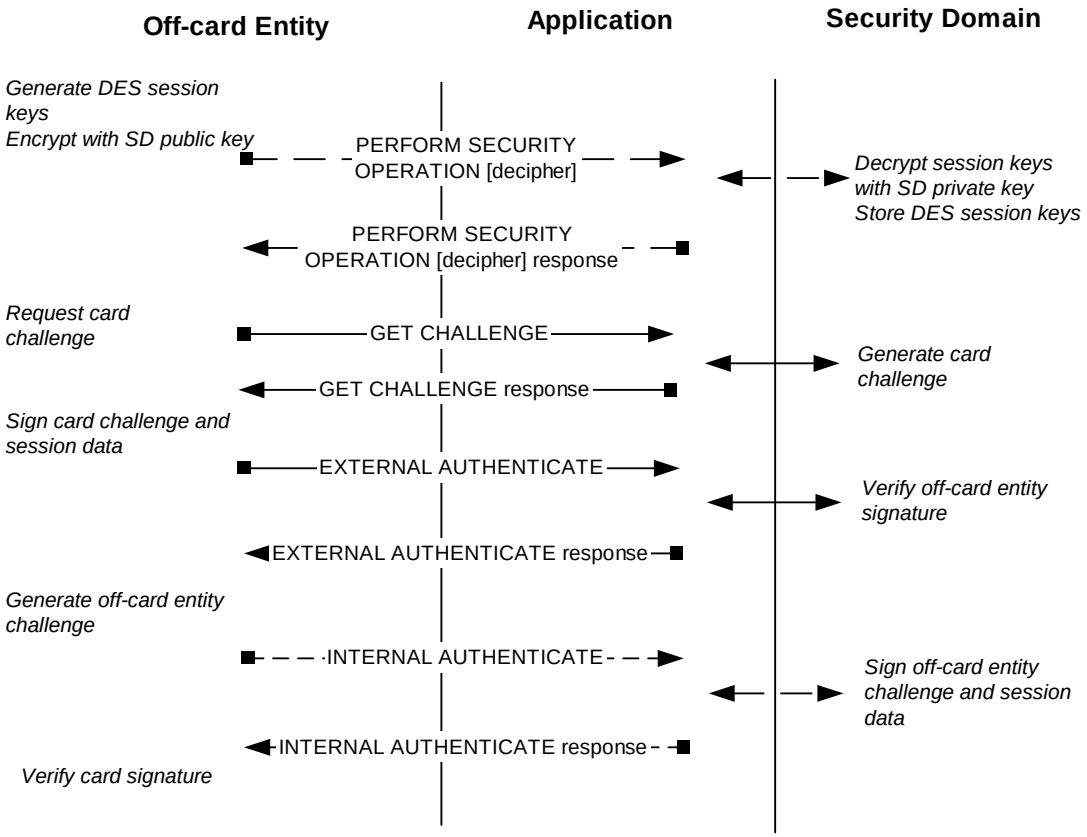

Figure F-9: Entity Authentication Flow

The following commands shall be supported:

- PERFORM SECURITY OPERATION [decipher] command, see section F.4.6 for further details;
- GET CHALLENGE command, see section F.4.2 for further details;
- EXTERNAL AUTHENTICATE command, see section F.4.1 for further details;
- INTERNAL AUTHENTICATE command, see section F.4.4 for further details.

The PERFORM SECURITY OPERATION [decipher] command is optional but shall be executed when value '02' of parameter "i" is supported - see section F.1.1.

The successful processing of the EXTERNAL AUTHENTICATE command (i.e. successful verification of the Off-Card Entity's signature) shall set both the Current Security Level and Session Security Level according to the rules described in section F.1.4.2 - *Security Level Establishment*. The failed processing of the EXTERNAL AUTHENTICATE command (i.e. unsuccessful verification of the Off-Card Entity's signature) shall reset both the Current Security Level and Session Security Level to NO_SECURITY_LEVEL and all the public keys previously verified within the current initiation phase shall be discarded.

The INTERNAL AUTHENTICATE command is optional for the key transport option and mandatory for key agreement. The INTERNAL AUTHENTICATE command shall only be received and processed once during Secure Channel Session initiation. Any error in the sequence flow or signing operation must abort the current Secure

Channel Session: both the Current Security Level and Session Security Level shall be reset to NO_SECURITY_LEVEL.

F.1.3.3 Security Domain Authentication

The format and contents of Security Domain signature vary depending on the options chosen, as follows:

Key transport, signature without message recovery

The Security Domain signature is the result of generating a digest (hash) over a set of data, creating a signature block, and signing the signature block with the Security Domain private key (SK.SD.AUT). The data to be hashed and the contents of the signature block are as shown below.

Name	Length	Value	Presence
Session Key(s)	n x (16 or 24)	'xxxx...'	Mandatory
Off-Card Entity challenge	16	'xxxx...'	Mandatory

Table F-3: Data to Hash

Session Keys shall be in the same order as provided in the PERFORM SECURITY OPERATION [decipher] command: see sections F.3.1.2 and F.4.7.

Name	Length	Value	Presence
Padding	2	'0001'	Mandatory
Padding ('FF')	8-n	'FF'...'FF'	Mandatory
Padding ('00')	1	'00'	Mandatory
DER encoded digest algorithm id - encoded as an object identifier	Variable	'xxxx...' (see section F.2.2)	Mandatory
DER encoded Hash (length and contents depend on the digest algorithm)	Variable	'xxxx...'	Mandatory

Table F-4: Security Domain Signature Block

Key agreement, signature with message recovery

The Security Domain signature is the result of generating a digest (hash) over a set of data, creating a signature block, and signing the signature block with the Security Domain private key (SK.SD.AUT). The data to be hashed and the contents of the signature block are as shown below.

Name	Length	Value	Presence
Random Padding (RP)	1-n	Same value as RP in Table F-6	Mandatory
Card Secret (CS)	32	Same value as CS in Table F-6	Mandatory
Off-Card Entity challenge	8	'xxxx...'	Mandatory
Off-Card Entity id	8	'xxxx...' (part of CERT.OCE.AUT)	Mandatory

Table F-5: Data to Hash

Name	Length	Value	Presence
Padding	1	'6A'	Mandatory

Name	Length	Value	Presence
Random Padding (RP)	1-n	Same value as RP in Table F-5	Mandatory
Card Secret (CS)	32	Same value as CS in Table F-5	Mandatory
Hash	20	'xxxx...'	Mandatory
Padding	1	'BC'	Mandatory

Table F-6: Security Domain Signature Block

The Security Domain signature is encrypted to ensure that the Card Secret is not divulged. To do this, the minimum of the values SIG.SD.AUT and (N.PK.SD.AUT – SIG.SD.AUT) is encrypted with the Off-Card Entity public key (PK.OCE.AUT), where N.PK.SD.AUT denotes the modulus of the Security Domain public key. This ensures that the data to be encrypted is always smaller than the modulus of the Off-Card Entity public key. Note that the modulus of the Security Domain public key and modulus of the Off-Card Entity public key must have the same length in bits. See ISO/IEC 9796-2, Digital Signature scheme 1.

F.1.3.4 Off-Card Entity Authentication

The format and contents of Off-Card Entity signature vary depending on the option chosen, as follows:

Key transport, signature without message recovery

The Off-Card Entity signature is the result of generating a digest (hash) over a set of data, creating a signature block, and signing the signature block with the Off-Card Entity private key (SK.OCE.AUT). The data to be hashed and the contents of the signature block are as shown below.

Name	Length	Value	Presence
Tag of Security Level	1	'D3'	Mandatory
Length of Security Level	1	'01'	Mandatory
Security Level	1	'xx'	Mandatory
Control reference template (CRT) tag	1	'B4' or 'B8'	Mandatory
Length of CRT	1	'00' - '7F'	Mandatory
CRT for Session Key(s)	n	'xxxx...'	Mandatory
…	…	…	…
CRT tag	1	'B4' or 'B8'	Optional
Length of CRT	1	'00' - '7F'	Conditional
CRT for Session Key(s)	n	'xxxx...'	Conditional
Card challenge	16		Mandatory

Table F-7: Data to Hash

The control reference templates (CRTs) include session keys, and must be in the same order as provided in the PERFORM SECURITY OPERATION [decipher] command - see Table F-32.

Name	Length	Value	Presence
Padding ('0001')	2	'0001'	Mandatory
Padding ('FF')	8-n	'FF'	Mandatory
Padding ('00')	1	'00'	Mandatory

Name	Length	Value	Presence
DER encoded digest algorithm id (encoded as an object identifier)	Variable	'xxxx...' (see section F.2.2)	Mandatory
DER encoded Hash (length and contents depend on the digest algorithm)	Variable	'xxxx...'	Mandatory

Table F-8: Off-Card Entity Signature Block

Control reference templates (CRTs) include session keys, and must be in the same order as provided in the PERFORM SECURITY OPERATION [decipher] command.

The format and contents of Card Signature vary depending on the options chosen, as discussed below.

Key agreement, signature with message recovery

The Off-Card Entity signature is the result of generating a digest (hash) over a set of data, creating a signature block, and signing the signature block with the Off-Card Entity private key (SK.OCE.AUT). The data to be hashed and the contents of the signature block are as shown below.

Name	Length	Value	Presence
Random Padding (RPD)	1-n	Same value as RPD in Table F-10	Mandatory
Tag of Security Level	1	'D3'	Mandatory
Length of Security Level	1	‘01’	Mandatory
Security Level	1	‘xx’	Mandatory
CRT tag	1	‘B4’ or ‘B8’	Mandatory
Length of CRT	1	‘00’ - ‘7F’	Mandatory
CRT for session key(s)	n	'xxxx...'	Mandatory
…	…	…	…
CRT tag	1	‘B4’ or ‘B8’	Optional
Length of CRT	1	‘00’ - ‘7F’	Conditional
CRT for Session Key(s)	n	'xxxx...'	Conditional
Off-Card Entity Secret (OES)	32	Same value as OES in Table F-10	Mandatory
Card challenge	8	'xxxx...'	Mandatory
Card id: TBD (part of CERT.SD.AUT)	8	'xxxx...'	Mandatory

Table F-9: Data to Hash

Control reference templates (CRTs) exclude session keys, and must be in the same order as provided in signature block.

Name	Length	Value	Presence
Padding	1	‘6A’	Mandatory
Random Padding (RPD)	1-n	Same value as RPD in Table F-9	Mandatory
Tag of Security Level	1	'D3'	Mandatory
Length of Security Level	1	‘01’	Mandatory

Name	Length	Value	Presence
Security Level	1	‘xx’	Mandatory
CRT tag	1	‘B4’ or ‘B8’	Mandatory
Length of CRT	1	‘00’ - ‘7F’	Mandatory
CRT for session key(s)	1-n	'xxxx...'	Mandatory
…	…	…	…
CRT tag	1	‘B4’ or ‘B8’	Optional
Length of CRT	1	‘00’ - ‘7F’	Conditional
CRT for Session Key(s)	n	'xxxx...'	Conditional
Off-Card Entity Secret (OES)	32	Same value as OES in Table F-9	Mandatory
Hash	20	'xxxx...'	Mandatory
Padding	1	‘BC’	Mandatory

Table F-10: Off-Card Entity Signature Block

Control reference templates (CRTs) exclude session keys, and must be in the same order as input to the hash function.

The Off-Card Entity signature is encrypted to ensure that the Off-Card Entity secret is not divulged. To do this, the minimum of the values SIG.OCE.AUT and (N.PK.OCE.AUT – SIG.OCE.AUT) is encrypted using the Security Domain public key (PK.SD.AUT). N.PK.OCE.AUT denotes the modulus of the Off-Card Entity public key. This ensures that the data to be encrypted is always smaller than the modulus of the Security Domain public key. Note that the modulus of the Security Domain public key and modulus of the Off-Card Entity public key must have the same length in bits. See ISO/IEC 9796-2, Digital Signature scheme 1.

F.1.4 Session Key and Security Level Establishment

When using the key transport option, Entity Authentication is preceded by the session keys and requested Security Level being sent to the Security Domain using the PERFORM SECURITY OPERATION [decipher] command; the Security Domain stores them until session initiation is complete.

A Security Domain supporting the key transport option shall decrypt with its private key (SK.SD.AUT or if the Security Domain has more than one key pair, the private key identified in the MANAGE SECURITY ENVIRONMENT command) the command data field of the PERFORM SECURITY OPERATION [decipher] command. Any failure in the decryption operation aborts the current Secure Channel Session initiation phase, and any public keys validated during that initiation phase shall be discarded.

In the key transport option the Secure Channel Session is established after successful processing of the EXTERNAL AUTHENTICATE command. An INTERNAL AUTHENTICATE command can be issued immediately after the EXTERNAL AUTHENTICATE command without secure messaging.

In the key agreement option the Secure Channel Session is established after successful processing of the EXTERNAL AUTHENTICATE and INTERNAL AUTHENTICATE commands.

F.1.4.1 Session Key Establishment

The Off-Card Entity supplies the Security Domain with details of what session keys are to be established. This information is in the form of 'control reference templates' (see section F.3.1.2) which are supplied either in the PERFORM SECURITY OPERATION [decipher] command (with the key transport option) or in the EXTERNAL AUTHENTICATE command (with the key agreement option).

If the key transport option is used, then the session keys are provided by the Off-Card Entity within the 'control reference templates' (see section F.3.1.2).

If the key agreement option is used, then the secrets exchanged between the Off-Card Entity and the Security Domain during the Entity Authentication process are used to establish session keys, as defined in section F.3.1 - *DES Session Keys*.

Once session keys have been established successfully, ICV sequence counter(s), used for secure messaging on subsequent commands and responses, are initialized as described in section F.3.2 - *Secure Messaging*.

F.1.4.2 Security Level Establishment

The requested Security Level is supplied in the PERFORM SECURITY OPERATION [decipher] command (with the key transport option) or in the EXTERNAL AUTHENTICATE command (with the key agreement option).

The successful initiation of a Secure Channel Session shall set the Current Security Level and Session Security Level to the requested Security Level combined with the AUTHENTICATED or ANY_AUTHENTICATED indicator (see section 10.4.2 - *Authentication with asymmetric cryptography* for further details). If the requested Security Level is set to zero, the successful initiation of a Secure Channel Session shall set the Current Security Level and Session Security Level to AUTHENTICATED or ANY_AUTHENTICATED only.

F.1.5 Protocol Rules

The Current Security Level of a communication not included in a Secure Channel Session shall be set to NO_SECURITY_LEVEL. In accordance with the general rules described in chapter 10 - *Secure Communication*, the following rules shall apply:

- The successful initiation of a Secure Channel Session shall set the Current Security Level to the requested Security Level from the selected Application's perspective: it is at least set to AUTHENTICATED or ANY_AUTHENTICATED (see section 10.4.2 - *Authentication with asymmetric cryptography* for details;
- The Current Security Level shall apply to the entire Secure Channel Session unless successfully modified at the request of the Application;
- When the Current Security Level is set to NO_SECURITY_LEVEL, then:
  - If the Secure Channel Session was aborted during the same Application Session, the incoming command shall be rejected with a security error;
  - Otherwise no security verification of the incoming command shall be performed. The Application processing the command is responsible for applying its own security rules.
- If a Secure Channel Session is active for incoming commands (i.e. Current Security Level at least set to either AUTHENTICATED or ANY_AUTHENTICATED), the security of the incoming command shall be checked according to the Current Security Level, or if the APDU class byte indicates Secure Messaging and Secure Messaging data objects are present in the command data field:
  - When the security of the command does not match or exceed the Current Security Level, the command shall be rejected with a security error, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;
  - If a security error is found, the command shall be rejected with a security error, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;
  - If (one of) the appropriate session key(s) is not available, the command shall be rejected with a security error, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;

  - In all other cases, the Secure Channel Session shall remain active and the Current Security Level shall reflect the level of security established by the current command (e.g. C-MAC and/or C-ENCRYPTION). The Application is responsible for further processing the command.

- If a Secure Channel Session is active for outgoing responses (i.e. Current Security Level at least set to AUTHENTICATED or ANY_AUTHENTICATED), secure messaging protection shall be applied to the outgoing response according to the Current Security Level (i.e. R-MAC and/or R-ENCRYPTION):
  - If a cryptographic error occurs, a security error shall be returned, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;
  - If (one of) the appropriate session key(s) is not available, a security error shall be returned, the Secure Channel Session aborted and the Current Security Level reset to NO_SECURITY_LEVEL;
  - Otherwise, the Secure Channel Session shall remain active and the Current Security Level unmodified.
- If a Secure Channel Session is aborted, it is still considered not terminated;
- If the Security Domain supports application data encryption and/or decryption, it shall decrypt or encrypt a block of secret data upon request. If the service is not supported or if (one of) the appropriate cryptographic key(s) is not available, the request shall be rejected but the Current Security Level, Session Security Level and Secure Channel Session in operation (if any) shall not be impacted;
- The current Secure Channel Session shall be terminated (if aborted or still open), both the Current Security Level and Session Security Level reset to NO_SECURITY_LEVEL on either:
  - Attempt to initiate a new Secure Channel Session;
  - Termination of the Application Session (e.g. new Application selection);
  - Termination of the associated logical channel;
  - Termination of the Card Session (card reset or power off);
  - Explicit termination by the Application (e.g. invoking GlobalPlatform API).

F.2 Cryptographic Algorithms

The cryptographic and hashing algorithms described in appendix B - *Algorithms (Cryptographic and Hashing)* apply to SCP10. This section defines the additional requirements for SCP10.

F.2.1 Asymmetric cryptography

In this appendix, signing means the deciphering of a signature block using the signer's private RSA key.

For certificates to be verified by the card, and message signatures signed and verified by the card, the cryptographic scheme shall be RSA. The signature block and signature are the same length as the key modulus.

In Entity Authentication, Digital Signature Scheme 1 in ISO/IEC 9796-2 shall be used for signature with message recovery, and the signature scheme with appendix RSASSA-PKCS1-V1_5 in PKCS#1 v2.1 for signature without message recovery.

The details of the signature scheme for certificates shall be either implicitly known by the Off-Card Entity or specified by the Security Domain Trust Point through the contents of Card Recognition Data or Security Domain Management Data in tag '67'.

In the key transport option, the cryptographic scheme for encrypting the session keys and their CRT templates shall be RSA according to the encryption scheme RSAES-PKCS1-v1_5 as defined in PKCS#1 v2.1.

F.2.2 Digest Algorithm

The default digest algorithm for use in conjunction with SCP10 asymmetric cryptography in Entity Authentication shall be SHA-1 for this version of the Specification. An alternative algorithm may be specified in Card Recognition Data or Security Domain Management Data in tag '67'.

The Object Identifier for SHA-1 is:

{iso(1) identified-organization(3) oiw(14) secsig(3) algorithms(2) 26}

which is DER-TLV encoded as '2B 0E 03 02 1A'.

F.2.3 Message Integrity ICV

The ICV for the each C-MAC and R-MAC calculation is obtained by enciphering an ICV sequence counter. The ICV sequence counter is initialized during Secure Channel initiation to either:

- the value supplied in tag '91' of the control reference template, for the key transport option (separate initial value for each key), or
- the concatenation of the last 4 bytes of the card secret and the last 4 bytes of Off-Card Entity secret, for the key agreement option (same initial value for all keys).

The ICV sequence counter is incremented by one for each C-MAC and R-MAC. For calculating a C-MAC ICV, the ICV sequence counter is single-DES enciphered using the first part of the Secure Channel C-MAC key. For calculating an R-MAC ICV, the ICV sequence counter is single-DES enciphered using the first part of the Secure Channel R-MAC key.

F.2.4 Message Integrity C-MAC and R-MAC

Message integrity is achieved by applying a MAC to message data. The MAC may be:

- C-MAC for APDU command messages (generated by the Off-Card Entity);
- R-MAC for APDU response messages (generated by the card).

The receiving entity, on receipt of the message containing a MAC, using the same session key, performs the same operation and by comparing its generated MAC with the MAC received from the sending entity is assured of the integrity of the full command or response.

The integrity of the sequence of APDU command or response messages being transmitted to the receiving entity is achieved by using an encrypted sequence counter as part of the MAC generation. This ensures the receiving entity that all messages in a sequence have been received.

F.2.5 APDU Encryption and Decryption for Message Confidentiality

Message confidentiality is achieved by encrypting the whole of the command or response data field. This includes any data within the data field that has already been protected for another purpose, such as secret or private keys encrypted with the data encryption key.

F.3 Cryptographic Usage

F.3.1 DES Session Keys

F.3.1.1 Overview

All session keys shall be double or triple length DES keys.

The Off-Card Entity supplies information on cryptographic keys to be established for the session in a set of control reference templates. Each control reference template specifies the usage of the key.

In the case of key transport, the templates are supplied in the PERFORM SECURITY OPERATION [decipher] command and contain the actual key values.

In the case of key agreement, the templates are supplied in the EXTERNAL AUTHENTICATE command, and do not contain the key values.

F.3.1.2 Control Reference Templates

A control reference template (CRT) is structured as follows:

Tag	Length	Name	Presence
‘B4’ or ‘B8’	'00' - '7F'	CRT tag = ‘B4’ (CCT) or ‘B8’ (CT)	Mandatory
'95'	1	Usage Qualifier = ‘10’ (secure messaging for commands), ‘20’ (secure messaging for responses), ‘30’ (secure messaging for commands and responses), ‘40’ (encipherment of sensitive data in responses), ‘80’ (encipherment of sensitive data in commands), ‘C0’ (encipherment of sensitive data in commands & responses)	Mandatory
'80'	0 or 1	Optional cryptographic mechanism. Contains Key Type, coded according to Table 11-16	Optional
'D1'	0, 16 or 24	Off-Card Entity Session Key	Conditional
'91'	0 or 8	Initial value of sequence counter, for use in secure messaging	Conditional

Table F-11: Single CRT

Note: Usage Qualifier and the CRT tag together define the key usage, which is C-MAC and/or R-MAC for control reference template 'B4', and C-ENC and/or R-ENC or DEK for control reference template 'B8'.

The Key Type identifies the cryptographic algorithm.

The Off-Card Entity Session Key and sequence counter data objects are only present with the key transport option.

The sequence counter data object is used in secure messaging as the initial value for the ICV sequence counter for this key, which is encrypted as defined in section F.2.3 - *Message Integrity ICV* to derive the ICV for the first MAC generated using this key.

F.3.1.3 Session Key Derivation

With the key agreement option, session keys are derived from the secret data exchanged during Secure Channel initiation as follows:

- The two 32-byte secrets Off-Card Entity Secret and Card Secret are exclusive or-ed, giving result (1);
- A 32 byte binary counter is set to a value depending on the key usage and its position in the set of CRTs supplied, as shown below;
- Result (1) is appended with a 32 bit binary counter with the appropriate value for this key, and the result is hashed using SHA-1, giving result (2);
- Bytes 1-16 of result (2) form the double-length DES session key.

Counter value	Key
1	The MAC key whose CRT is first in the set of CRTs supplied by the Off-Card Entity
2	The ENC key whose CRT is first in the set of CRTs supplied by the Off-Card Entity
3	A subsequent MAC key, if any
4	A subsequent ENC key, if any
5	The data encryption key whose CRT is first in the set of CRTs supplied by the Off-Card Entity
6	A subsequent data encryption key, if any

Table F-12: Counter Value for Session Key Calculation

F.3.2 Secure Messaging

F.3.2.1 APDU Command C-MAC Protection

This section applies where command integrity (C-MAC) is required but not command confidentiality (C-ENC).

A C-MAC is generated by an Off-Card Entity and applied across the full APDU command being transmitted to the card including the header, the command data field (if present) and Le (if present). Input data to the MAC calculation is first prepared as defined in ISO/IEC 7816-4:

- The following data is concatenated:
  - The command header CLA, INS, P1, P2 from the unprotected APDU, with the logical channel bits in the CLA byte set to zero, and appended with four bytes '80 00 00 00';
  - If a command data field is present in the unprotected APDU, a BER-TLV data object with tag '81' containing the complete original command data field , regardless of its contents and format;
  - If Le is present in the unprotected APDU, a BER-TLV data object with tag '97' containing the original Le value;
- DES padding is applied as defined in appendix B.4 - *DES Padding*.

A C-MAC is generated using the Secure Channel C-MAC session key, the encrypted sequence counter as the ICV as defined in section F.2.3, and the signature method described in appendix B.1.2.2 - *Single DES Plus Final Triple DES MAC* across the input data.

To reflect the presence of a C-MAC in the command message, the unprotected APDU shall be modified as follows:

- The class byte shall be modified to indicate that this APDU command includes secure messaging. This is achieved by setting to '11' bits 4-3 of a class byte indicating a logical channel number 0 to 4 (unprotected CLA set to '00' - '03' or '80' - '83') or by setting to '1' bit b6 of a class byte indicating a logical channel number 4 to 19 (unprotected CLA set to '40' - '4F' or 'C0' - 'CF'): see section 11.1.4. The logical channel bits are unchanged;
- The length of the command message (Lc) shall be incremented by:
  - 10 bytes to allow for the C-MAC data object, plus
  - 2 or more bytes to allow for the tag and length of the command data field data object (if command data is present - note: the length field may be longer than one byte), plus
  - 3 bytes to allow for the Le data object (if Le is present).
- The command data, if present in the unprotected APDU, shall be encapsulated in a BER-TLV data object with tag '81' and a length field coded according to ISO 8825-1;
- The Le byte, if present in the unprotected APDU, shall be contained in a BER-TLV data object with tag '97';
- The C-MAC shall be encapsulated in a BER-TLV data object with tag '8E' and appended at the end of the command data field.

No padding is present in the transmitted APDU.

The following diagram shows the message reformatting that is performed by the Off-Card Entity when a command is protected for integrity.

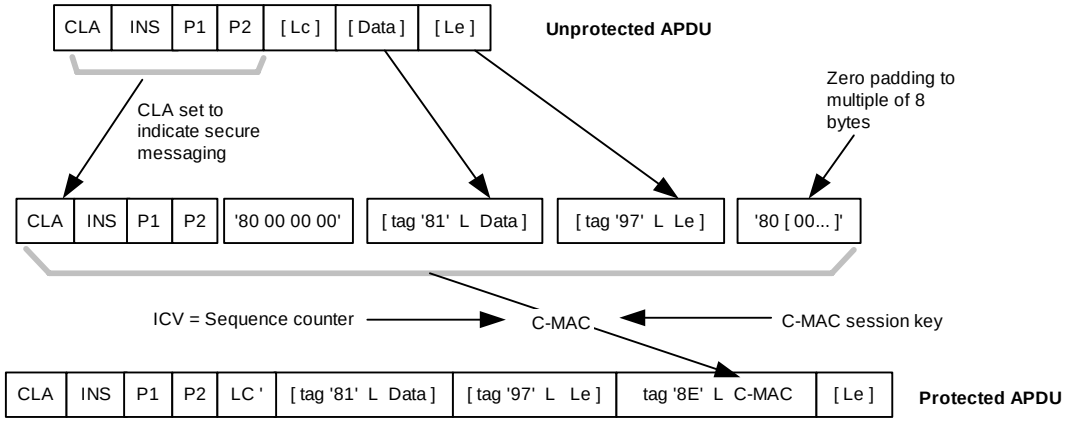

Figure F-10: APDU C-MAC Generation

The card, in order to verify the C-MAC, shall perform the same procedure as employed by the Off-Card Entity in order to verify the C-MAC. The ICV sequence counter used in ICV calculation is then incremented. This is true regardless of whether the APDU processing completes successfully or not i.e. a new sequence counter value shall always be used for the next C-MAC or R-MAC.

F.3.2.2 APDU Command C-ENC Protection

This section applies where command confidentiality (C-ENC) is required but not command integrity (C-MAC).

No encryption shall be applied to a command where there is no command data field: in this case the command message (header and optional Le) is sent without modification.

Otherwise the Off-Card Entity encrypts the command data field of the command message being transmitted to the card. This includes any data within the data field that has already been protected for another purpose e.g. secret or private keys encrypted with the data encryption session key.

Prior to encrypting the data, DES padding is applied as defined in appendix B.4 - *DES Padding*.

The padded command data field is enciphered using triple DES in CBC mode as defined in appendix B.1.1.1, the C-ENC session key established during the Secure Channel initiation process and an ICV of zero.

To reflect the C-ENC protection of the command, the unprotected APDU shall be modified as follows:

- The class byte shall be modified to indicate that this APDU command includes secure messaging. This is achieved by setting to '10' bits 4-3 of a class byte indicating a logical channel number 0 to 4 (unprotected CLA set to '00' - '03' or '80' - '83') or by setting to '1' bit b6 of a class byte indicating a logical channel number 4 to 19 (unprotected CLA set to '40' - '4F' or 'C0' - 'CF'): see section 11.1.4. The logical channel bits are unchanged;
- Lc shall be incremented by:
  - 4 or more bytes to allow for the tag and length of the command data field data object, the padding indicator and the variable padding (the length field may be longer than one byte), plus
  - 3 bytes to allow for the Le data object (if Le is present).
- The encrypted command data shall be preceded by the ISO/IEC 7816 padding indicator '01' and encapsulated in a BER-TLV data object with tag '86' and a length field coded according to ISO 8825-1;
- The Le byte, if present in the unprotected APDU, shall be contained in a BER-TLV data object with tag '96'.

The following diagram shows the message reformatting that is performed by the Off-Card Entity when a command is protected for confidentiality.

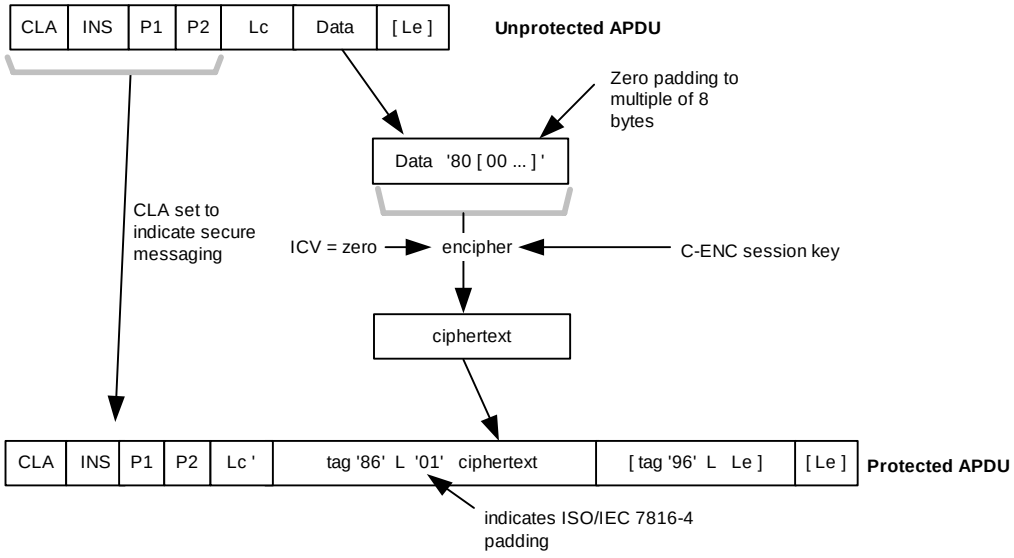

Figure F-11: Secure Messaging: Command message protected for confidentiality

F.3.2.3 APDU Command C-MAC and C-ENC Protection

This section applies where both command confidentiality (C-ENC) and integrity (C-MAC) are required.

No encryption shall be applied to a command where there is no command data field: in this case the message shall be protected as defined in section F.3.2.1 - *APDU Command C-MAC Protection*.

Otherwise the Off-Card Entity first encrypts the command data field of the command message being transmitted to the card as defined in section F.3.2.2.

A C-MAC is generated by an Off-Card Entity as defined in section F.3.2.1. Input data to the MAC calculation is first prepared as defined in ISO/IEC 7816-4:

- The following data is concatenated:
  - The command header CLA, INS, P1, P2 from the unprotected APDU, with the logical channel bits in the CLA byte set to zero, appended with four bytes '80 00 00 00';
  - If a command data field is present in the unprotected APDU, a BER-TLV data object with tag '87' containing the ISO/IEC 7816 padding indicator '01' followed by the encrypted command data;
  - If Le is present in the unprotected APDU, a BER-TLV data object with tag '97' containing the original Le value;
- DES padding is applied as defined in appendix B.4 - *DES Padding*.

To reflect the presence of a C-MAC and C-ENC protection of the command, the unprotected APDU shall be modified as follows:

- The class byte shall be modified to indicate that this APDU command includes secure messaging. This is achieved by setting to '11' bits 4-3 of a class byte indicating a logical channel number 0 to 4 (unprotected CLA set to '00' - '03' or '80' - '83') or by setting to '1' bit b6 of a class byte indicating a logical channel number 4 to 19 (unprotected CLA set to '40' - '4F' or 'C0' - 'CF'): see section 11.1.4. The logical channel bits are unchanged;.
- Lc shall be incremented by:
  - 10 bytes to allow for the C-MAC data object, plus
  - 4 or more bytes to allow for the tag and length of the command data field data object, the padding indicator and the variable padding (the length field may be longer than one byte), plus
  - 3 bytes to allow for the Le data object (if Le is present).
- The encrypted command data shall be preceded by the ISO/IEC 7816 padding indicator '01' and encapsulated in a BER-TLV data object with tag '87' and a length field coded according to ISO 8825-1;
- The Le byte, if present in the unprotected APDU, shall be contained in a BER-TLV data object with tag '97';
- The C-MAC shall be encapsulated in a BER-TLV data object with tag '8E' and appended at the end of the command data field.

The following diagram shows the message reformatting that is performed by the Off-Card Entity when a command is protected for integrity and confidentiality.

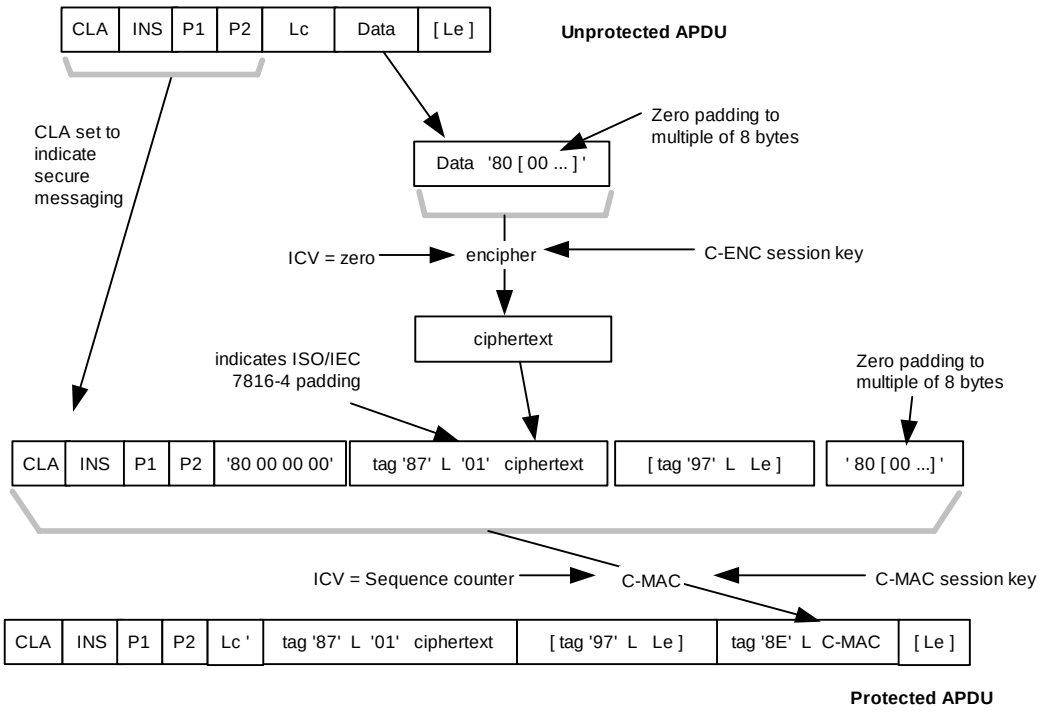

Figure F-12: Secure Messaging: Command message protected for integrity and confidentiality

F.3.2.4 APDU Response R-MAC Protection

This section applies where response integrity (R-MAC) is required but not confidentiality (R-ENC).

No R-MAC shall be generated and no protection shall be applied to a response where status bytes SW1 and SW2 indicate an error: in this case only status bytes shall be returned in the response.

When R-MAC protection is required for a case 1 or case 3 command, the card shall process the command as a case 2 or case 4 command respectively and treat Le as if it were present and set to zero.

An R-MAC is generated by the card across the response data field (if present) and status bytes. Input data to the MAC calculation is first prepared as defined in ISO/IEC 7816-4:

- The following data is concatenated:
  - If a response data field is present in the unprotected APDU, a BER-TLV data object with tag '81' containing the complete original response data field, regardless of its contents and format;
  - A BER-TLV data object with tag '99', containing the original status word SW1 - SW2 value.
- DES padding is applied as defined in appendix B.4 - *DES Padding*.

An R-MAC is generated using the Secure Channel R-MAC session key, the encrypted sequence counter as the ICV as defined in section F.2.3, and the signature method described in appendix B.1.2.2 - *Single DES Plus Final Triple DES MAC* across the input data.

To reflect the presence of an R-MAC protection of the response, the unprotected APDU shall be modified as follows:

- The response data, if present in the unprotected APDU, shall be encapsulated in a BER-TLV data object with tag '81' and a length field coded according to ISO 8825-1;
- The status word SW1 - SW2 of the unprotected APDU shall be contained in a BER-TLV data object with tag '99';
- The R-MAC shall be encapsulated in a BER-TLV data object with tag '8E' and appended at the end of the response data field.

No padding is present in the transmitted APDU.

The Off-Card Entity, in order to verify the R-MAC, shall perform the same processing in order to generate an R-MAC and compare it with the transmitted R-MAC.

The following diagram shows the message reformatting that is performed by the card when a response message is protected for integrity.

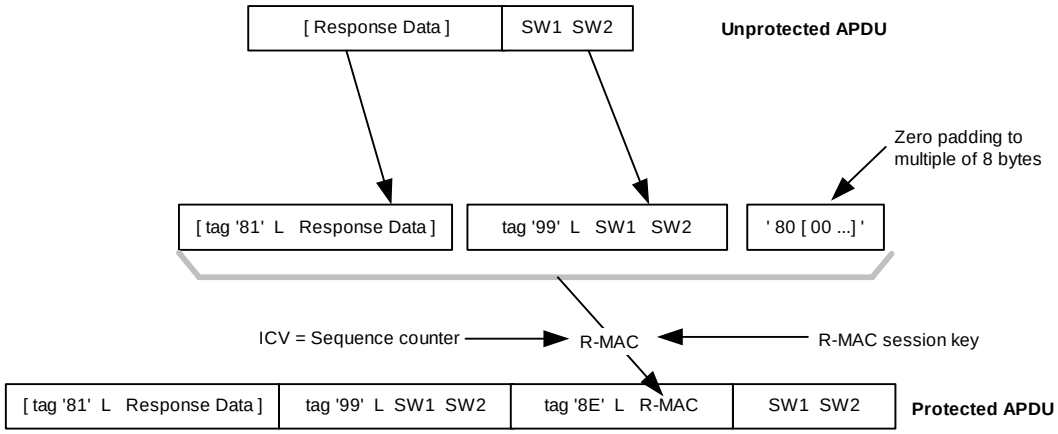

Figure F-13: Secure Messaging: Response message protected for integrity

F.3.2.5 APDU Response R-ENC Protection

This section applies where response confidentiality (R-ENC) is required but not integrity (R-MAC).

No protection shall be applied to a response where status bytes SW1 and SW2 indicate an error or where there is no response data field: in this case only status bytes shall be returned in the response.

Otherwise, the Security Domain encrypts the response data field. This includes any data within the data field that has already been protected for another purpose, such as secret or private keys encrypted with the data encryption session key.

Prior to encrypting the response data field, DES padding is applied as defined in appendix B.4 - *DES Padding*.

The padded response data field is then enciphered using triple DES in CBC mode as defined in appendix B.1.1.1 - *CBC Mode*, the R-ENC session key established during the Secure Channel initiation process and an ICV of zero.

To reflect the R-ENC protection of the response, the unprotected APDU shall be modified as follows:

- The encrypted response data shall be preceded by the ISO/IEC 7816 padding indicator '01' and encapsulated in a BER-TLV data object with tag '86' and a length field coded according to ISO 8825-1.

The following diagram shows the message reformatting that is performed by the card when a response message is protected for confidentiality.

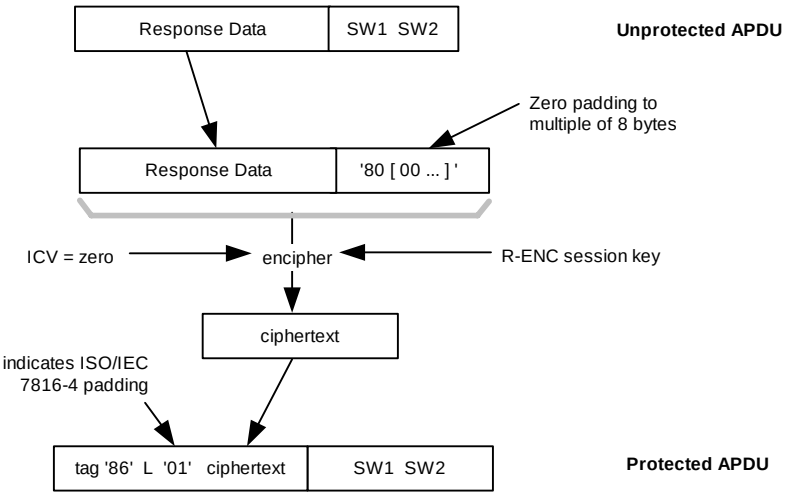

Figure F-14: Secure Messaging: Response message protected for confidentiality

F.3.2.6 APDU Response R-MAC and R-ENC Protection

This section applies where both response confidentiality (R-ENC) and response integrity (R-MAC) are required.

No R-MAC or encryption shall be applied to a response where status bytes SW1 and SW2 indicate an error: in this case only status bytes shall be returned in the response.

No encryption shall be applied to a response where there is no response data field: in this case the message shall be protected as defined in section F.3.2.4 - *APDU Response R-MAC Protection*.

Otherwise, the card first encrypts the response data field of the response message being transmitted to the Off-Card Entity as defined in section F.3.2.5.

An R-MAC is then generated by the card as defined in section F.3.2.4. Input data to the MAC calculation is first prepared as defined in ISO/IEC 7816-4:

- The following data is concatenated:
  - If a response data field is present in the unprotected APDU, a BER-TLV data object with tag '87' containing the ISO/IEC 7816 padding indicator '01' followed by the encrypted response data;
  - A BER-TLV data object with tag '99' containing the original SW1 - SW2 status word value.
- DES padding is applied as defined in appendix B.4 - *DES Padding*.

To reflect the presence of an R-MAC and R-ENC protection of the response, the unprotected APDU shall be modified as follows:

- The encrypted response data shall be preceded by the ISO/IEC 7816 padding indicator '01' and encapsulated in a BER-TLV data object with tag '87' and a length field coded according to ISO 8825-1;
- The status word SW1 - SW2 of the unprotected APDU shall be contained in a BER-TLV data object with tag '99';

- The R-MAC shall be encapsulated in a BER-TLV data object with tag '8E' and appended at the end of the response data field.

No R-MAC padding is present in the transmitted APDU.

The Off-Card Entity, in order to verify the R-MAC, shall perform the same processing in order to generate an R-MAC and compare it with the transmitted R-MAC.

The following diagram shows the message reformatting that is performed by the card when a response message is protected for integrity and confidentiality.

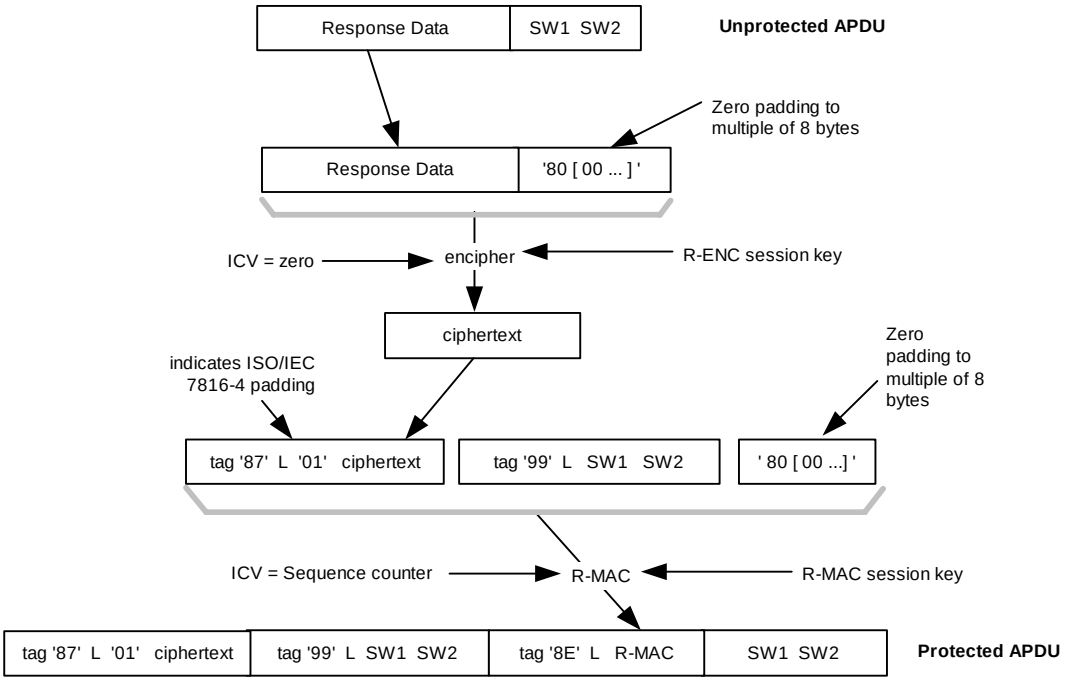

Figure F-15: Secure Messaging: Response message protected for integrity and confidentiality

F.3.2.7 Sensitive Data Encryption and Decryption

Data encryption is used when transmitting sensitive data to and from the card. For instance all keys transmitted to a card (e.g. in a PUT KEY command) should be encrypted. Data encryption is over and beyond the Current Security Level required for the Secure Channel Session. The encryption process uses the relevant data encryption session key (DEK) for sensitive data in command messages or for sensitive data in response messages. The encryption method uses DES in ECB or CBC mode depending on the Key Type of the DEK Key in the CRT: see appendix B.1.1.1 - *CBC Mode* or appendix B.1.1.2 - *ECB Mode*. If the key type is omitted for the DEK Key it shall be known implicitly. The sensitive data block length shall be constructed as a multiple of 8-byte long block before the encryption operations: the eventual padding method is application specific.

The encryption is performed across the sensitive data and the result of each encryption becomes part of the encrypted data. This encrypted data becomes part of the clear text data field in the command/response message. The decryption is the exact opposite of the above operation: in particular, no padding is removed by the decryption operation.

F.4 Commands

Because certificates, digital signatures and some data fields can be long, command and response chaining as defined in ISO/IEC 7816-4 is used to transfer successive data blocks.

With command chaining, the command data is sent in multiple APDUs, the command data being segmented arbitrarily. All except the final command in the chain shall indicate command chaining by setting to '1' bit 5 of the class byte according to ISO/IEC 7816-4.

With response chaining, the response data is sent in multiple APDUs, the response data being segmented arbitrarily.

Command	Secure Channel Initiation	
	signature without message recovery	signature with message recovery
EXTERNAL AUTHENTICATE	✓	✓
GET CHALLENGE	✓	✓
GET DATA [certificate]	✓	✓
INTERNAL AUTHENTICATE	✓	✓
MANAGE SECURITY ENVIRONMENT	✓	✓
PERFORM SECURITY OPERATION [decipher]	✓	
PERFORM SECURITY OPERATION [verify certificate]	✓	✓

Table F-13: SCP10 Command Support

Table F-14 summarizes the minimum security requirements for the APDU commands.

Command	Minimum Security
EXTERNAL AUTHENTICATE	Validated PK.OCE.AUT and card challenge
GET CHALLENGE	None
GET DATA [certificate]	None
INTERNAL AUTHENTICATE	Current Security Level is at least AUTHENTICATED or ANY_AUTHENTICATED
MANAGE SECURITY ENVIRONMENT	None
PERFORM SECURITY OPERATION [decipher]	None
PERFORM SECURITY OPERATION [verify certificate]	None

Table F-14: Minimum Security Requirements for SCP10 commands

The following table provides the list of SCP10 command support per card Life Cycle State.

Command	OP_READY			INITIALIZED			SECURED			CARD_LOCKED		TERMINATED	
	AM SD	DM SD	SD	AM SD	DM SD	SD	AM SD	DM SD	SD	FA SD	SD	FA SD	SD
EXTERNAL AUTHENTICATE	✓	✓		✓	✓		✓	✓		✓			
GET CHALLENGE	✓	✓		✓	✓		✓	✓		✓			
GET DATA [certificate]	✓	✓		✓	✓		✓	✓		✓			
INTERNAL AUTHENTICATE	✓	✓		✓	✓		✓	✓		✓			
MANAGE SECURITY ENVIRONMENT	✓	✓		✓	✓		✓	✓		✓			
PERFORM SECURITY OPERATION [decipher]	✓	✓		✓	✓		✓	✓		✓			
PERFORM SECURITY OPERATION [verify certificate]	✓	✓		✓	✓		✓	✓		✓			

Table F-15: SCP10 command support per Card Life Cycle State

Legend of Table F-13 and Table F-15:

AM SD: Security Domain with Authorized Management privilege.

DM SD: Supplementary Security Domain with Delegated Management privilege.

FA SD: Security Domain with Final Application privilege. (Note: command support for an Application with Final Application Privilege which is not a Security Domain is subject to Issuer policy, within the constraints of what is permitted according to the card Life Cycle State.)

SD: Other Security Domain.

✓ : Support required.

Blank cell: Support optional.

Striped cell: Support prohibited.

F.4.1 EXTERNAL AUTHENTICATE Command

F.4.1.1 Definition and Scope

This command is used to authenticate the Off-Card Entity by the Security Domain. This command is also used with the key agreement option to support session key establishment. It shall be immediately preceded (on the same logical channel of the same card I/O interface) by a GET CHALLENGE command. It may be followed (on the same logical channel of the same card I/O interface) by an INTERNAL AUTHENTICATE command.

F.4.1.2 Command Message

The EXTERNAL AUTHENTICATE command message is coded as follows:

Code	Value	Meaning
CLA	'00' - '03', '40' - '4F', '10' - '13' or '50' - '5F'	See section 11.1.4
INS	'82'	EXTERNAL AUTHENTICATE
P1	'00'	Reference control parameter P1: no information given
P2	'00'	Reference control parameter P2: no information given
Lc	'xx'	Length of Entity Signature (key transport) or encrypted Off-Card Entity Signature (key agreement)
Data	'xx xx…'	Command data field
Le	-	Not present

Table F-16: EXTERNAL AUTHENTICATE Command Message

A Security Domain may support other values of Reference Control Parameters P1 and P2 as defined in ISO/IEC 7816-4.

F.4.1.3 Data Field Sent in the Command Message

The data field of the EXTERNAL AUTHENTICATE command message contains the Off-Card Entity Signature (key transport) or encrypted Off-Card Entity Signature (key agreement) - see section F.1.3.4 - *Off-Card Entity Authentication* for further details.

F.4.1.4 Data Field Returned in the Response Message

No data is returned by this command.

F.4.1.5 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following specific error and warning conditions.

SW1	SW2	Meaning
'63'	'00'	Verification of certificate failed
'94'	'84'	Algorithm not supported

Table F-17: Error Conditions

F.4.2 GET CHALLENGE Command

F.4.2.1 Definition and Scope

This command is used to obtain a random challenge from the Security Domain, to support authentication of the Off-Card Entity to the Security Domain. It precedes the EXTERNAL AUTHENTICATE command. It shall have been preceded (on the same logical channel of the same card I/O interface), immediately or not, by a MANAGE SECURITY ENVIRONMENT or PERFORM SECURITY OPERATION [verify certificate] command.

F.4.2.2 Command Message

The GET CHALLENGE command message is coded according to the following table:

Code	Value	Meaning
CLA	'00' - '03' or '40' - '4F'	See section 11.1.4
INS	'84'	GET CHALLENGE
P1	'00'	Reference Control Parameter P1: no information given
P2	'00'	Reference Control Parameter P2: no information given
Lc	Absent	
Data	Absent	
Le	'00'	

Table F-18: GET CHALLENGE Command Message

A Security Domain may support other values of Reference Control Parameters P1 and P2 as defined in ISO/IEC 7816-4.

F.4.2.3 Data Field Sent in the Command Message

There is no command data.

F.4.2.4 Data Field Returned in the Response Message

Data returned comprises a card challenge, coded on 8 bytes with the key agreement option and 16 bytes with the key transport option.

F.4.2.5 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may return a general error condition as listed in section 11.1.3 - *General Error Conditions*.

F.4.3 GET DATA [certificate] Command

F.4.3.1 Definition and Scope

The GET DATA [certificate] command is used to retrieve either information about all certificates that can be retrieved from the card, or a single certificate. This command may be issued at any time, in particular it may be interleaved with PERFORM SECURITY OPERATION [verify certificate] commands. If it is issued when a Secure Channel Session is active, it must comply with the Current Security Level of that Secure Channel Session.

F.4.3.2 Command Message

The following command is used to obtain certificate information or a single certificate from the Security Domain.

Code	Value	Meaning
CLA	'00' - '0F', '40' - '4F', '60' - '6F', '80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'	See section 11.1.4
INS	‘CA’ or ‘CB’	If CLA = '00' - '0F', '40' - '4F' or '60' - '6F', even or odd instruction code 'CA' or 'CB' If CLA = '80' - '8F', 'C0' - 'CF' or 'E0' - 'EF', even instruction code 'CA'
P1 P2		Reference Control Parameters P1 and P2: there are three options:
	'7F 21'	Tag of certificate
	'50 31'	Data object identifier of EF.OD
	'xx xx'	(Any other value) data object identifier of a certificate
Lc	'xx' or omitted	Not present if P1 P2 = '7F 21', otherwise length of command data
Data	'xx xx...' or omitted	Not present, or command data
Le	'00'	

Table F-19: GET DATA [certificate] Command Message

F.4.3.3 Reference Control Parameters P1 and P2

If P1 P2 = ‘7F 21’, the command is a request to retrieve the certificate of the Security Domain’s default public key, CERT.SD.AUT.

Otherwise, for any value other than those defined in section 11.3 - *GET DATA Command*, the command is a request to access EF.OD or another certificate and the instruction code shall be set to 'CA' if the class byte indicates a GlobalPlatform command (CLA set to '80' - '8F', 'C0' - 'CF' or 'E0' - 'EF') or 'CB' if the class byte indicates an ISO command (CLA set to '00' - '0F', '40' - '4F' or '60' - '6F').

- If P1 P2 = ‘50 31’, the command is a request to retrieve details of all available certificates held by the Security Domain for retrieval and verification by an Off-Card Entity;
- Otherwise, for any other value of P1 P2, the command shall be treated as a request to retrieve a certificate whose pointer is given in P1 P2. This would typically follow a command with P1 P2 = '50 31', where the

Cryptographic Information Objects returned in the response have pointers to individual certificates. However, the Off-Card Entity may already know the location of a required certificate, and issue this command directly.

F.4.3.4 Data Field Sent in the Command Message

When P1-P2 is different from '7F21', the command data shall be present and coded as follows:

Name	Length	Name	Presence
Tag list tag	1	'5C'	Mandatory
Tag list length	1	'00' (empty, indicating 'retrieve all data')	Mandatory

Table F-20: GET DATA [certificate] Command Data Message

F.4.3.5 Data Field Returned in the Response Message

When the command is issued to retrieve a certificate (P1-P2 different from '5031') and the class byte indicates a GlobalPlatform command (CLA set to '80' - '8F', 'C0' - 'CF' or 'E0' - 'EF'), the certificate shall be returned TLV-coded as follows:

Name	Length	Name	Presence
Certificate tag	2	'7F21'	Conditional
Certificate length	1, 2 or 3	'00' - '7F', or '81 80' - '81 FF' or '82 01 00' - '82 FF FF'	Conditional
Certificate data	n	'xxxx...' (as defined in section F.1.2.4).	Mandatory

Table F-21: GET DATA [certificate] Response Data Field - Certificate

When the command is issued to retrieve a certificate (P1-P2 different from '5031') and the class byte indicates an ISO command (CLA set to '00' - '0F', '40' - '4F' or '60' - '6F'), only the certificate value shall be returned.

If the command was issued to retrieve certificate information details (P1-P2 = '5031'), the contents of EF.OD listing all the Cryptographic Information Objects for the different certificates held in by the Security Domain shall be returned as defined in section F.1.2.3.

F.4.3.6 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following specific error and warning conditions:

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in command data
'6A'	'88'	Referenced data not found

Table F-22: Error Conditions

F.4.4 INTERNAL AUTHENTICATE Command

F.4.4.1 Definition and Scope

This command is used to authenticate the Security Domain, by the Off-Card Entity. This command is also used with the key agreement option, to support session key establishment. It shall be immediately preceded (on the same logical channel of the same card I/O interface) by an EXTERNAL AUTHENTICATE command.

F.4.4.2 Command Message

The INTERNAL AUTHENTICATE command message is coded according to the following table:

Code	Value	Meaning
CLA	'00' - '03' or '40' - '4F'	See section 11.1.4
INS	'88'	INTERNAL AUTHENTICATE
P1	'00'	Reference control parameter P1: no information given
P2	'00'	Reference control parameter P2: no information given
Lc	'xx'	Length of Off-Card Entity challenge
Data	'xx xx…'	Command data field
Le	'00'	

Table F-23: INTERNAL AUTHENTICATE Command Message

A Security Domain may support other values of Reference Control Parameters P1 and P2 as defined in ISO/IEC 7816-4.

F.4.4.3 Data Field Sent in the Command Message

The data field of the INTERNAL AUTHENTICATE command message contains the following data:

Name	Length	Name	Presence
Off-Card Entity challenge	8 or 16	'xxxx...'	Mandatory
Off-Card Entity id	8	'xxxx...' (part of CERT.OCE.AUT)	Mandatory

Table F-24: INTERNAL AUTHENTICATE Command Data Field

Off-Card Entity challenge is 16 bytes for the key transport option, 8 bytes for the key agreement option.

F.4.4.4 Data Field Returned in the Response Message

The Card Signature (key transport) or encrypted Card Signature (key agreement) is returned - see section F.1.3.3 for details.

F.4.4.5 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following specific error and warning conditions:

SW1	SW2	Meaning
'61'	'xx'	Response data incomplete, 'xx' more bytes available

Table F-25: Warning Conditions

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in command data

Table F-26: Error Conditions

F.4.5 MANAGE SECURITY ENVIRONMENT Command

F.4.5.1 Definition and Scope

This command selects the Secure Channel Protocol '10' and its options as well as defining specific keys to be used by the Security Domain. If a Secure Channel Session is active on this logical channel and card I/O interface, it shall be terminated on receipt of this command, regardless of the validity of the command.

F.4.5.2 Command Message

The MANAGE SECURITY ENVIRONMENT command message is coded according to the following table:

Code	Value	Meaning
CLA	'00' - '03' or '40' - '4F'	See section 11.1.4
INS	'22'	MANAGE SECURITY ENVIRONMENT
P1	'81' or 'C1'	Reference Control Parameter P1 '81': External (Off-Card Entity) Authentication only 'C1': External and Internal (Mutual) Authentication
P2	'A4' or 'B6'	Reference Control Parameter P2: 'A4': Authentication: no certificate verification will be performed by the card 'B6': Digital signature: certificate verification will be performed by the card
Lc	'xx'	Length of command data
Data	'xx xx…'	Off-Card Entity data
Le	-	Not present

Table F-27: MANAGE SECURITY ENVIRONMENT Command Message

F.4.5.3 Reference Control Parameter P1

The value of P1 is based on ISO/IEC 7816-4, as follows:

b8	b7	b6	b5	b4	b3	b2	b1	Description
1	-	-	-	-	-	-	-	Verification, encipherment, external authentication and key agreement
-	1	-	-	-	-	-	-	Computation, decipherment, internal authentication and key agreement
-	-	-	-	-	-	-	1	SET
-	-	X	X	X	X	X	-	Values defined in ISO/IEC 7816-4

Table F-28: MANAGE SECURITY ENVIRONMENT Reference Control Parameter P1

F.4.5.4 Reference Control Parameter P2

This is set according to the template that is appropriate for the subsequent message flow, as defined in ISO/IEC 7816-4: 'A4' if certificate verification by the card is omitted, 'B6' if certificate verification is to be performed by the card.

F.4.5.5 Data Field Sent in the Command Message

The command data field is formatted as follows:

Tag	Length	Name	Presence
'80'	2	Cryptographic mechanism reference: SCP id + options 'i' (= scp || i)	Mandatory
'83'	0 or 1-n	Public key reference	Conditional
'84'	0 or 1-n	Private key reference	Conditional

Table F-29: MANAGE SECURITY ENVIRONMENT Command Data Field

The 'cryptographic mechanism reference' (tag '80', value '10') designates GlobalPlatform asymmetric Secure Channel Protocol '10' (SCP10) and its options, and distinguishes it from any other protocol that might be supported by the selected Issuer Security Domain, Security Domain or Application. Option 'i' is described in section F.1.1.

The 'public key reference' (tag '83') designates the public key to be used by the Security Domain in subsequent cryptographic operations during Secure Channel Session initiation. The referenced public key shall have already been validated by the Security Domain or shall be the public key of the Security Domain's Trust Point for External Authentication. If no reference value is provided in the command, the Security Domain shall use by default the public key of its Trust Point for External Authentication, designated PK.TP_EX.AUT, as the first key to use for certificate verification within a session: see section F.1.2.1.

The 'private key reference' (tag '84') designates the private key to be used by the Security Domain in subsequent cryptographic operations during Secure Channel Session initiation. The referenced private key shall be present and known to the Security Domain. If no reference value is provided in the command, the Security Domain shall use by default its private key designated SK.SD.AUT

A Security Domain may support other data elements as defined in ISO/IEC 7816-4.

F.4.5.6 Data Field Returned in the Response Message

There is no data returned from this command.

F.4.5.7 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following specific error and warning conditions.

SW1	SW2	Meaning
6A	88	Referenced data not found
94	84	Algorithm not supported

Table F-30: Error Conditions

F.4.6 PERFORM SECURITY OPERATION [decipher] Command

F.4.6.1 Definition and Scope

This command is used with the session key transport option, and transmits the DES session keys from the Off-Card Entity to the Security Domain. It shall have been preceded (on the same logical channel of the same card I/O interface), immediately or not, by a MANAGE SECURITY ENVIRONMENT or PERFORM SECURITY OPERATION [verify certificate] command.

F.4.6.2 Command Message

The PERFORM SECURITY OPERATION [decipher] command message is coded according to the following table:

Code	Value	Meaning
CLA	'00' - '03', '40' - '4F', '10' - '13' or '50' - '5F'	See section 11.1.4
INS	'2A'	PERFORM SECURITY OPERATION [decipher]
P1	'80'	Reference Control Parameter P1: clear text object
P2	‘84’	Reference Control Parameter P2: cryptogram (plain value encoded in BER-TLV) present in the command
Lc	'xx'	Length of encrypted data
Data	'xx xx…'	Encrypted data
Le	-	Not present

Table F-31: PERFORM SECURITY OPERATION [decipher] Command Message

A Security Domain may support other values of Reference Control Parameters P1 and P2 as defined in ISO/IEC 7816-4.

F.4.6.3 Data Field Sent in the Command Message

The data field of the PERFORM SECURITY OPERATION [decipher] command message contains the Encrypted Off-Card Entity Session Data.

The Off-Card Entity session keys are in control reference templates as defined in section F.3.1.2, one template per session key, concatenated for encryption. Off-Card Entity Session Key Data is formed by padding the session key CRTs to the length of the Security Domain public key modulus as shown in the table below, and encrypting the result with the Security Domain public key (PK.SD.AUT).

Meaning	Length	Meaning	Presence
Padding	2	'0002'	
Padding	8-n	'FF'...'FF'	
Padding	1	'00'	
Tag of Security Level ('D3')	1	'D3'	Mandatory
Length of Security Level	1	‘01’	Mandatory

Meaning	Length	Meaning	Presence
Security Level	1	‘xx’	Mandatory
CRT tag	1	‘B4’ (CCT) or ‘B8’ (CT)	Mandatory
Length of CRT	1	'00' - '7F'	Mandatory
CRT contents (with session key)	n	'xxxx...'	Mandatory
…..	….	…..	…
CRT tag	1	‘B4’ (CCT) or ‘B8’ (CT)	Optional
Length of CRT	1	‘00’ - ‘7F’	Conditional
CRT contents (with session key)	n	'xxxx...'	Conditional

Table F-32: Off-Card Entity Session Key Data - Clear Text before Encryption

F.4.6.4 Data Field Returned in the Response Message

No data is returned from this command.

F.4.6.5 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following specific error and warning conditions.

SW1	SW2	Meaning
'6A'	'80'	Incorrect values in command data

Table F-33: Error Conditions

F.4.7 PERFORM SECURITY OPERATION [verify certificate] Command

F.4.7.1 Definition and Scope

This command is used to provide a certificate to the Security Domain for verification. It may be preceded (on the same logical channel of the same card I/O interface) by a MANAGE SECURITY ENVIRONMENT command or a PERFORM SECURITY OPERATION [verify certificate] command and may be interleaved with GET DATA [certificate] commands.

F.4.7.2 Command Message

The PERFORM SECURITY OPERATION [verify certificate] command message is coded according to the following table:

Code	Value	Meaning
CLA	'00' - '03', '40' - '4F', '10' - '13' or '50' - '5F'	See section 11.1.4
INS	'2A'	PERFORM SECURITY OPERATION [verify certificate]
P1	'00'	Reference Control Parameter P1: no object in the response
P2	‘AE’ or ‘BE’	Reference Control Parameter P2: input template for certificate verification present in the command ‘AE’: non self descriptive card verifiable certificate (only the concatenated value fields are certified) ‘BE’: self descriptive card verifiable certificate (the TLV data elements are certified)
Lc	'xx'	Length of certificate data
Data	'xx xx…'	Certificate data
Le	-	Not present

Table F-34: PERFORM SECURITY OPERATION [verify certificate] Command Message

F.4.7.3 Data Field Sent in the Command Message

The command data field is certificate data as follows.

Name	Length	Name	Presence
Certificate tag	2	‘7F21’	Mandatory
Certificate length	1, 2 or 3	'00' - '7F', or '81 80' - '81 FF' or '82 01 00' - '82 FF FF'	Mandatory
Certificate data	n	'xxxx...' (as described in section F.1.2.4)	Mandatory

Table F-35: PERFORM SECURITY OPERATION [verify certificate] Command Data Field

The Security Domain verifies the certificate presented using the Current Public Key as known to the Security Domain. For the first certificate presented in the session, the Current Public Key is either the default Public Key - the Public Key of the Trust Point, or another Public Key as announced in the MANAGE SECURITY ENVIRONMENT command. A series of certificates can be presented to the Security Domain in subsequent PERFORM SECURITY OPERATION [verify certificate] commands. For each subsequent PERFORM SECURITY

OPERATION [verify certificate] command, the Current Public Key is the one that was certified in the certificate presented in the previous PERFORM SECURITY OPERATION [verify certificate] command.

F.4.7.4 Data Field Returned in the Response Message

No data is returned from this command.

F.4.7.5 Processing State Returned in the Response Message

A successful execution of the command shall be indicated by status bytes '90' '00'.

The command may either return a general error condition as listed in section 11.1.3 - *General Error Conditions* or one of the following specific error and warning conditions.

SW1	SW2	Meaning
'63'	'00'	Verification of the certificate failed
'68'	'83'	The last command of the chain was expected
'6A'	'80'	Incorrect values in command data

Table F-36: Error Conditions

G Trusted Framework Inter-Application Communication

GlobalPlatform supports a general Trusted Framework scheme for inter-application communication. This general scheme is compatible with the CAT Runtime Environment and secure communication as defined in ETSI TS 102 225, TS 102 241 and related specifications. In summary:

- An Application, the Receiving Entity, receives a message from an off-card entity, whose contents is destined to another Application (the Target Application);
- The Receiving Entity interacts with a Trusted Framework on the card to communicate with the Target Application;
- The Trusted Framework handles the security of the inter-application communication by applying its own rules to the interaction, which may include finding the Security Domain associated with the Target Application and having it handle the security of the incoming message;
- The Trusted Framework finds the appropriate interface of the Target Application and passes to it the incoming message contents;
- The Target Application processes the event and eventually provides to the Trusted Framework some response message to be returned;
- The Trusted Framework, being responsible of the inter-application communication security, applies its own security rules to the response message, which may include requesting the Security Domain associated with the Target Application to handle the response message security;
- The Receiving Entity handles the transmission of the response message to the off-card entity.

Passing the incoming message through to the Target Application, and returning the Target Application's response to the off-card entity implies security requirements for not only the Trusted Framework but also the Receiving Entity. Each Application present on the card playing the role of a Receiving Entity shall:

- Enforce the Issuer's security rules for inter-application communication;
- Ensure that incoming messages are properly provided unaltered to the Trusted Framework;
- Ensure that any response messages are properly returned unaltered to the off-card entity.

The 'Trusted Path' privilege qualifies an Application as a Receiving Entity.

To be able to interact with a required Trusted Framework, the Receiving Entity shall obtain a handle to the required Trusted Framework.

To be able to receive incoming messages from a Receiving Entity, the Target Application declares its interface(s) to the corresponding Trusted Framework. The Target Application is registered at installation time for each event it is capable of handling.

When requested to perform inter-application communication, the Trusted Framework shall check that

- The Receiving Entity has the 'Trusted Path' privilege;
- The Target Application is registered as being willing to accept the event or incoming message.

Trusted Frameworks shall comply with the general requirements stated in this section but are otherwise outside the scope of this version of the specification.

H GlobalPlatform Data Values and Card Recognition Data

H.1 Data Values

The Object Identifier (OID) assigned to GlobalPlatform is as follows:

```
GlobalPlatform OID ::=  {iso(1) member-body(2) country-USA(840)
                        GlobalPlatform(114283)}
```

The hexadecimal representation of the above OID is '2A864886FC6B'. The Object Identifier value for Card Recognition Data `{globalPlatform 1}` or `{1 2 840 114283 1}`, which also identifies GlobalPlatform as the Tag Allocation Authority for Card Recognition Data objects has an hexadecimal representation of '2A864886FC6B01' and so on and so forth for all subsequent GlobalPlatform Object Identifiers.

The Registered Application Provider Identifier (RID) assigned to GlobalPlatform is as follows:

GlobalPlatform RID = 'A000000151'

The default AID of the Issuer Security Domain, based on the above ISO assigned RID, is 'A0000001510000'.

The encoding of an OID is defined in ISO/IEC 8825-1. The definition of a RID is described in ISO/IEC 7816-4.

H.2 Structure of Card Recognition Data

All data is TLV encoded. Application tags not defined in this specification are RFU. GlobalPlatform may assign additional Application tags in the future.

Many of the data objects contain Object Identifiers (OIDs). An OID is made up of a series of numeric values. Each OID is shown here in curly brackets, with its component values separated by spaces. The actual numeric values of some of the symbolic OID values shown, such as “globalPlatform”, are defined in this document and may themselves comprise a series of values.

Tag	Explanation	Length	Value	Presence
'66'	Tag for 'Card Data'	variable - see note 1	Data objects identified in 7816-6, including tag '73'	Mandatory
'73'	Tag for 'Card Recognition Data'	variable	Data objects listed below	Mandatory
'06'	Universal tag for "Object Identifier" (OID)	variable	`{globalPlatform 1}` OID for Card Recognition Data, also identifies Global Platform as the Tag Allocation Authority	Mandatory
'60' '06'	Application tag 0 'OID' tag	variable variable	`{globalPlatform 2 v}` OID for Card Management Type and Version - see note 2	Mandatory
'63' '06'	Application tag 3 'OID' tag	variable variable	`{globalPlatform 3}` OID for Card Identification Scheme - see note 3	Mandatory
'64' '06'	Application tag 4 'OID' tag	variable variable	`{globalPlatform 4 scp i}` OID for Secure Channel Protocol of the Issuer Security Domain and its implementation options- see note 4	Mandatory
'65'	Application tag 5	variable	Card configuration details - see note 5	Optional
'66'	Application tag 6	variable	Card / chip details - see note 6	Optional
'67'	Application tag 7	variable	Issuer Security Domain's Trust Point certificate information - see note 7	Optional
'68'	Application tag 8	variable	Issuer Security Domain certificate information - see note 8	Conditional

Table H-1: Structure of Card Recognition Data

Note 1: Card Data should not exceed 127 bytes. This recommendation is to avoid the complexity of extending the length field of the '66' and '73' data objects and also to minimize the overheads involved in transferring the data.

Note 2: Tag '60': The OID **`{globalPlatform 2 v}`** identifies a card that conforms to the GlobalPlatform Card Specification version "**v**". Thus a card conforming to the GlobalPlatform Card Specification 2.2 would use OID **`{globalPlatform 2 2 2}`** and a card conforming to the GlobalPlatform Card Specification 2.1.1 would use OID **`{globalPlatform 2 2 1 1}`**.

Note 3: Tag '63': The OID **`{globalPlatform 3}`** indicates a GlobalPlatform card that is uniquely identified by the Issuer Identification Number (IIN) and Card Image Number (CIN), as defined in sections 7.4.1.1 - *Issuer Identification Number* and 7.4.1.2 - *Card Image Number*. The objective is that an off-card entity is able to construct a globally unique identifier for the card by concatenating this **`{globalPlatform 3}`** OID, the IIN and the CIN.

Note 4: Tag '64'. The OID **`{globalPlatform 4 scp i}`** identifies the Secure Channel Protocol of the Issuer Security Domain. "**scp**" identifies the Secure Channel Protocol identifier as defined in section 10.7 - *Secure Channel Protocol Identifier*. "**i**" identifies the eventual implementation options as defined in appendix D.1.1 - *SCP01 Secure Channel* for SCP01, appendix E.1.1 - *SCP02 Secure Channel* for SCP02 or appendix F.1.1 - *SCP10 Secure Channel* for SCP10.

Note 5: The data object with tag '65' may contain information about the GlobalPlatform implementation details or commonly used Card Issuer options. Such information shall be TLV encoded. The structure of this data object is under definition by GlobalPlatform.

Note 6: Tag '66': this data object may contain information about the card and chip implementation, such as the operating system/runtime environment or a security kernel. Such information shall be TLV encoded and may consist of one (or more) OID(s), each OID being introduced by tag '06' and indicating the organization responsible for specifying the operating system, runtime environment or security kernel, and the identification of the corresponding specification and its version number.

Note 7: Tag '67': this may contain information on the certification policies, certificate formats and certificate ids associated with the Issuer Security Domain's Trust Point (TP_ISD), primarily relating to the use of a public key Secure Channel Protocol. Such information shall be TLV encoded and may consist of one (or more) OID(s).

Note 8: Tag '68': this may contain information such as certificate types, formats and ids associated with on-card Security Domains relating to the use of a public key Secure Channel Protocol. Such information shall be TLV encoded and may consist of one (or more) OID(s) .

H.3 Security Domain Management Data

The Security Domain management data may be returned in the SELECT response message within template '73' as described in section 11.9.3.1 - *Data Field Returned in the Response Message.* When present, the Security Domain management data shall be coded as follows.

Tag	Explanation	Length	Value	Presence
'73'	Template	variable	Data objects listed below	Mandatory
'06'	Universal tag for "Object Identifier" (OID)	variable	Identifies Global Platform as the Tag Allocation Authority	Mandatory
'60' '06'	Application tag 0 'OID' tag	variable variable	`{globalPlatform 2 v}` OID for Card Management Type and Version	Optional
'63' '06'	Application tag 3 'OID' tag	variable variable	`{globalPlatform 3}` OID for Card Identification Scheme	Optional
'64' '06'	Application tag 4 'OID' tag	variable variable	`{globalPlatform 4 scp i}` OID for Secure Channel Protocol of the selected Security Domain and its implementation options	Optional
'65'	Application tag 5	variable	Card configuration details	Optional
'66'	Application tag 6	variable	Card / chip details	Optional
'67'	Application tag 7	variable	Security Domain's Trust Point certificate information	Optional
'68'	Application tag 8	variable	Security Domain certificate information	Conditional

Table H-2: Security Domain Management Data

The data objects in this table are equivalent to those in Table H-1, but are specific to this Security Domain and override data objects in Card Recognition Data. See Table H-1 and the notes which follow it for further details of each data object.